ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΕΦΑΡΜΟΣΜΕΝΩΝ ΜΑΘΗΜΑΤΙΚΩΝ ΚΑΙ ΦΥΣΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΟΜΕΑΣ ΦΥΣΙΚΗΣ
ΕΡΓΑΣΤΗΡΙΟ ΠΕΙΡΑΜΑΤΙΚΗΣ ΦΥΣΙΚΗΣ ΥΨΗΛΩΝ ΕΝΕΡΓΕΙΩΝ

ΑΝΑΠΤΥΞΗ ΣΥΣΤΗΜΑΤΟΣ DAQ ΓΙΑ ΤΗΛΕΣΚΟΠΙΟ MICROMEGAS

ΜΕΤΑΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

του

Νικόλαου Δ. Καραστάθη

Επιβλέπων: Γεώργιος Τσιπολίτης
Αν. Καθηγητής

Αθήνα, Ιανουάριος 2012

Εθνικο Μετσοβιο Πολυτεχνειο
ΣΧΟΛΗ ΕΦΑΡΜΟΣΜΕΝΩΝ ΜΑΘΗΜΑΤΙΚΩΝ ΚΑΙ ΦΥΣΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΟΜΕΑΣ ΦΥΣΙΚΗΣ
ΕΡΓΑΣΤΗΡΙΟ ΠΕΙΡΑΜΑΤΙΚΗΣ ΦΥΣΙΚΗΣ ΥΨΗΛΩΝ ΕΝΕΡΓΕΙΩΝ

ΑΝΑΠΤΥΞΗ ΣΥΣΤΗΜΑΤΟΣ DAQ ΓΙΑ ΤΗΛΕΣΚΟΠΙΟ MICROMEGAS

ΜΕΤΑΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

του

Νικόλαου Δ. Καραστάθη

Επιβλέπων: Γεώργιος Τσιπολίτης
Αν. Καθηγητής

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή τον Ιανουάριο του 2012.

.....	.....	.....
-Θεόδωρος Αλεξόπουλος-	-Ευάγγελος Γαζής-	-Τσιπολίτης Γεώργιος-
-Καθηγητής-	-Καθηγητής-	-Αν. Καθηγητής-

Αθήνα, Ιανουάριος 2012

.....
Νικόλαος Δ. Καραστάθης
Πτυχιούχος σχολής Εφαρμοσμένων Μαθηματικών και Φυσικών Επιστημών

Contents

List of Tables

List of Figures

Περίληψη

Ο ανιχνευτής Micromegas διανύει ήδη τη δεύτερη δεκαετία ζωής του και ήδη προσφέρει πολλά υποσχόμενες ιδιότητες (καλή χωρική ακρίβεια, αντοχή στην ακτινοβολία, υψηλός ρυθμός μέτρησης, καλή ενεργειακή διακριτική ικανότητα κ.α.) οι οποίες τον καθιστούν κατάλληλο για ένα εύρος εφαρμογών από την Φυσική Υψηλών Ενεργειών έως τη βιομηχανική παραγωγή. Στη συγκεκριμένη εργασία, παρουσιάζεται ένα νέο λογισμικό για την λήψη δεδομένων από το τηλεσκόπιο Micromegas και τους ελέγχους στους οποίους υποβλήθηκε.

Συγκεκριμένα, στο πρώτο κομμάτι της εργασίας παρουσιάζεται το υλικό που χρησιμοποιήθηκε κατά τον έλεγχο και την δομή του τηλεσκοπίου των έξι ανιχνευτών Micromegas. Αναπτύσσονται οι βασικές πτυχές πίσω από το VMEbus κι αναλύονται τα χαρακτηριστικά των μονάδων για τις οποίες δημιουργήθηκαν βιβλιοθήκες οδηγών (drivers) για τον έλεγχο τους μέσω του ηλεκτρονικού υπολογιστή. Τέλος, παρουσιάζεται η λογική trigger που χρησιμοποιήθηκε κατά το test beam της συνεργασίας RD51.

Το δεύτερο κομμάτι της εργασίας είναι αφιερωμένο στην παρουσίαση του λογισμικού και τα χαρακτηριστικά του. Αναλύονται οι ιδιότητες και οι ευκολίες που περιλαμβάνονται στο γραφικό περιβάλλον καθώς και γίνεται μια αναφορά στις ρουτίνες που απαρτίζουν τον κώδικα ώστε να διευκολυνθεί κάποιος που ενδιαφέρεται να επέμβει σε αυτόν.

Το τελευταίο κομμάτι της εργασίας αποτελεί την επιβεβαίωση ότι το λογισμικό λειτουργεί όπως αναμενόταν. Ελέγχθηκε κατά το test beam της συνεργασίας RD51 τον Οκτώβριο του 2011 και παρουσιάζονται γραφήματα με το προφίλ της δέσμης, όπως αυτά δημιουργήθηκαν από το λογισμικό.

Εν κατακλείδι, μπορούμε να πούμε ότι το λογισμικό που παρουσιάζεται σε αυτή την εργασία αποτελεί μια βελτιωμένη έκδοση του ήδη υπάρχοντος. Λειτουργεί σταθερά και με συχνότητα λήψης κατά πολύ υψηλότερη από το προηγούμενο λογισμικό, ενώ τέλος οι ευκολίες που εισάγει το γραφικό του περιβάλλον είναι κρίσιμες για την χρήση του από έμπειρους κι όχι shifters.

Abstract

The Micromegas detector is already in its second decade of life and already offers many promising qualities (good spatial resolution, radiation hardness, high rate measurements, good energy resolution, etc.) which make it suitable for a range of applications from High Energy Physics to industrial production. In this study, a new data acquisition software is presented which is capable of reading out a Micromegas telescope, as well as the tests in which it was submitted.

In particular, during the first part of this paper the hardware that was used for the tests on the software is presented in addition to the set-up of the telescope. The standards of the VMEbus are unfolded and the specifications of the modules for which special drivers were written are noted. Finally, the trigger logic that was used during the test beam of the RD51 collaboration is also presented.

The second part of this thesis is devoted to present the software that was developed and its characteristics. Its features and the capabilities of the graphical user interface are unrolled and the various routines that are included in the code are presented in case one would like to interfere.

The last part of the paper is the experimental proof that the software works as expected. It was tested during the test beam of the RD51 collaboration in October 2011 and plots of the beam profile are presented.

In conclusion, one could say that the software introduced in this paper is an improved version of the old one. Its operation is stable and its rate is higher than before, while its capabilities make its use easier by expert and novice shifters.

Ευχαριστίες

Ολοκληρώνοντας την πτυχιακή εργασία μου , θα ήθελα να ευχαριστήσω όλους τους δασκάλους και καθηγητές μου όλα αυτά τα χρόνια, για την αγάπη που μου ενέπνευσαν στη μόρφωση και ιδιαιτέρως όλους αυτούς που με καθοδήγησαν και με βοήθησαν καθ΄όλη τη διάρκεια τόσο της πειραματικής διαδικασίας όσο και της συγγραφής της εργασίας.

Κατ' αρχάς, θα ήθελα να ευχαριστήσω ιδιαίτερα τον καθηγητή μου κύριο Γεώργιο Τσιπολίτη, ο οποίος με τη διδασκαλία του κίνησε αρχικά το ενδιαφέρον μου και στη συνέχεια την αγάπη μου για την Πειραματική Φυσική Υψηλών Ενεργειών. Θερμές ευχαριστίες για την ευκαιρία που μου έδωσε να δουλέψω μαζί του σε θέματα σύγχρονης έρευνας, για τις συμβουλές του σε θέματα Φυσικής κι όχι μόνο καθώς και το κίνητρο που συνεχίζει να μου δίνει για να συνεχίσω την προσπάθεια.

Ευχαριστώ τον καθηγητή μου κύριο Θεόδωρο Αλεξόπουλο, ο οποίος όλα αυτά τα χρόνια συνέβαλλε στο να αποκτήσω το μεράκι για την έρευνα, για τις συμβουλές του κατά την ανάλυση των δεδομένων και κυρίως για την υπομονή του στον καταιγισμό ερωτήσεων στον οποίον τον υπέβαλα κατά τις νυχτερινές ώρες εργασίας στη Γενεύη.

Θα ήθελα να ευχαριστήσω ακόμη τον καθηγητή μου κύριο Ευάγγελο Γαζή για την υποστήριξη του, καθώς χωρίς την βοήθεια του τόσο σε θεωρητικά όσο και σε διαδικαστικά θέματα η εργασία αυτή δεν θα είχε ολοκληρωθεί.

Μια ξεχωριστή θέση ανήκει στον συνάδελφο και φίλο Κωνσταντίνο Καρακώστα η βοήθεια του οποίου ήταν κρίσιμη για την ολοκλήρωση της εργασίας.

Τέλος, ένα μεγάλο ευχαριστώ ανήκει στους δικούς μου ανθρώπους, στην οικογένεια μου, που χωρίς την ψυχολογική και υλική υποστήριξη τους όλα αυτά τα χρόνια δε θα μπορούσα να φτάσω έως εδώ, καθώς και στους φίλους μου για τις ώρες ξεγνοιασιάς που περνάμε, οι οποίες αποτελούν πανάκεια για όλα τα προβλήματα που εμφανίζονται στη ζωή μου, ενώ παράλληλα θα ήθελα να τους ζητήσω συγνώμη για τις μέρες που τους παραμελούσα κατά τη διάρκεια της εργασίας αυτής αλλά και σε κάθε εξεταστική περίοδο.

"If I have seen further than others, it is by standing upon the shoulders of giants."
Sir Isaac Newton

CHAPTER 1

Εισαγωγή

Σε αυτό το κεφάλαιο θα αναπτυχθεί το θεωρητικό υπόβαθρο καθώς και τα βασικά στοιχεία του λογισμικού που αναπτύχθηκε από τον συγγραφέα.

1.1 Ο Ανιχνευτής MicroMeGaS

Ο ανιχνευτής MicroMeGaS (MicroMesh Gaseous Structure) ανήκει στην κατηγορία των MicroPattern Gaseous Detectors (MPGD) κι ηταν μια πρόταση των J.Collar και Γ. Γιοματάρη στα μέσα της δεκαετίας του '90. Αρχικά ο ανιχνευτής αποτελούσε μια εναλλακτική πρόταση για την ανίχνευση φωτονίων χαμηλής ενέργειας $(1 - 10keV)$, αλλά με την πάροδο του χρόνου κέρδισε τις εντυπώσεις με τα πλεονεκτήματα που προσφέρει έναντι στους υπόλοιπους ανιχνευτές αερίου. Στη λίστα αυτών των πλεονεκτημάτων συμπεριλαμβάνονται η σταθερότητα, η ταχέα απόκριση, η καλή ενεργειακή και χωρική διακριτική ικανότητα, η υψηλή απόδοση και ακρίβεια του καθώς και η ανθεκτικότητα στην ακτινοβολία που προσφέρει ο συγκεκριμένος ανιχνευτής.

Ο ανιχνευτής είναι μια ασύμμετρη κατασκευή που το μεγαλύτερο πλεονέκτημα του έναντι στους υπόλοιπους ανιχνευτές αερίου, είναι το γεγονός ότι είναι ιδανικός για μετρήσεις μικρής ενέργειας που και πρέπει να έχει ένας παλμός προκειμένου να είναι δυνατή η καταγραφή του από τα ηλεκτρονικά. Η διαφορά του στη γεωμετρία που καθιστά δυνατή την ανίχνευση αυτών των παλμών είναι η διαμέριση του όγκου του ανιχνευτή σε δύο επιμέρους περιοχές, οι οποίες δε διαχωρίζονται πλέον από ένα επίπεδο με σύρματα, αλλά από ένα πλέγμα, γνωστό ως micromesh, που αποτελεί την κάθοδο της ανιχνευτικής διάταξης.

Η αρχή λειτουργίας του micromegas φαίνεται στο σχήμα 1.1. Το πρώτο τμήμα που διασχίζει ένα προσπίπτον ηλεκτρόνιο είναι το ηλεκτρόδιο ολίσθησης (drift electrode). Μετά από αυτό, το σωματίδιο βρίσκεται στην περιοχή μετατροπής (conversion region), η οποία έχει μέγεθος μερικά mm μέχρι το micromesh. Η τάση σε αυτή την περιοχή είναι της τάξης του $1keV/cm$. Στην περιοχή αυτή γίνεται η αρχική αλληλεπίδραση του σωματιδίου με τον ανιχνευτή και δημιουργούνται τα ζεύγη ιόντων-ηλεκτρονίων. Τα ηλεκτρόνια οδηγούνται από το ηλεκτρικό πεδίο να διασχίσουν το micromesh και να συνεχίσουν στην επόμενη περιοχή, ενώ τα ιόντα συλλέγονται από το micromesh.

Η περιοχή ανάμεσα στο micromesh και στην άνοδο καλείται περιοχή ενίσχυσης κι εκτείνεται για περίπου $100\mu m$. Η άνοδος αποτελείται από αγώγιμα μικροστοιχεία (strips) που είναι τυπωμένα πάνω σε μονωτική πλακέτα (Printed Circuit Board - PCB). Κατά την κατασκευή του ανιχνευτή, η πρόκληση είναι να παραμείνει το κενό ανάμεσα στην άνοδο και στο micromesh σταθερό καθ' όλη την επιφάνεια του ανιχνευτή. Για να γίνει κάτι τέτοιο υλοποιήσιμο, χρησιμοποιούνται μικροί μονωτικοί στύλοι (pillars) οι οποίοι εναποθέτονται με φωτογραφικές μεθόδους και καλύπτουν ελάχιστο μέρος του ανιχνευτή ($\approx 1\%$).

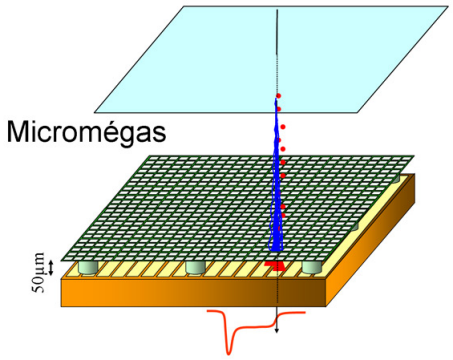


Figure 1.1: Οι περιοχές στις οποίες χωρίζεται ένας ανιχνευτής MicroMeGaS καθώς και το πως περιγράφεται η αλληλεπίδραση του με την προσπίπτουσα δέσμη.

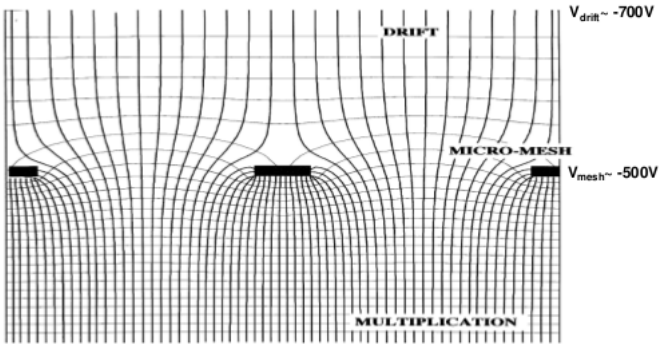


Figure 1.2: Ηλεκτρικό πεδίο γύρω από το micromesh.
Σχηματική αναπαράσταση του ηλεκτρικού πεδίου γύρω από το micromesh. Η πυκνότητα των δυναμικών γραμμών στην περιοχή ενίσχυσης είναι χαρακτηριστική της έντασης του πεδίου στην περιοχή αυτή σε σχέση με την αντίστοιχη στην περιοχή ολίσθησης.

Ο ρόλος του micromesh δεν είναι μόνο να διαχωρίζει την περιοχή μετατροπής από την περιοχή ενίσχυσης (amplification region). Είναι κατασκευασμένο από χαλκό ($5\mu m$) με μια διαδικασία που βασίζεται στην τεχνική της φωτολιθογραφίας κι επιτρέπει την χάραξη ανοιγμάτων $25\mu m$ με βήμα $50\mu m$. Επιπροσθέτως, επιτρέπει την διέλευση των ηλεκτρονίων στην περιοχή ενίσχυσης ενώ παράλληλα αποτρέπει τα θετικά ιόντα που δημιουργούνται κατά την χιονοστιβάδα να κάνουν το ίδιο. Στο πλέγμα εφαρμόζεται τάση της τάξης των $500mV$ ώστε ο λόγος του ηλεκτρικού πεδίου στην περιοχή ενίσχυσης και στην περιοχή μετατροπής να είναι μεγάλος. Όσο μεγαλύτερος είναι ο λόγος, τόσο μεγαλύτερος είναι κι ο αριθμός των ηλεκτρονίων που εισέρχονται στη περιοχή ενίσχυσης. Η ύπαρξη μικρού λόγου καθιστά το πλέγμα μη διαπερατό μειώνοντας έτσι την διαφάνεια (transparency) του και κατ' επέκταση του ανιχνευτή. Στην περιοχή ενίσχυσης, η ηλεκτρονιακή χιονοστιβάδα ξεκινάει με μεγάλη ευκολία καθώς το χωρικό κενό ανάμεσα στο πλέγμα και στην άνοδο είναι πολύ μικρό, ενώ ταυτόχρονα το ηλεκτρικό πεδίο είναι αρκετά μεγάλο (έως της τάξης των $40kV/cm$).

Τα ηλεκτρόνια που διέσχισαν το micromesh περνώντας στην περιοχή ενίσχυσης συνεχίζουν προς την άνοδο. Η άνοδος αποτελείται από έναν αριθμό (συνήθως 96) χάλκινων strips με τυπικό μέγεθος $150nm \times 100\mu m$ που γειώνονται, με τη βοήθεια ενός προενισχυτή με χαμηλό θόρυβο και υψηλή ενίσχυση, σε μια μια μονωτική στρώση.

Ένα από τα σημαντικότερα στοιχεία του ανιχνευτή για να επιτευχθεί η βέλτιστη απόδοση είναι η γνώση της

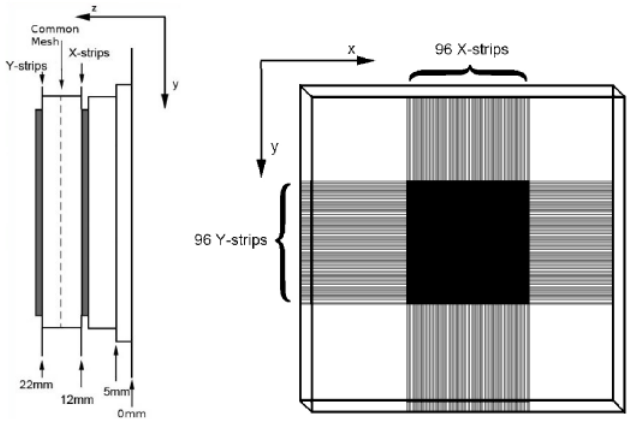


Figure 1.3: Γραφική αναπαράσταση ενός σταθμού του τηλεσκοπίου (αριστερά) και της ενεργού περιοχής του κάθε σταθμού (δεξιά).

μορφής του ηλεκτρικού πεδίου κοντά στο πλέγμα. Το ομογενές πεδίο που επιθυμείται για την περιοχή ενίσχυσης είναι εύκολα υλοποιήσιμο παρά την μικρή διάσταση της περιοχής αυτής. Συνεπώς, είναι αναμενόμενη η εύκολη δημιουργία μιας ηλεκτρονιακής χιονοστιβάδας. Το πρόβλημα έγκειται στη μορφή του ηλεκτρικού πεδίου γύρω από τις εγκοπές του πλέγματος, όπου γνωρίζουμε ότι παραμορφώνεται και χάνει την ομοιογένεια του. Η γνώση του βαθμού αυτής της παραμόρφωσης κι εν γένει της μορφής του πεδίου στις ειδικές αυτές περιοχές είναι απαραίτητη για την επίτευξη της απόδοσης και της διαφάνειας του ανιχνευτή. Οι δυναμικές γραμμές που ξεκινούν από το ηλεκτρόδιο ολίσθησης καταλήγουν περνώντας μέσα από το πλέγμα στα strips. Όπως όμως είναι εμφανές και στο σχήμα ?? η πυκνότητα τους μεταβάλλεται άρδην μετά την διέλευση τους από το micromesh. Αυτό αναδεικνύει την διαφορά της έντασης των ηλεκτρικών πεδίων στις δύο αυτές περιοχές. Η ενίσχυση κατά τη διέλευση από το πλέγμα είναι αμελητέα σε σχέση με την συνολική και γι' αυτό δε λαμβάνεται υπ' όψιν, το οποίο αποτελεί και μια ειδοποιό διαφορά ανάμεσα στον micromegas και τους υπόλοιπους MSGD. Τέλος, σε μερικές περιπτώσεις είναι βολικό να αυξηθεί το ηλεκτρικό πεδίο στην περιοχή μετατροπής με σκοπό να παραχθεί κάποια προενίσχυση.

Το γεγονός ότι η απόσταση ανάμεσα στα διάκενα του πλέγματος είναι μικρή, η χρονική διασπορά κατά μήκος της τροχιάς του σωματιδίου είναι πολύ μικρή. Συνεπώς, ο micromegas προσφέρει εξαιρετική χρονική διακριτική ικανότητα. Επιπλέον, σε μια μη κάθετη στην ενεργό περιοχή τροχιά, η ανάπτυξη της χιονοστιβάδας είναι κάθετη προς την άνοδο κι έτσι το σήμα επάγεται σε μια μικρή περιοχή των καναλιών ανάγνωσης, με αποτέλεσμα η ανιχνευτική διάταξη να προσφέρει εξίσου εξαιρετική χωρική διακριτική ικανότητα. Το χαρακτηριστικό αυτό, της μικρής περιοχής συγκέντρωσης των ηλεκτρονίων δίνει την δυνατότητα στον micromegas να λειτουργήσει ως θάλαμος χρονικής προβολής (Time Projection Chamber, TPC). Με γνωστή την πληροφορία των strips που έδωσαν σήμα, καθώς και τον χρόνο στον οποίο δόθηκε το σήμα, είναι απλός ο καθορισμός της τροχιάς ενός σωματιδίου μέσα στον θάλαμο. Τέλος, η υψηλή ενίσχυση που δύναται να επιτευχθεί είναι χαρακτηριστική ιδιότητα όλων των MPGD.

1.2 Το Τηλεσκόπιο MicroMeGaS του RD51

Η συνεργασία RD51 στοχεύει στην ανάπτυξη τεχνολογιών για τους ανιχνευτές αερίου, τα ηλεκτρονικά συστήματα που απαιτούνται για τη λειτουργία τους και τη λήψη των δεδομένων από τους ανιχνευτές καθώς και τις εφαρμογές που δύναται να έχουν οι ανιχνευτές αερίου στη βασική και την εφαρμοσμένη έρευνα.

Το τηλεσκόπιο MicroMeGaS αποτελείται από τρεις όμοιους σταθμούς με διπλούς ανιχνευτές MicroMeGaS. Κάθε σταθμός αποτελείται από δύο ανιχνευτές από τους οποίους ο ένας διαβάζει στην οριζόντια συνιστώσα κι ο άλλος στην κάθετη. Το πλήρες τηλεσκόπιο με τους έξι ανιχνευτές τοποθετείται σε ένα ειδικά κατασκευασμένο

πλαίσιο στο οποίο η αντίσταση μεταξύ του πρώτου και του δεύτερου σταθμού είναι $338mm$ και ανάμεσα στον δεύτερο και τον τρίτο $403mm$. Σε κάθε σταθμό οι δυο ανιχνευτές που τον αποτελούν τοποθετούνται ο ένας συνεχόμενα από τον άλλον με τέτοιο τρόπο ώστε να μοιράζεται σε κάθε σταθμό ένα επίπεδο ολίσθησης, ενώ παράλληλα ο κάθε ανιχνευτής να έχει το δικό του επίπεδο mesh. Με αυτή την συνδεσμολογία ο κάθε σταθμός απαιτεί τρία κανάλια τροφοδορίας ηλεκτρικής τάσης. Επιπρόσθετα, κάθε σταθμός αποτελείται από δύο PCB με διαστάσεις $10 \times 10cm$, η κάθε μια για τις ανάγκες ανάγνωσης της κάθε κατεύθυνσης. Κάθε PCB αποτελείται από 96 strips τα οποία απέχουν $250\mu m$ μεταξύ τους δημιουργώντας έτσι μια ενεργή περιοχή $2.4 \times 10cm$ για κάθε PCB και $2.4 \times 2.4cm$ για κάθε σταθμό. Κατά την διαδρομή του, ένα σωματίδιο που διασχίζει τον κάθε σταθμό διασχίζει πρώτα τον ανιχνευτή που διαβάζει το οριζόντιο άξονα κι έπειτα τον ανιχνευτή που διαβάζει τον κάθετο.

1.3 Πειραματική Διάταξη

Σε αυτή την παράγραφο περιγράφεται εν συντομία η γενικότερη πειραματική διάταξη για την οποία μπορεί να λειτουργήσει το λογισμικό, σε ότι αφορά το υλικό.

1.3.1 VMEbus Crate

Το υλικό που διαχειρίζεται το λειτουργικό φιλοξενείται σε μια μονάδα πρωτοκόλλου VMEbus του οποίου η αρχιτεκτονική βασίζεται στο VERSAbus που αναπτύχθηκε από την Motorola στα τέλη του 1970. Το πρωτόκολλο αυτό χαρακτηρίζει την επικοινωνία μεταξύ των μικροεπεξεργαστών, των μονάδων αποθήκευσης και τα περιφερειακά συστήματα ελέγχου με έναν πεπλεγμένο μεταξύ τους τρόπο. Το όλο σύστημα έχει αναπτυχθεί με τέτοιο τρόπο ώστε

- Να επιτρέπει την επικοινωνία μεταξύ των συσκευών του VMEbus crate χωρίς να διαταράσονται οι εσωτερικές διαδικασίες της κάθε συσκευής.
- Να παγκοσμιοποιηθούν οι ηλεκτρονικές και μηχανικές παράμετροι που απαιτούνται για την δημιουργία συσκευών που θα λειτουργούν απροβλημάτιστα σε αρμονία με άλλες συσκευές του ίδιου πρωτοκόλλου.
- Να παρέχει ορολογία και ορισμούς που περιγράφουν το πρωτόκολλο.
- Να παρέχει ένα σύστημα όπου η συνολική απόδοση είναι ανάλογη από τις επιμέρους συσκευές κι όχι από το σύστημα επικοινωνίας.

Ένα συμβατικό VMEbus crate κατασκευάζεται σε ένα subrack. Το τελευταίο είναι ένα σταθερό πλαίσιο που παρέχει τη μηχανική υποστήριξη στις κάρτες που εισάγωνται στο σύστημα και διαφυλάσει την επιτυχία των συνδέσεων καθώς και τη ροή αέρα για την ψύξη του συνολικού συστήματος.

1.3.2 CAEN V2718 VME-PCI Optical Link Bridge

Η κάρτα V2718 είναι μια μονάδα master που συνδέεται μέσω του CONET (Chainable Optical NETwork) με έναν ηλεκτρονικό υπολογιστή μέσα από τον οποίο δίνονται οι εντολές που θα εκτελέσει. Ο ηλεκτρονικός υπολογιστής με τον οποίο θα συνδεθεί η μονάδα θα πρέπει να είναι εξοπλισμένος με την κάρτα PCI A2818 η οποία είναι μια 32bit 33MHz PCI κάρτα. Η επικοινωνία μεταξύ υπολογιστικού συστήματος και κάρτας γίνεται μέσω οπτικής ίνας. Έως οκτώ V2718 μπορούν να ελεγχθούν από μια A2818 κάρτα. Η μονάδα V2718 δύναται να εκτελέσει όλους τους κύκλους που προβλέπονται από το πρωτόκολλο VME64X.

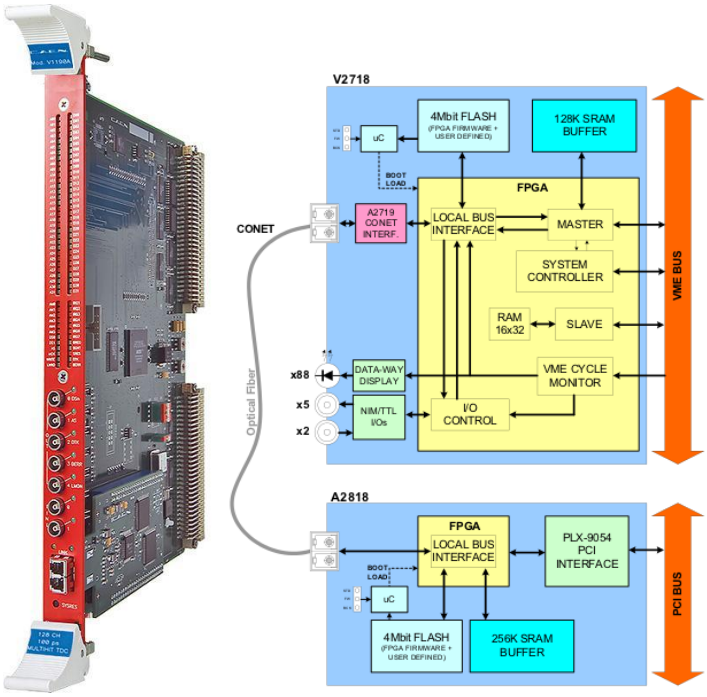


Figure 1.4: Μπροστινή όψη και το διάγραμμα της μονάδας V2718.

1.3.3 CAEN V551B C-RAMS Sequencer

Η μονάδα V551B είναι η μονάδα VME που χειρίζεται την λήψη των δεδομένων από ένα chip που λαμβάνει δεδομένα με τη μέθοδο του multiplexing. Η μονάδα μπορεί να χειρηστεί την οικογένεια VA των chips (όπως αυτά παράχθηκαν από την IDE AS, Όσλο) αλλά λόγω της πλαστικότητας του μπορεί να λειτουργήσει εξίσου καλά με παρόμοια chips (Amplex, Gasplex κλπ).

Η μονάδα V551B κατασκευάστηκε ώστε να ελέγχει τα σήματα προς και από τις κάρτες C-RAMS (CAEN Readout for Analog Multiplexing Signals) V550 που εκτελούν την αποπλεγματοποίηση της πληροφορίας από τα chips λήψης και την μετατροπή του σήματος σε ψηφιακή μορφή. Μια μονάδα V551B μπορεί να ελέξει έως 19 C-RAMS, το οποίο σημαίνει την δυνατότητα ελέγχου 76608 καναλιών.

Για να ολοκληρωθεί ένας κύκλος λήψης δεδομένων ο χρήστης θα πρέπει να προβεί σε μια σειρά από ρυθμίσεις και ύστερα να ξεκινήσει ο κύκλος. Το παρακάτω περιγράφει τις ρυθμίσεις που θα πρέπει να γίνουν και τη σειρά της διαδικασίας λήψης.

Ο κύκλος λήψης δεδομένων ξεκινάει από ένα εξωτερικό ή VME σήμα `TRIGGER`. Ύστερα, η γραμμή `BUSY` ενεργοποιείται δείχνοντας πως η μονάδα δε μπορεί να δεχτεί άλλο σήμα `TRIGGER`. Το σήμα `TRIGGER` ξεκινάει έναν κύκλο εσωτερικών διαδικασιών. Μετά από χρόνο t_1 από το σήμα αυτό ενεργοποιούνται τα σήματα `HOLD` και `SHIFT IN`. Όταν το σήμα `HOLD` αναγνωρίζεται ξεκινάει ένας κύκλος από σήματα `CLOCK` μετά από μια καθυστέρηση χρόνου t_2. Το σήμα `CLOCK` παράγεται από έναν εσωτερικό ταλαντωτή συχνότητας $50MHz$ και μετέπειτα ξεκινάει ο κύριος κύκλος λήψης με μια καθυστέρηση περίπου $\pm 10ns$ (jitter). Εν συνεχεία, N στον αριθμό παλμοί `CLOCK` και `CONVERT` παράγονται. Ο αριθμός αυτός αντιστοιχεί στον αριθμό των καναλιών του ανιχνευτή ο οποίος τίθεται μεταξύ 1 και 2047. Κάθε παλμός `CLOCK` ακολουθείται από έναν παλμό `CONVERT` με μια καθυστέρηση $t_5 ns$, που προγραμματίζεται μέσω VME. Ο σκοπός της καθυστέρησης αυτής είναι ώστε να αναμένεται η αναγνώρηση του αναλογικού σήματος το οποίο προέρχεται από τον multiplexer. Το πλάτος της ενεργής φάσης των παλμών `CLOCK` και `CONVERT` είναι $t_3 ns$ που ρυθμίζεται μέσω του VME me b;hma $20ns$ στο εύρος από $20ns$ έως $5\mu s$ περίπου.

Η περίοδος επανάληψης t_4 των παλμών `CLOCK` και `CONVERT` ρυθμίζεται μέσω του VME σε βήματα των $20ns$

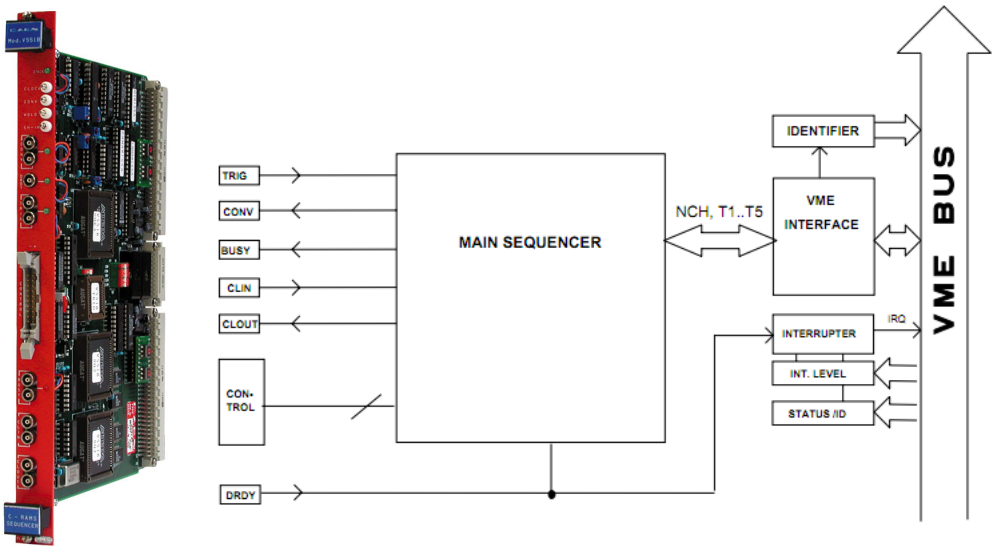


Figure 1.5: Μπροστινή όψη και το διάγραμμα της μονάδας V551.

στο εύρος από $40ns$ έως $10\mu s$.

Τα σήματα ελέγχου `CLOCK`, `CONVERT`, `HOLD` και `SHIFT IN` είναι διαθέσιμα σε τέσσερα σημεία ελέγχου τοποθετημένα στο μπροστινό πάνελ.

Με τον τελευταίο παλμό `CONVERT`, η γραμμή `DRESET` του VME γίνεται ενεργή για $1\mu s$ στον οποίο επανεκινείται ο κύκλος. Επιπλέον, το σήμα `BUSY` απενεργοποιείται αν καμία από τις κάρτες V550 δεν έχει παράξει το σήμα `DRDY`, δηλώνοντας πως υπάρχουν δεδομένα για λήψη. Τέλος, υπάρχει ένα σήμα `VETO` τοπ οποίο ελέγχεται από το λογισμικό και αναγκάζει την κάρτα V551B να εισέλθει σε κατάσταση `BUSY`.

1.3.4 CAEN V550 C-RAMS

Η μονάδα V550 C-RAMS είναι μια μονάδα VME που αποτελείται από δύο κανάλια μετατροπής αναλογικής πληροφορίας σε ψηφιακή και χρησιμοποιείται για την λήψη δεδομένων όταν αυτά είναι σε πεπλεγμένη μορφή. Κάθε κανάλι της μονάδας δέχεται θετικά κι αρνητικά σήματα εισόδου τα οποία ενισχύονται και τροφοδοτούνται στη μονάδα μετατροπής. Η ευαισθησία (mV/bit) μπορεί να καθοριστεί ανάμεσα σε τέσσερεις διαφορετικούς λόγους με αναλογίες 1,2,5 και 10 με χρήση εσωτερικών διακοπτών. Η μονάδα μπορεί να μετατρέψει το αναλογικό σε ψηφιακό σήμα με ρυθμό έως και 5MHz, δέχεται διαφορική είσοδο με επιλεγμένη ενίσχυση, έχει 12bit γραμμική μετατροπή και επιτρέπει τη χρήση των μεθόδων zero suppression κι αφαίρεσης pedestal, ενώ τέλος περιλαμβάνει εφαρμογές διαγνωστικών εργασιών και τη δυνατότητα αυτοελέγχου.

Με την ύπαρξη ενός εξωτερικού `CONVERT` σήματο, το σήμα που εισάγεται δειγματίζεται από έναν ADC και η ψηφιακή του τιμή συγκρίνεται με μια τιμή κατωφλιού. Αν το σήμα είναι μεγαλύτερο από το κατώφλι, αφαιρείται το pedestal και το αποτέλεσμα αποθηκεύεται στην προσωρινή μνήμη της μονάδας με την λογική FIFO. Γι' αυτό τον σκοπό κάθε ένα από τα δύο κανάλια της μονάδας αποτελείται από μνήμη για την αποθήκευση των τιμών κατωφλιού και pedestal.

Ο αριθμός N των καναλιών που μπορεί να διαβαστεί από τη μονάδα μπορεί να προγραμματιστεί ανάμεσα σε 32 και 2016 με βήμα 32 καναλιών. Στο τέλος του κύκλου μετατροπής (ύστερα δηλαδή από παλμούς `CONVERT`) κι αφού και η τελευταία λέξη αποθηκευτεί στη μνήμη, αν η μνήμη περιέχει έστω και μια λέξη τότε η μονάδα τίθεται σε κατάσταση Data Ready δηλώνοντας πως τα δεδομένα θα πρέπει να διαβαστούν μέσω του VME.

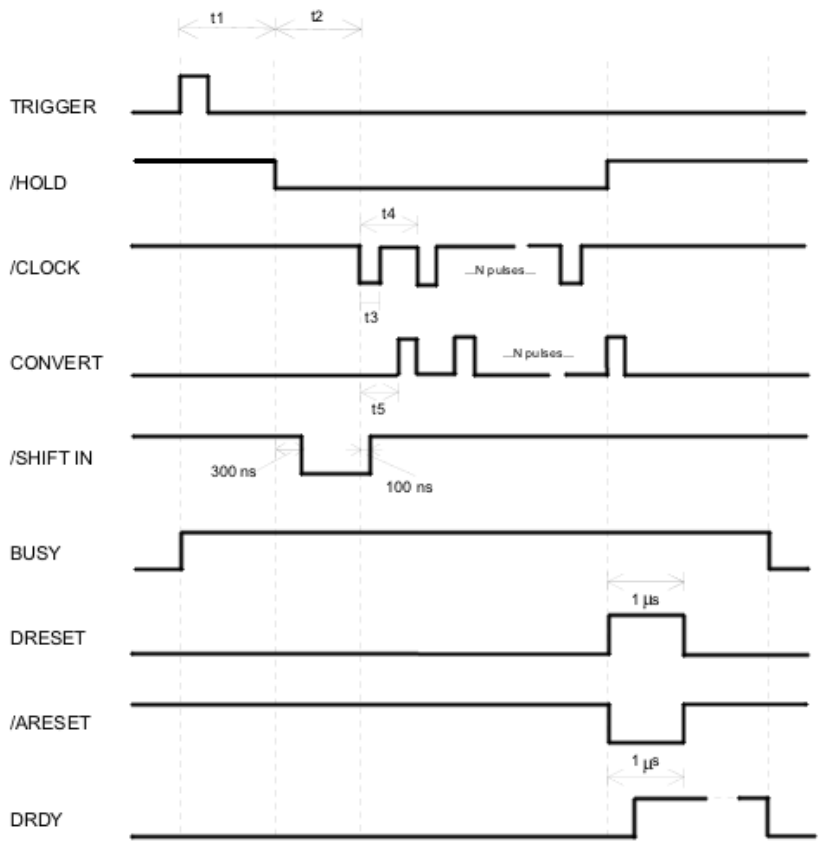


Figure 1.6: Mod. V551B.

1.3.5 CAEN V462 Dual Gate Generator

Η μονάδα V462 αποτελείται από δύο ανεξάρτητους gate generators ικανούς να λειτουργούν ξεχωριστά στο εύρος των $100ns$ έως $9.999999s$. Παράγουν τρία σήματα NIM: την πύλη (Gate), το σήμα αρχής (Begin Marker) πλάτους $100ns$ και το σήμα τέλους (End Marker) πλάτους $100ns$ στο τέλος του παλμού πύλης.

Η μονάδα βασίζεται στην λειτουργία των Programmable Gate Arrays που περιέχουν σχεδόν όλη τη λογική για καθεμια από τις λειτουργίες της γεννήτριας. Υπάρχουν δύο registers ανά κανάλι: ένα για τον τοπικό τρόπο (local mode) κι ένα για τον τρόπο VME (VME mode).

Στον τοπικό τρόπο, το περιεχόμενο των registers μεταφέρεται σε οκτώ 4bit multiplexers που επιτρέπουν την εμφάνηση του πλάτους της πύλης και παρουσιάζεται σε μερικούς συγκριτές bit-to-bit. Η μονάδα περιέχει οκτώ 4bit BDC μετρητές που τοποθετούνται στο μηδέν όταν η πύλη δεν έχει σήμα trigger. Όταν το σήμα Start ενεργοποιείται οι μετρητές αρχίζουν να μετρούν σε μια συχνότητα 10MHz και το αποτέλεσμα περνιέται στην άλλη είσοδο των συγκριτών. Όταν η 32bit λέξη από τους μετρητές φτάσει σε ένα από τα επιλεγμένα registers το μέτρημα σταματάει και το σήμα πύλης τελειώνει. Ένας νέος κύκλος είναι δυνατός αμέσως μετά το σήμα End Marker.

Ο τοπικός τρόπος (Local Mode) χαρακτηρίζεται από την παρουσίαση του περιεχομένου στο τοπικό register που γίνεται το register που θα συγκριθεί με τους μετρητές. Επιπλέον, σε αυτό τον τρόπο ο χρήστης μπορεί να τροποποιήσει το περιεχόμενο των registers μέσω τεσσάρων κουμπιών και να θέσει ένα διαφορετικό πλάτος στον παλμό πύλης. Κάθε ψηφίο που επιλέγεται μέσω των κουμπιών `DIG SEL` $+/-$ ξεκινάει να αναβοσβήνει

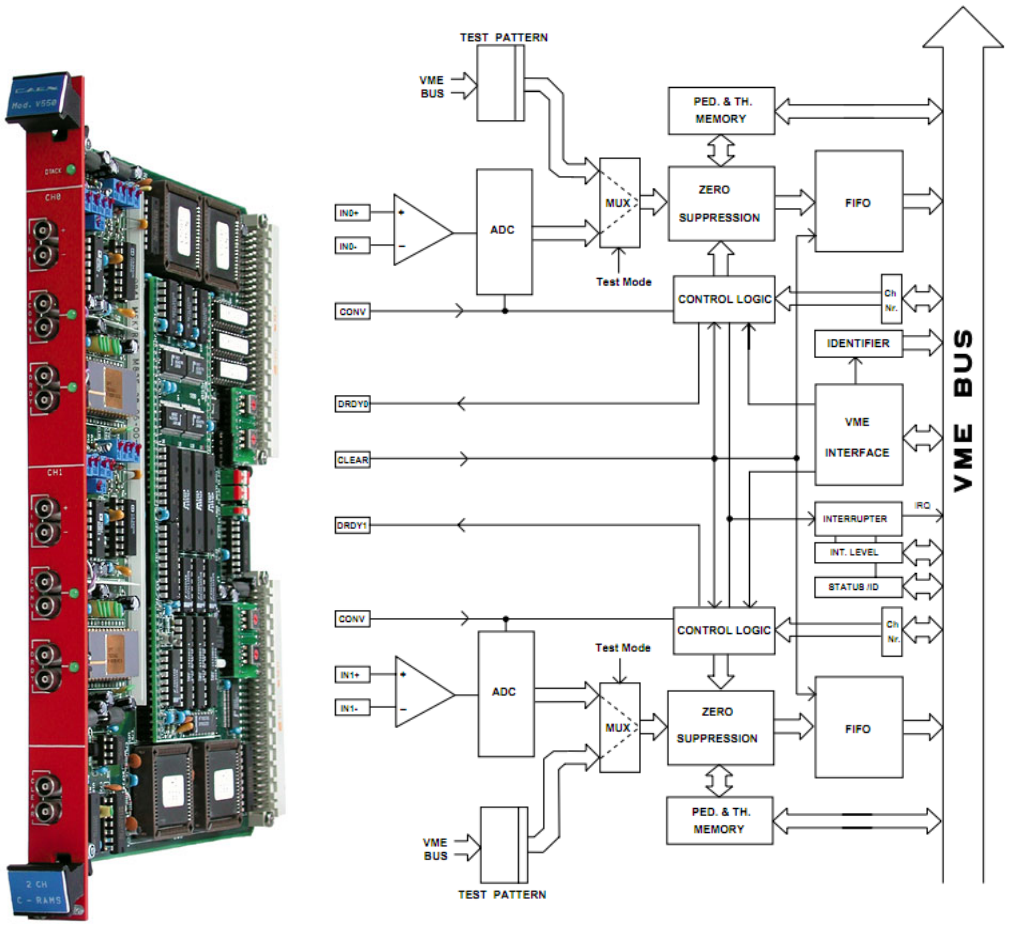


Figure 1.7: Μπροστινή όψη και το διάγραμμα της μονάδας V550.

και η τιμή του μπορεί να αλλαχθεί μέσω των κουμπιών `FIG SEL` $+/-$.

Ο τρόπος VME (VME mode) χαρακτηρίζεται από την παρουσίαση του περιεχομένου στο register του VMEπου γίνεται το register που θα συγκριθεί με τους μετρητές. Το trigger της πύλης και το πλάτος του παλμού καθορίζονται μέσω του VME και τα κουμπιά δεν χρησιμοποιούνται όταν έχει επιλεχθεί ο συγκεκριμένος τρόπος.

Οποιοσδήποτε και να είναι ο τρόπος λειτουργίας που έχει επιλεχθεί η πύλη πάντα κάνει trigger από ένα σήμα NIM ή από το πάτημα ενός `START` κουμπιού για κάθα κανάλι. Όταν υπάρχει το σήμα πύλης και για όλη του τη διάρκεια όλα τα άλλα σήματα εκκίνησης αγνοούνται.

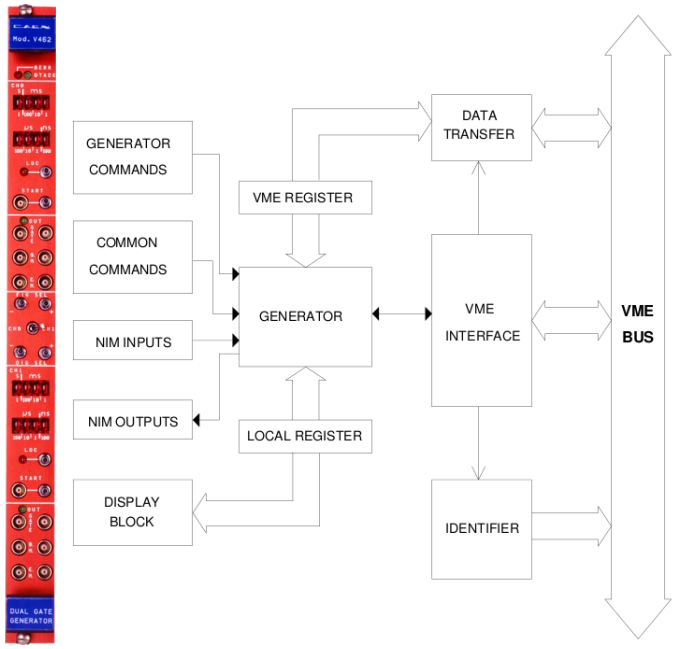


Figure 1.8: Μπροστινή όψη και Block Diagram του CAEN Mod. V462 Dual Gate Generator.

1.4 Λογισμικό DAQ

Για την κατασκευή του λογισμικού για την ανάγνωση των δεδομένων από το τηλεσκόπιο MicroMeGaS χρησιμοποιήθηκε η βιβλιοθήκη που προσφέρει η CAEN για το υλικό της, η γλώσσα προγραμματισμού Qt4 για τη δημιουργία του γραφικού περιβάλλοντος και η $C++$ για τον κώδικα της λήψης δεδομένων. Τέλος, ως εφαρμογή εμφάνισης διαγραμμάτων χρησιμοποιήθηκε η εργαλειοθήκη ROOT.

1.4.1 Γραφικό Περιβάλλον

Για το λογισμικό που αναπτύχθηκε και παρουσιάζεται σε αυτή την εργασία η Qt όπως αναφέρθηκε χρησιμοποιείται για την δημιουργία κι ορθή λειτουργία του γραφικού περιβάλλοντος. Η προγραμματιστική κλάση που είναι υπεύθυνη για το γραφικό περιβάλλον (GUI) καλείται Interface και ο κώδικας της μπορεί να βρεθεί στον υποκατάλογο `src/` του λογισμικού.

Το κεντρικό παράθυρο του GUI αποτελείται από ένα QTabWidget με τέσσερις tabs (Display, Configuration, Log Viewer, About) κι από ένα σύνολο από QPushButtons που σηματοδοτούν μια ποικιλία από λειτουργίες που αφορούν την επικοινωνία του λογισμικού με το υλικό και τον κύκλο λήψης δεδομένων. Οπουδήποτε στο γραφικό περιβάλλον εμφανίζεται κάποιο γράμμα από το όνομα ενός widget να είναι υπογραμμισμένο, δηλώνει πως το γράμμα αυτό χρησιμοποιείται για την συντόμευση ενεργοποίησης του συγκεκριμένου widget. Το πλήκτρο που χρησιμοποιείται για τις συντομεύσεις είναι το `ALT`. Συνεπώς, για παράδειγμα, το κουμπί Initialize έχει υπογραμμισμένο το γράμμα `i` το οποίο σημαίνει ότι μπορεί να ενεργοποιηθεί μέσω του πληκτρολογίου αν πατηθούν τα πλήκτρα `ALT+i`.

Καρτέλα Display

Κάτω από την καρτέλα Display υπάρχει ένας καμβάς ROOT στον οποίο εμφανίζεται ένας αριθμο ιστογραμμάτων που είναι ίσος με τον αριθμό των ανιχνευτών. Τα ιστογράμματα αυτά αποτελούν hitmaps στους αντίστοιχους ανιχνευτές. Για κάθε κύκλο λήψης δεδομένων ο χρήστης επιλέγει έναν αριθμό ο οποίος υποδεικνύει με ποιο

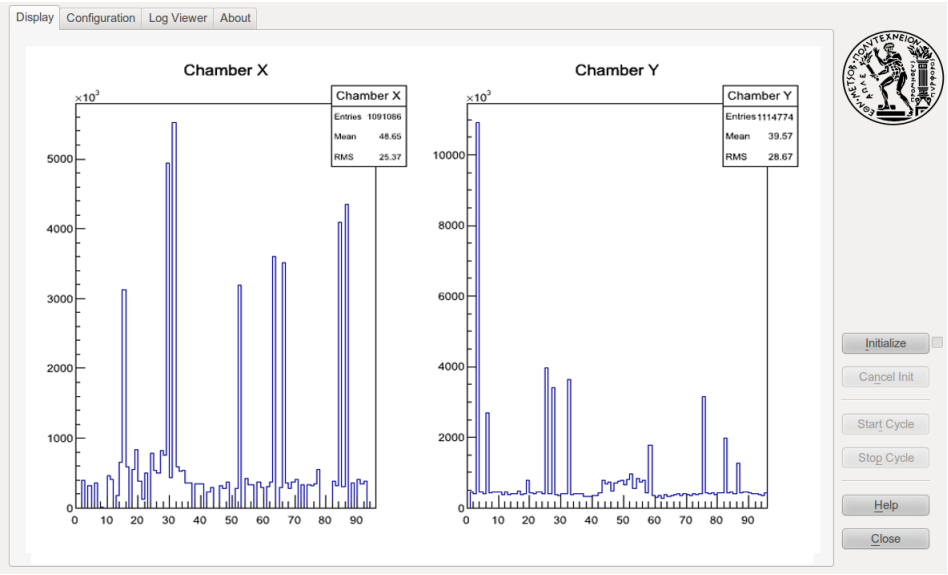


Figure 1.9: Στιγμιότυπο της καρτέλας Display.

ρυθμό θα ανανεώνονται τα ιστογράμματα σε μονάδες γεγονότων. Ο αριθμός αυτός καλείται `Display Refresh Rate` κι αν ο χρήστης τον θέσει ίσο με 1000, τότε αυτό σημαίνει πως ο καμβάς θα ανανεώνεται κάθε 1000 γεγονότα.

Ο καμβάς χωρίζεται δυναμικά σε έναν αριθμό από pads ίσο με το διπλάσιο του αριθμού των C-RAMS που υπάρχουν στη διάταξη. Αυτό γίνεται διότι κάθε C-RAMS αποτελειται από δύο FIFO και συνεπώς ο αριθμός των ανιχνευτών θα είναι ίσος με τον αριθμό των FIFO. Ο τίτλος του κάθε ιστογράματος επίσης τίθεται δυναμικά σε σχέση με την αρίθμιση των FIFO. Για παράδειγμα ο ανιχνευτής που θα διαβάζεται από το δεύτερο FIFO της τρίτης C-RAMS θα παρουσιάζεται με ένα ιστόγραμμα με τίτλο `Chamber 6`.

Καρτέλα Configuration

Μεγάλη προσοχή έχει δοθεί στην κατασκευή ενός απλού περιβάλλοντος για την ρύθμιση του λογισμικού στις ανάγκες της κάθε πειραματικής διάταξης κι ανάλογα με τις διαθέσιμες μονάδες ανάγνωσης.

Η καρτέλα των ρυθμίσεων χωρίζεται σε επτά μέρη. Ξεκινώντας από το πάνω αριστερά κομμάτι, ο χρήστης μπορεί να ρυθμίσει τη διεύθυνση και το χρονισμό του Sequencer. Ακριβώς, δίπλα σε αυτό είναι το κομμάτι ρύθμισης των C-RAMS όπου ο χρήστης εισάγει τον αριθμό και τις διευθύνσεις των μονάδων. Στη συνέχεια, υπάρχει το κομμάτι του Gate Generator, στο οποίο με τη χρήση ενός checkbox ο χρήστης δηλώνει αν υπάρχει ή όχι μια τέτοια μονάδα στη διάταξη.

Από κάτω, υπάρχει το κομμάτι ρύθμισης των readout markers. Αυτές είναι συγκεκριμένες λέξεις των 32bit που υποδεικνύουν την έναρξη και το τέλος κάθε κομματιού στο αρχείο που εξάγει το πρόγραμμα. Ύστερα, υπάρχει το κομμάτι όπου ρυθμίζεται ο αριθμός του run και ο τύπος του. Τέλος, υπάρχει το κομμάτι που περιέχει γενικές ρυθμίσεις όπως ο αριθμός των καναλιών ανάγνωσης, ο μέγιστος αριθμός γεγονότων που θα συλλεχθούν για το συγκεκριμένο run, η χρονική καθυστέρηση στην αναμονή του interrupt σήματος και τα paths των αρχείων που χρειάζονται για τη ρύθμιση του προγράμματος.

Καρτέλα LogViewer

Η καρτέλα καταγραφής γεγονότων κρατάει πληροφορίες κατά τον κύκλο λήψης δεδομένων σχετικές με το ίδιο εμφανίζοντας είτε πληροφορίες, είτε σφάλματα που τυχόν συμβαίνουν κατά τη λειτουργία του λογισμικού. Κάθε γραμμή χαρακτηρίζεται από την χρονική πληροφορία και έναν δείκτη δείχνοντας πόσο σημαντική για την ορθή

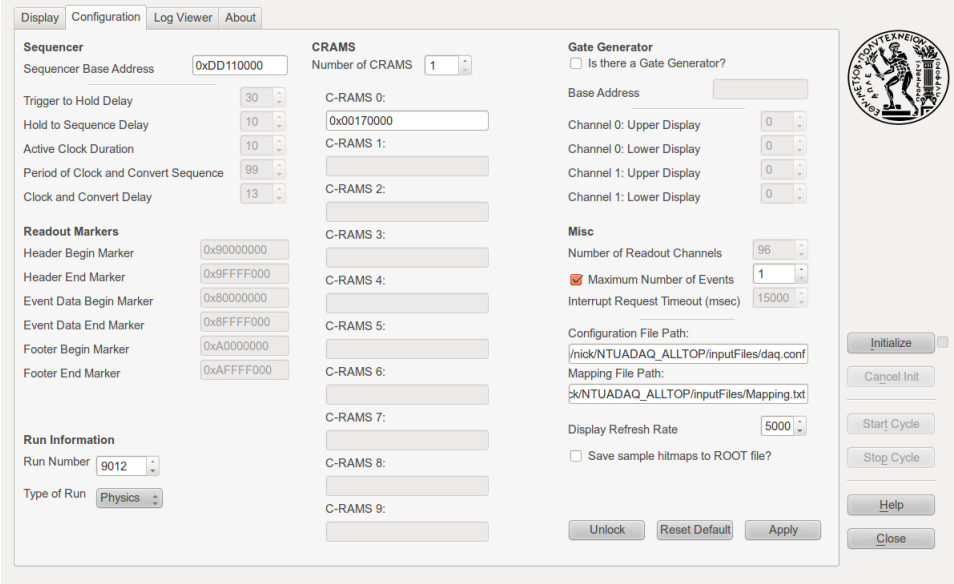


Figure 1.10: Στιγμιότυπο από την καρτέλα ρυθμίσεων.

λειτουργία του προγράμματος είναι η πληροφορία που εισάγεται. Υπάρχουν τριών ειδών δείκτες: ο δείκτης `INFO` που χαρακτηρίζει τις πληροφορίες σχετικά με τον κύκλο, ο δείκτης `WARNING` που αποτελεί ένα σφάλμα κατά τη λειτουργεία του λογισμικό το οποίο μπορεί να μην είναι κρίσιμο για τον συνολικό κύκλο κι ο δείκτης `FATAL` που χαρακτιρίζει κρίσιμα σφάλματα τα οποία δηλώνουν είτε σφάλμα σύνδεσης με το υλικό είτε κάποιο πρόβλημα στο αρχείο που παράγεται.

Για παράδειγμα, μια πληροφορία μπορεί να είναι

```
[INFO] Configuration locked for non-expert users.
```

ενώ ένα μη κρίσιμο σφάλμα

```
[WARNING] Error while generating Event Data Bottom Header. Possibility of corrupted file.
```

Τέλος, ένα κρίσιμο σφάλμα είναι

```
[FATAL] Error while initializing controller.
```

Επιπλέον, η καρτέλα αυτή περιέχει κι ένα κουμπί `Clear` που καθαρίζει την ιστορία του ημερολογίου.

Καρτέλα About

Η καρτέλα αυτή περιέχει πληροφορίες για τον δημιουργό του λογισμικού και την έκδοση του.

Πλήκτρα Κύκλου

Τα πλήκτρα του κύκλου λήψης δεδομένων είναι τα έξι πλήκτρα στο δεξί κομμάτι του γραφικού περιβάλλοντος.

- **Initialize:** Όταν το συγκεκριμένο πλήκτρο πατηθεί τότε πραγματοποιείται η σύνδεση μεταξύ του λογισμικού και του υλικού, τα αρχεία εισόδου διαβάζινται και τα αρχεία εξόδου δημιουργούνται και γράφεται σε αυτά το πρώτο κομμάτι τους. Αν η αρχικοποίηση είναι επιτυχής ένα <<τικ>> εισάγεται στο κουτί ελέγχου δίπλα στο κουμπί, εισάγεται μια γραμμή στο log και το κουμπί `Start Cycle` γίνεται πλέον ενεργό. Αν η αρχικοποίηση δεν είναι επιτυχής το κουτί ελέγχου μένει αμετάβλητο και ένα μήνυμα σφάλματος εμφανίζεται στο log. Υπάρχει τέλος και μια συντόμευση από το πληκτρολόγιο που είναι `ALT+I`.

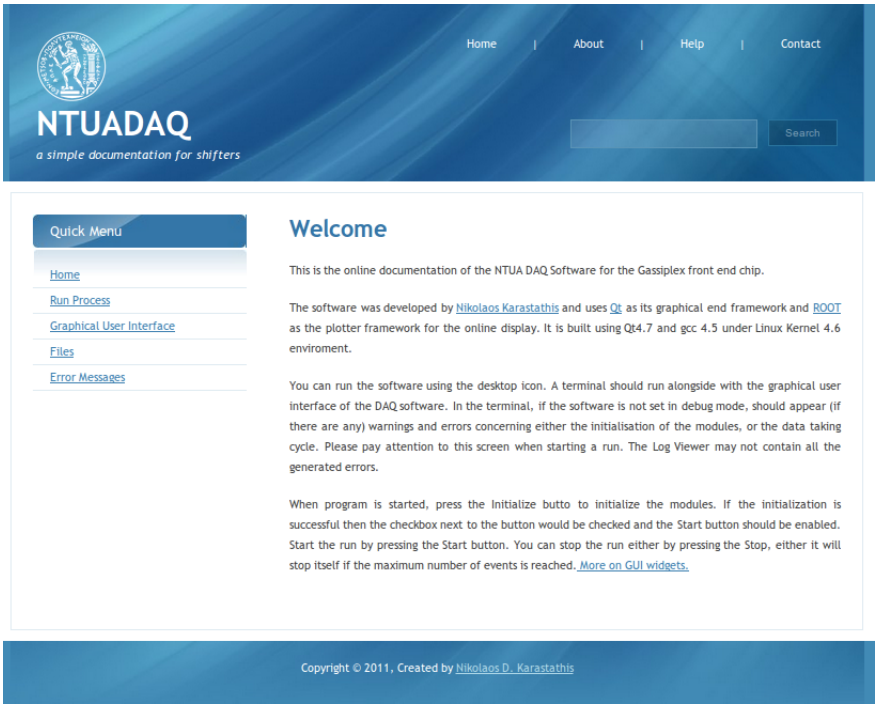


Figure 1.11: Στιγμιότυπο της ιστοσελίδας με την τεκμηρίωση του λογισμικού.

- **Cancel Init.:** Αν για οποιοδήποτε λόγο ο χρήστης θέλει να ακυρώσει την αρχικοποίηση του συστήματος το συγκεκριμένο κουμπί εκτελεί την ενέργεια αυτή. Συγκεκριμένα, ελευθερώνει τη μνήμη από τα αντικείμενα που δημιουργήθηκαν για τις μονάδες λήψης δεδομένων καθώς και κλείνει όποια αρχεία πιθανόν να άνοιξαν κατά την αρχικοποίηση. Αν η ενέργεια αυτή είναι επιτυχής το κουμπί `Start Cycle` απενεργοποιείται. Η συντόμευση για αυτό το κουμπί είναι `ALT+N`.

- **Start Cycle:** Το κουμπί αυτό χρησιμοποιείται για την έναρξη του κύκλου λήψης δεδομένων. Ο κύκλος γεννά ένα νέο νήμα διαδικασιών (thread) και ξεκινάει την καταγραφή των δεδομένων του συγκεκριμένου run. Ποικίλες πληροφορίες τυπώνονται στο log κατά τη διάρκεια της συλλογής. Όταν το κουμπί `Start Cycle` πατηθεί το κουμπί `Stop Cycle` γίνεται ενεργό. Η συντόμευση για αυτό το κουμπί είναι `ALT+T`.

- **Stop Cycle:** Αν έχει επιλεχθεί πως για να σταματήσει ένα run θα πρέπει αυτό να φτάσει έναν μέγιστο αριθμό τότε έτσι θα γίνει. Ειδάλλως, για το σταμάτημα ενός ενεργού run θα πρέπει να πατηθεί αυτόι το πλήκτρο. Το κουμπί `Stop Cycle` ενεργοποιεί ξανά το κουμπί `Start Cycle` ενώ η συντόμευση του είναι `ALT+P`.

- **Help:** Το κουμπί `Help` ανοίγει μια σελίδα στον φυλλομετρητή του λειτουργικού συστήματος όπου υπάρχει μια ιστοσελίδα με πληροφορίες σχετικά με την διαδικασία που χρειάζεται να εκτελεστεί ώστε να τρέξει με επιτυχία το λογισμικό. Εν συντομία είναι ένα πρακτικό και γρήγορο αντίγραφο της παρούσας εργασίας δίνοντας κατηγοροιοποιημένες πληροφορίες. Η συντόμευση για το συγκεκριμένο κουμπί είναι `ALT+H`.

- **Close :** Είναι το πλήκτρο με το οποίο κλείνει το παράθυρο του λογισμικού. Η συντόμευση είναι `ALT+C`.

1.4.2 Τυπική Διαδικασία Λήψης Δεδομένων

Σε αυτό το κομμάτι περιγράφεται εν συντομία η τυπική διαδικασία για τη λήψη του αρχείου δεδομένων από ένα run. Αρχικά, για να εκτελέσει κάποιος το λογισμικό χρειάζεται να εκτελέσει την εντολή

```
./NTUADAQ
```

Στη συνέχεια θα πρέπει να ρυθμιστεί το λογισμικό μέσω της καρτέλας ρυθμίσεων. Σε αυτή θα πρέπει να ελεγχθεί η εγκυρότητα των πληροφοριών που περιέχονται στα widgets. Μετά ακολουθεί η σύνδεση με το υλικό με τη χρήση του κουμπιού `Initialize`. Όταν πατηθεί το συγκεκριμένο κουμπί γίνεται η σύνδεση του ηλεκτρονικού υπολογιστή με τη μονάδα V2718, αρχικοποιούνται οι μονάδες V551B και V550 (και να υπάρχει η V462) και ανοίγει το αρχείο εξαγωγής. Αν η αρχικοποίηση είναι επιτυχής τότε το κουτί ελέγχου δίπλα στο κουμπί `Initialize` αλλάζει κατάσταση σε `Qt::Checked` και τα δύο κουμπιά από κάτω ενεργοποιουνται. Το πρώτο (`Cancel Init.`) αν πατηθεί ακυρώνει την αρχικοποίηση ενώ το δεύτερο (`Start Cycle`) ξεκινάει τον κύκλο λήψης δεδομένων. Ενώ ο κύκλος είναι σε εξέλιξη ένα σήμα διακοπής (interrupt) παράγεται από τις C-RAMS δηλώνοντας πως υπάρχουν δεδομένα για λήψη. Όταν αυτό το σήμα εγκυροποιηθεί η διαδικασία καταγραφής ξεκινάει γράφοντας ένα event data μπλοκ στο αρχείο εξόδου μεταφέροντας τα στοιχεία 32bit λέξεων από το FIFO της κάθε C-RAMS στο αρχείο με τη μορφή ενός δεκαεξαδικού. Το run θα τελειώσει είτε όταν φτάσει στον μέγιστοαριθμό γεγονότων στον οποίο έχει ρυθμιστεί, είτε αν πατηθεί το κουμπί `Stop Cycle`. Για να κλείσει το run, το αρχείο εξόδου κλείνει, οι μονάδες υλικού επανέρχονται στην αρχική τους κατάσταση και η καρτέλα ρυθμίσεων ανανεώνεται στην νέα της μορφή (στην ουσία αλλάζει ο αριθμός του run που είναι έτοιμο να ληφθεί) όντας έτοιμη για την επόμενη χρήση της.

CHAPTER 2

The Hardware

2.1 Introduction

During the installation and implementation phase of the software a specific set of hardware modules was used. In this chapter, each part of this set will be presented along with its most basic properties. It goes without saying that for someone to present all of the properties of each module would be a Herculean task, so only the ones that are used throughout the code are presented in detail whereas the rest are simply referred.

2.2 The VME Crate

2.2.1 The Crate

The architectural concepts of VMEbus are based on the VERSAbus developed by Motorola in the late 1970s. The VMEbus specification defines an interfacing system used to interconnect microprocessors, data storage and peripheral control devices in a closely coupled hardware configuration. The system has been conceived with the following objectives:

- To allow communication between devices on the VMEbus without disturbing the internal activities of other devices interfaced to the VMEbus.
- To specify the electrical and mechanical system characteristics required to design devices that will reliably and unambiguously communicate with other devices interfaced to the VMEbus
- To specify protocols that precisely define the interaction between the VMEbus and devices interfaced to it.
- To provide terminology and definitions that describe system protocol.
- To allow a broad range of design latitude so that the designer can optimize cost and/or performance without affecting the system compatibility.

- To provide a system where performace is primarily device limited, rather than system interface limited.

A conventional VME Crate is constructed on a subrack. That is a rigid framework that provides mechanical support for boards inserted to the backplane, ensuring that the connectors mate properly and that adjacent boards do not contact each other. Furthermore, it guides the cooling airflow through the system and ensures that inserted boards do not disengage themselves from the backplane due to vibration or shock. The VMEbus backplane is a printed circuit board with 96 or 160 pin connectors and signal paths that bus the connector pins. The subrack provides the slots, which are positions where a board can be inserted into a VMEbus backplane. Finally, a board is a printed circuit board, its collection of electronic components with either one or two 96 or 160 pin connectors that can be plugged into VMEbus backplane connectors.

The VMEbus provides the end-user with a collection of cycles performing a wide range of processes. However before going in further detail a definition of the Master and Slave terms is needed. A Master is a functional module, that means a collection of electronic circuitry that resides on one board, which initiates a Data Transfer Bus (DTB) Cycle in order to transfer data between itself and a Slave module. In addition, a Slave is a functional module that detects DTB cycles initiated by a Master and when those cycles specify its participation, transfers data between itself and the Master. The types of cycles on the VMEbus are :

- **Read Cycle** - A Data Transfer Bus cycle used to transfer 1, 2, 3, 4 or 8 bytes from a Slave to a Master. The cycle begins when the Master broadcasts an address and an address modifier. Each slave captures the address modifier and address and checks to see if it is to respond to the cycle. If so, it retrieves the data from its internal storage, places it on the data bus and acknowledges the transfer. The Master then terminates the cycle.
- **Write Cycle** - A DTB cycle used to transfer 1, 2, 3, 4 or 8 bytes from a Master to a Slave. The exact stages as at the read cycle are followed.
- **Block Read Cycle** - A DTB cycle used to transfer a block of 1 to 256 bytes from a Slave to a Master. This transfer is done using a string of 1, 2 or 4 byte data transfers. Once the block transfer is started, the Master does not release the DTB until all of the bytes have been transferred. It differs from a string of read cycles in that the Master broadcasts only one address and address modifier (at the beginning of the cycle). Then the Slave increments this address on each transfer so that the data for the next transfer is retrieved from the next higher location.
- **Block Write Cycle** - A DTB cycle used to transfer a block of 1 to 256 bytes from a Master to a Slave. The block write cycle is very similar to the block read cycle. It uses a string of 1, 2 or 4 byte data transfers. The Master does not release the DTB until all of the bytes have been transferred. Again the difference from a string of write cycles is the incrementation of the address by the Slave.
- **Multiplexed Cycle** - A DTB cycle that transfers address information and/or data information using both the address and the data buses. Multiplexed cycles are used in four cases.

  A64 the full address bus and the full data bus are combined to create a 6 bit address.

  MBLT the full address bus and the full data bus are combined to create a 64 bit data word.

  A40 the full 24 bit address bus and the full 16 bit data bus on the P1/J1 connector are combined to create a 40 bit address.

  MD32 the lower 16 address lines and the lower 16 data lines are combined to create a 32 bit data word.

  A Multiplexed Cycle will have an Address Phase that is separate from the Data Phase. The Address Phase may include (i.e. A64 and A40 cycles) or may not include (i.e. A32, A24 cycles) the use of Data Bus. The Data Phase may include (i.e. MBLT, MD32 cycles) or may not include (i.e. D32, D16 cycles) the use of the Address Bus.
- **Read-Modify-Write Cycle** - A DTB cycle that is used to both read from and write to a Slave location without permitting any other Master to access that location. This cycle is most useful in multiprocessing systems where certain memory locations are used to provide semaphore functions.
- **Address-Only Cycle** - A DTB Cycle that consists of an address broadcast, but no data transfer. Slaves do not acknowledge Address-Only cycles and Masters terminate the cycle without waiting for an acknowledgement. No data strobes or acknowledge strobes are asserted in an Address-Only cycle.

- **Address-Only-With-Handshake Cycle** - A DTB cycle that consists of an address broadcast, but no data transfer. The addressed Slave responds in the same manner as a standard access cycle.
- **Interrupt Acknowledge Cycle** - A DTB cycle, initiated by an Interrupt Handler, which reads a STATUS/ID from an Interrupter. An Interrupt Handler generates this cycle whenever it detects an interrupt request from an Interrupter and it has control of the DTB.

The VMEbus functional structure can be divided into four categories. Each consists of a bus and its associated functional modules which work together to perform specific duties. Each category is briefly summarized below.

- **Data Transfer** - Devices transfer data over the Data Transfer Bus (DTB), which contains data and address pathways and associated control signals. Functional modules called Masters, Slaves, Interrupters and Interrupt Handlers use the DTB to transfer data between each other. The other modules, called Bus Timer and IACK Daisy Chain Driver also assist them in this process.
- **DTB Arbitration** - Since a VMEbus system can be configured with more than one Master or Interrupt Handler, a means is provided to transfer control of the DTB between them in an orderly manner and to guarantee that only one Master controls the DTB at a given time. The Arbitration Bus modules (Requesters and Arbiter) coordinate the control transfer.
- **Priority Interrupt** - The priority interrupt capability of the VMEbus provides the means by which devices can request services from an Interrupt Handler. These interrupt requests can be prioritized into a maximum of seven levels. Interrupters and Interrupt Handlers use the Priority Interrupt Bus signal lines.
- **Utilities** - Periodic clocks, initialization and failure detection are provided by the Utility Bus. It includes a general purpose system clock line, a system reset line, a system fail line, an AC fail line and two serial lines. Utilities also include power and ground pins.

2.2.2 CAEN V2718 VME-PCI Optical Link Bridge

The module V2718 is a single unit wide VME master module, which can be interfaced to the CONET (Chainable Optical NETwork) and controlled by a standard personal computer equipped with the PCI card CAEN Mod. A2818. The A2818 is a 32bit 33MHz PCI card; the communication path uses optical fiber cables as physical transmission line. Up to 8 V2718 VME masters can be controlled by one A2818 CONET controller. The module is capable of performing all of the cycles, addressing and data transfer modes foreseen by the VME64X specifications (see subsection 1.2.1). For our software it suffices to use the default settings of the controller so except for a call to initialize the module, no other registers are modified. A table of all the registers of the controller are presented in ??.
Each module that may be inserted in a slot of the crate and controlled by V2718 module needs a specific 32bit word that identifies the controller and assigns the module as a slave of the controller. Finally, the module supports Single Read/Write Cycle, Read Modify Write Cycle, Block Transfer Cycle and Multiplexed Block Transfer Cycle for data transfer, as well as A16, A24, A32, CR/CSR, ADO and ADOH addressing.

2.2.3 CAEN V551B C-RAMS Sequencer

The model V551B CAEN C-RAMS Sequencer is a 1-unit wide VME module that handles the Data Acquisition from multiplexing front-end chips. The module is well suited to handle the VA family of chips (produced by IDE AS, Oslo) but due to its flexibility it can also be used with similar chips (Amplex, Gasiplex, etc.).

The V551B has been developed to control the signals from and to the C-RAMS (CAEN Readout for Analog Multiplexing Signals) boards module V550, the latter taking care of the conversion of the multiplexed signals from the front end boards housing the above chips. A single V551B can control up to 19 C-RAMS modules in a complete VME crate, this enabling the readout of 76608 (19 C-RAMS $\times$ 2 FIFO $\times$ 2016 words) multiplexed detector channels.

The multiplexing frequency can be set via VME from 100kHz to 5MHz, with programmable Duty Cycle. The delay between the multiplexing Clock signal and the Convert signal of the acquisition cards

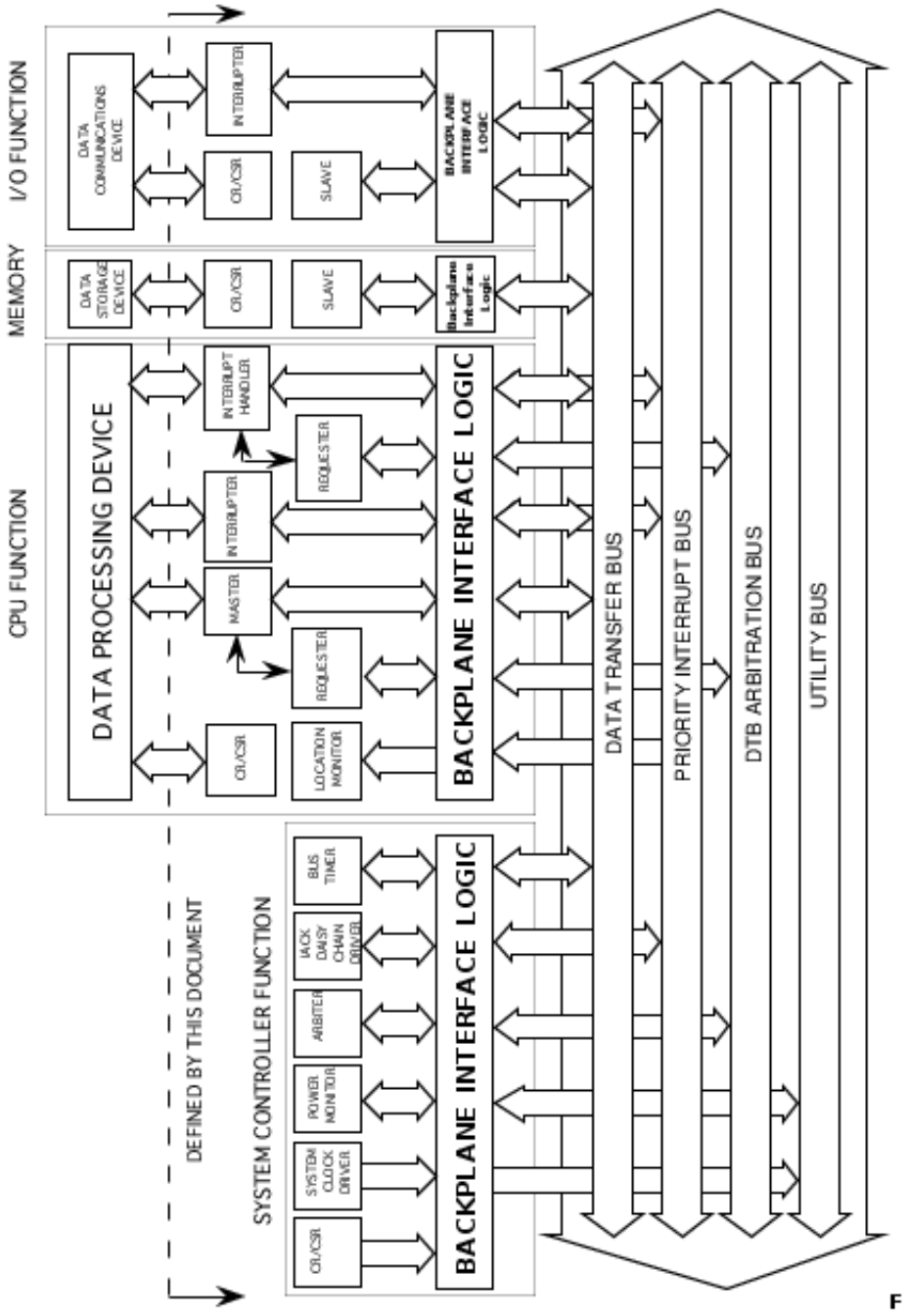


Figure 2.1: Interfaces of VMEbus.

NAME	ADDRESS	TYPE	BITS	FUNCTION
STATUS	00	read	16	Status register
VME_CTRL	01	read/write	16	VME control register
FW_REV	02	read only	16	Firmware revision
FW_DWNLD	03	read/write	8	Firmware download
FL_ENA	04	read/write	1	Flash enable
IRQ_STAT	05	read only	7	IRQ status
IRQ_MASK	06	read/write	7	IRQ mask
IN_REG	08	read/write	7	Front panel input register
OUT_REG_S	0A	read/write	11	Front panel output register
IN_MUX_S	0B	read/write	12	Input multiplexer set
OUT_MUX_S	0C	read/write	15	Output multiplexer set
LED_POL_S	0D	read/write	7	LED polarity set
OUT_REG_C	10	write only	11	Front panel output register clear
IN_MUX_C	11	write only	12	Input multiplexer clear
OUT_MUX_C	12	write only	15	Output multiplexer clear
LED_POL_C	13	write only	7	LED polarity clear
PULSEA_0	16	read/write	16	Period and width of pulser A
PULSEA_1	17	read/write	10	# of pulses and range of pulser A
PULSEB_0	19	read/write	16	Period and width of pulser B
PULSEB_1	1A	read/write	10	# of pulses and range of pulser B
SCALERB0	1C	read/write	11	End Count Limit and Autores of scaler
SCALER1	1D	read only	10	Counter value of scaler
DISP_ADL	20	read only	16	Display AD[15:0]
DISP_ADH	21	read only	16	Display AD[31:16]
DISP_DTL	22	read only	16	Display DT[15:0]
DISP_DTH	23	read only	16	Display DT[31:16]
DISP_PC1	24	read only	12	Display control left bar
DISP_PC2	25	read only	12	Display control right bar
LM_ADL	28	read/write	16	Local monitor AD[15:0]
LM_ADH	29	read/write	16	Local monitor AD[31:16]
LM_C	2C	read/write	9	Local monitor controls

Table 2.1: V2718 controller registers map

can be adjusted to wait for the settlement of the analog signal coming from the multiplexers. The delay between the Trigger and the Hold signal and the delay between the Hold and the Conversion Cycles are also programmable, thus extending the module's flexibility.

The module houses a VME RORA (Release on Register Access) Interrupter. Via VME it is possible to program the interrupt generation on the condition that DRDY signal is asserted, signaling that at least one channel in a system has data to be read out. The module works in A23/A32 mode and the data transfer occurs in D16 mode and its base address can be set via two rotary switches placed on the board.

The registers of the module will be briefly presented as they are accessed throughout the code.

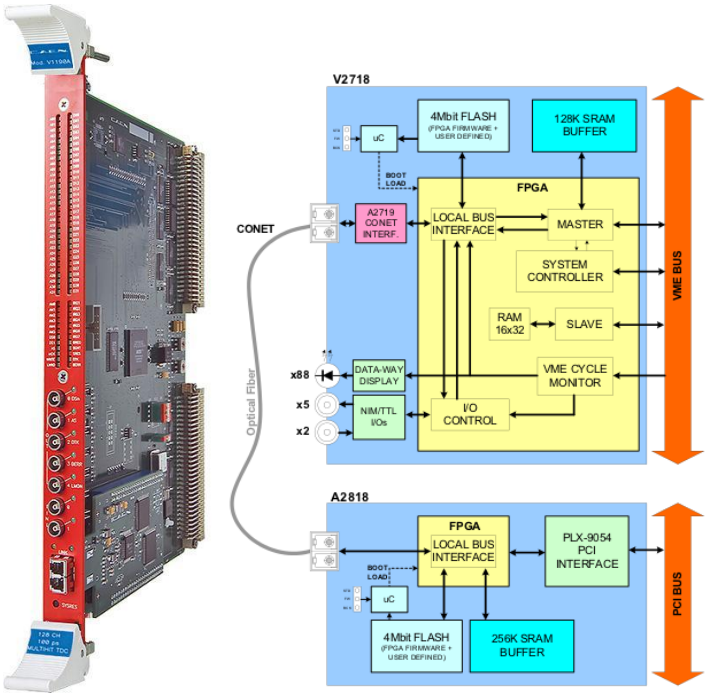


Figure 2.2: Front view and block diagram of CAEN V2718 Controller

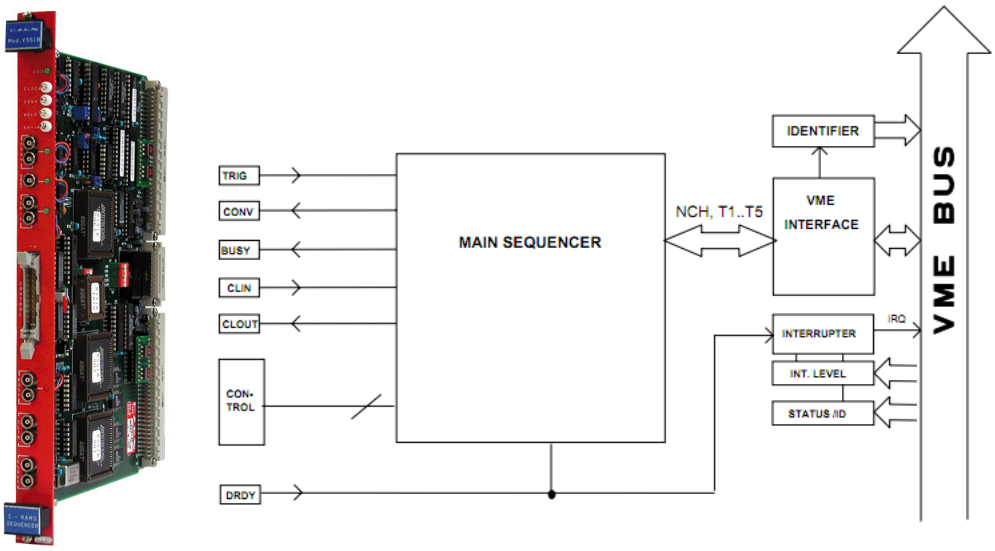
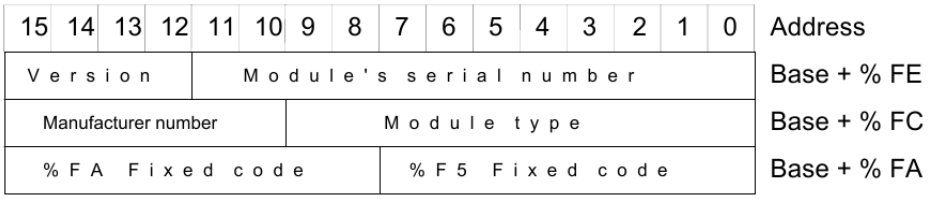


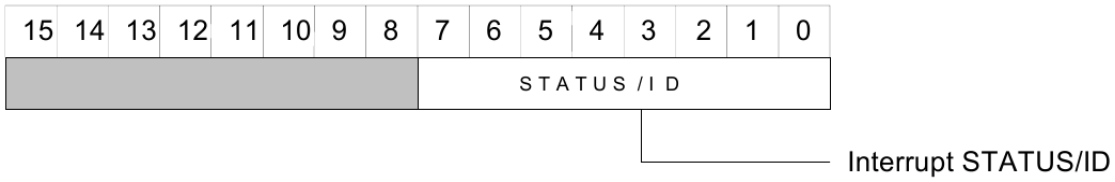
Figure 2.3: Front view and block diagram of mod. V551B.

Module Identifier Words Register

This 16bit register is located at the address Base+%FE and identifies a single module via a serial number and any change in the hardware will be shown by the Version number.

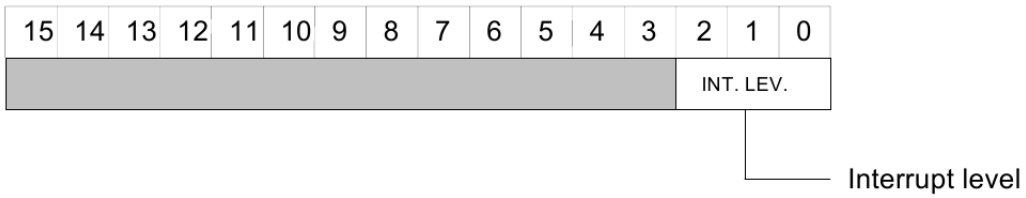


Interrupt Vector Register



This 16bit register is located at the address Base+%00 and contains the STATUS/ID that the V551B Interrupter places on the VME data bus during the Interrupt Acknowledge cycle.

Interrupt Level Register



This 16bit register is located at the address Base+%02 and contains the value that the V551B Interrupter places on the VME data bus during the Interrupt Acknowledge cycle.

Clear Register

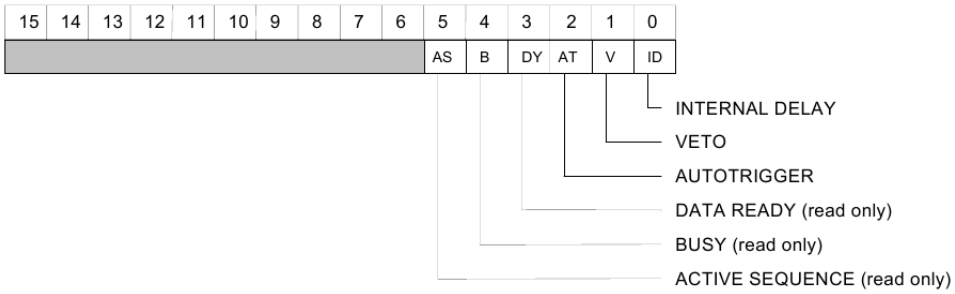
This register is located at the address Base+%04 and a VME access (read or write) to this location causes the following:

- a pulse of 500ns is generated on the `CLEAR` output
- a pulse of 500ns is generated on the `DRESET` line of the Control Bus
- a pulse of 500ns is generated (if enabled) on the `ARESET` line of the Control Bus
- if the conversion sequence is in progress, it is aborted and this causes an anticipated pulse (1μ s duration) on the `DRESET` (also `ARESET` if enabled) line of the Control Bus, while the `BUSY` output becomes not active.

Trigger Register

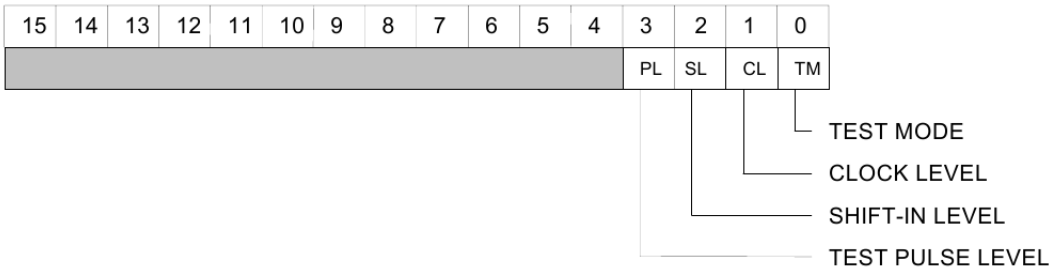
This register is located at the address Base+%06 and a VME access (read or write) to this location starts a conversion sequence. The same action is performed if the `TRIGGER` input signal is active.

Status Register



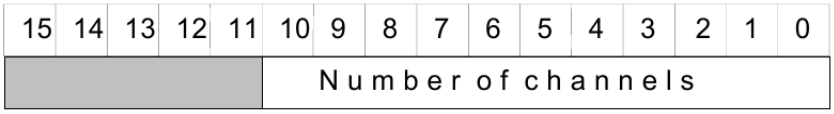
This 16bit register is located at the address Base+%08 and contains information concerning the status of the Sequencer module. Only six of sixteen bits are used which are presented in the figure above. The first bit (bit 0) defines if there is an internal delay on the Mod. V551B, next is the `VETO` bit. When in veto state, the Sequencer does not accept or send any signals for data taking. Incrementing the bit number, one is presented with the `AUTOTRIGGER` bit which defines if the module is autotriggering its way to data taking and the bit for data ready (places a signal of `DRDY` on the VME lines). Finally, the last two bits are for the `BUSY` state of the Sequencer and if there is an active readout sequence at the moment of accessing the register, that is if the Mod. V551B is in a status between an accepted `TRIGGER` and a `DRESET` generation. The `BUSY` bit is on during a conversion sequence in which no other data are to be taken for conversion.

Test Register



This 16bit register is located at the address Base+%0A and contains information about the test state which is provided by Mod. V551B. The first bit defines if the user will use the test mode or not and the others define the variables of the test state.

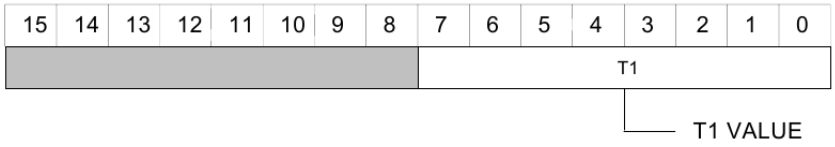
Number of Channels Register



This 16bit register is located at the address Base+%0C and contains the number of the channels that will be read out. The number N of detector channels to be read out by the C-RAMS can be programmed

via this register up to 2047 (though the V550 C-RAMS can accept only up to 2016 detector channels).

T1 Register

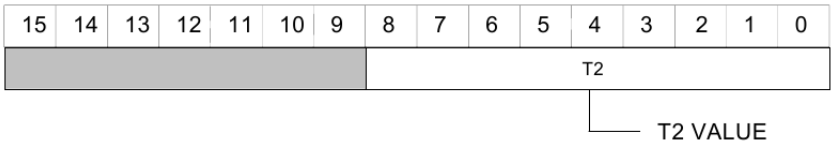


This 16bit register is located at the address Base+%0E and is used to set the $T1$ parameter on 8 bits. It gives the delay t_1 between the Leading Edge of the `TRIGGER` and the `HOLD` assertion. The actual delay t_1 (in nanoseconds) is calculated as follows:

$$t_1 = 500 + T_1 \times 10ns$$

where $0 \leq T_1 \leq 255$.

T2 Register

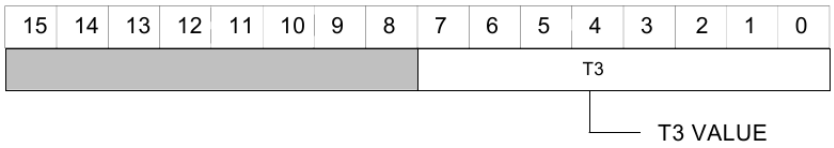


This 16bit register is located at the address Base+%10 and is used to set the $T2$ parameter on 9 bits. It gives the delay t_2 between the `HOLD` assertion and the start of the `CLOCK/CONVERT` sequence. The actual delay t_2 (in nanoseconds) is calculated as follows:

$$t_2 = 130 + T_2 \times 20 \pm 10ns$$

where $10 \leq T_2 \leq 511$.

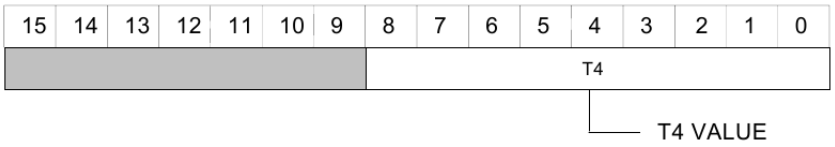
T3 Register



This 16bit register is located at the address Base+%12 and is used to set the $T3$ parameter on 8 bits. It gives the duration t_3 of the active phase of the `CLOCK` and the `CONVERT`. The actual duration t_3 (in nanoseconds) is calculated as follows:

$$t_3 = T_2 \times 20ns$$

where $1 \leq T_3 \leq T_4$ and $T_3 \leq 255$. This constraint ($T_3 \leq T_3$) follows automatically from the fact that the active phase of the `CLOCK` and `CONVERT` must be less than their own period.



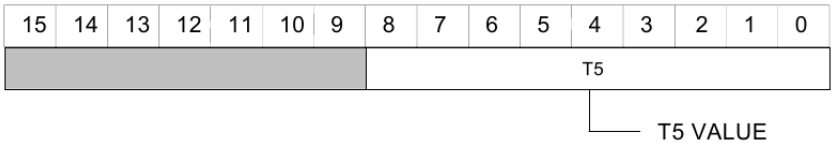
T4 Register

This 16bit register is located at the address Base+%14 and is used to set the $T4$ parameter on 9 bits. It gives the period t_4 of both the `CLOCK` and the `CONVERT` sequence. The actual period t_4 (in nanoseconds) is calculated as follows:

$$t_4 = 20 + T_4 \times 20ns$$

where $1 \leq T_4 \leq 511$.

T5 Register



This 16bit register is located at the address Base+%16 and is used to set the $T5$ parameter on 9 bits. In `NORMAL MODE` it gives the delay t_5 between the `CLOCK` and the relevant `CONVERT`. The actual delay t_5 (in nanoseconds) is calculated as follows:

$$t_5 = 40 + T_5 \times 20ns$$

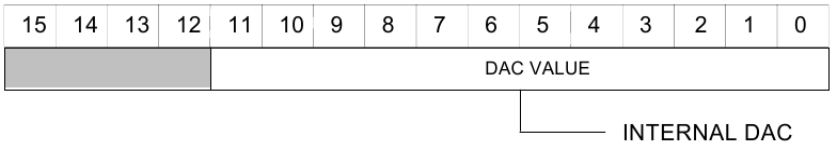
where $2 \leq T_5 \leq 511$.

In `TEST MODE`, it gives the delay t_6 between the Leading Edge of the `TEST PULSE` and the first `CONVERT` pulse. The actual delay t_6 (in nanoseconds) is calculated as follows:

$$t_6 = 150 + T_5 \times 20 \pm 10ns$$

where $2 \leq T_5 \leq 511$. The $10ns$ Jitter is due to the synchronization with an internal Oscillator.

Write Internal DAC Register



This register is located at the address Base+%18 and allows to set the Analog positive voltage VCAL on the front panel "CONTROL" connector. It is 12 bits long and the full scale value ($0xFFF$) corresponds to a $+5V$, $50mA$ (max.) output on the VCAL line. The polarity can be changed via an internal DIP (Dual In-line Package) switch.

Standard Operations

In order to proceed with an ordinary acquisition cycle, the user must perform a series of settings, and thereafter start with the ordinary cycle. The following describes the settings to be done and the corresponding operation sequence.

The readout sequence starts with an external or a VME TRIGGER. The BUSY becomes active, indicating that the module cannot accept another TRIGGER. The leading edge of the TRIGGER starts a monostable multivibrator circuit. This circuit, after a time t_1 programmable via VME in $10ns$ step in the range $500ns$ to $3\mu s$ approximately, activates the HOLD and the SHIFT IN signals (the latter after $t_1 + 300ns$). The HOLD signal is used to sample the signal at the output of the shapers at peaking time. The SHIFT IN is the token signal for the first chip in the multiplexing chain and must be active at the occurrence of the first CLOCK. For this purpose, a hold time of $100ns$ is provided.

Once the HOLD is asserted, the CLOCK cycles begin after a time t_2 programmable via VME in $20ns$ steps in the range of $170ns$ to $10\mu s$ approximately. The CLOCK is generated by an internal $50MHz$ oscillator: due to this, the acutal start of the readout (i.e. the first MUX CLOCK) will be delayed with respect to the HOLD assertion with a $\pm 10ns$ jitter.

In the following readout sequence, N CLOCK and CONVERT pulses are generated. The number N of detector channels (between 1 and 2047) can be programmed via VME (though the V550 C-RAMS can accept up to 2016 detector channels).

Each CLOCK pulse is followed by a CONVERT pulse after a delay of $t_5 ns$, programmable via VME in $20ns$ steps in the range of $80ns$ to $10\mu s$ approximately. The purpose of this delay is to wait for the settlement of the analog signal coming from the multiplexers. The width of the active phase of the CLOCK and CONVERT pulses is $t_3 ns$ programmable via VME in $20ns$ steps in the range $20ns$ to $5\mu s$ approximately.

The repetition period t_4 of the CLOCK and CONVERT pulses (and consequently the multiplexing frequency) is programmable via VME in $20ns$ steps in the range $40ns$ to $10\mu s$ approximately.

The CLOCK, CONVERT, HOLD and SHIFT-IN signals are available on four test points placed on the front panel. The active level on the test points is always high, disregarding the normal level of the relevant signals on the front panel connectors.

With the last CONVERT pulse, the DRESET line (also the ARESET if enabled) becomes active for a $1\mu s$ time and resets the front end circuitry.

After the generation of the last CONVERT pulse, the BUSY signal becomes not active if none of the V550 acquisition cards has asserted the DRDY signal. Otherwise, when the DRDY has been raised (at least one channel has data ready), the BUSY signal remains high until all the V550 FIFOs have been read out, i.e. until the DRDY becomes not active.

A software controlled VETO is also available. The VETO forces the V551B in a BUSY condition (no TRIGGER accepted).

Interrupt Generation

The operations of the V551B VME RORA Interrupter are fully programmable; via VME it is possible:

- to set the VME Interrupt Level
- to program the VME Interrupt Vector (STATUS/ID)

The interrupt is generated on the assertion of the DRDY input signal, which is the logical wired-OR of all DRDY signals coming form the acquisition cards. Thus, the interrupt is requested when, at the end of the readout sequence at least one channel in the system has data to be read out, and is released when all the FIFOs have been completely read out.

If the Interrupt Level is set to 0, no interrupts will be generated from the V551B module.

2.2.4 CAEN V550 C-RAMS

The model V550 CAEN Readout for Analog Multiplexed Signals (C-RAMS) is a single unit wide VME module housing two independent Analog to Digital Conversion blocks to be used for the readout of analog multiplexed signals coming from some of the well known front-end chips (Amplex, Gasiplex, Viking, etc.).

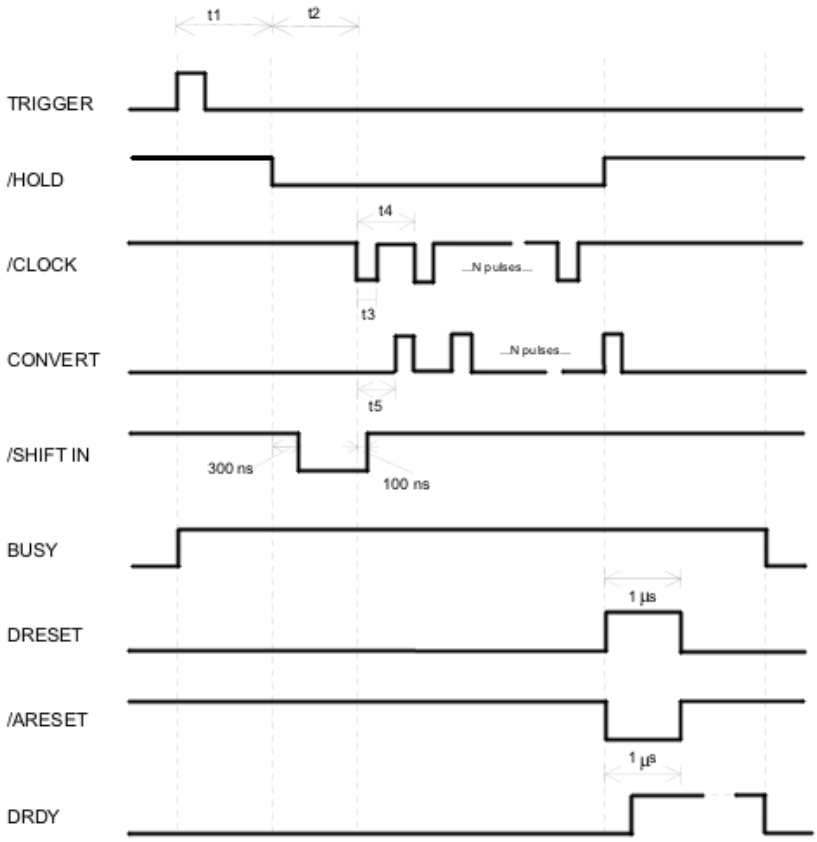


Figure 2.4: Standard operation modes of the Mod. V551B.

Each block of the module accepts positive, negative or differential input signals; the signals are amplified and fed to an ADC. The sensitivity (mV/bit) can be selected among 4 different values (with relative ratios of 1, 2, 5 and 10) by means of internal jumpers. The module has conversion rate up to $5MHz$, differential input with selectable amplification, 12 bit linear conversion, zero suppression and pedestal subtraction, diagnostics and self-test capabilities.

With the occurrence of an external `CONVERT` signal, the input signal is sampled by the ADC and its digital value is compared to a threshold value, if the signal is over the threshold, the pedestal is subtracted and the result is stored in an output buffer arranged in FIFO logic $2K \times 32$ bit. For this purpose each block of the module houses two memories for the storage of the thresholds and the pedestals of each detector channel. The pedestal and threshold values are independent for each channel and the pedestal/threshold memory, which is arranged in $2K \times 24$ bit, can be filled (and read back) via VME with the desired values.

The number N of detector channels to be read out can be programmed via VME between 32 and 2016 in steps of 32. At the end of a conversion cycle (N `CONVERT` pulses), with the last word stored in the FIFO, if there are data in the FIFO, the module channel goes in the Data Ready state signaling that the data must be read via VME. A positive open-collector signal ("DRDY") is available for each channel on the front panel and is provided with two bridged connectors for daisy chainng. A fast `CLEAR` signal is also available for cycle abort.

It is possible to operate the module also in `TEST` mode (VME selectable) by simulating some input

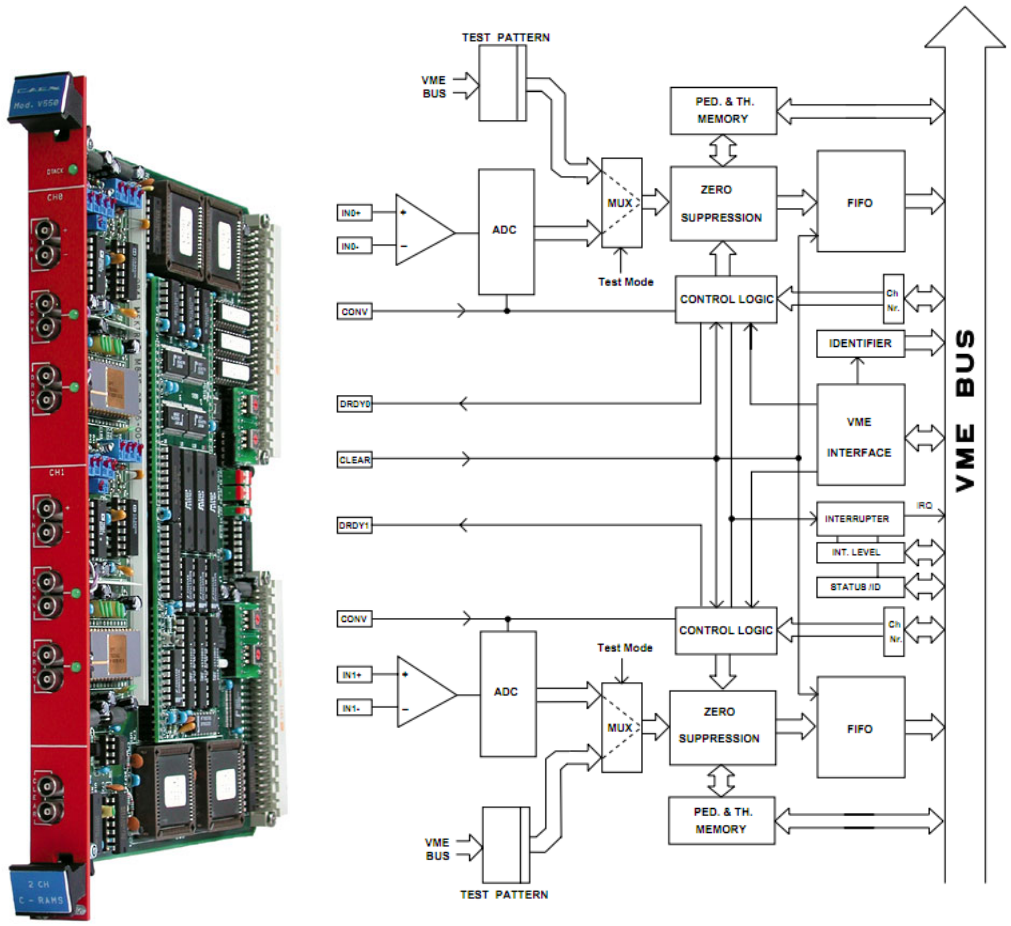


Figure 2.5: Front view and block diagram of mod. V550.

patterns, which can be written via VME, as if they were coming from the ADC.

Finally, the module works in A24/A32 mode. The data transfer occurs in D32 and a block transfer mode is also available.

In the following, the registers of a V550 module are presented in short.

Module Identifier Words Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Address
Version				Module's serial number												Base + % FE
Manufacturer number						Module type										Base + % FC
%FA Fixed code								%F5 Fixed code								Base + % FA

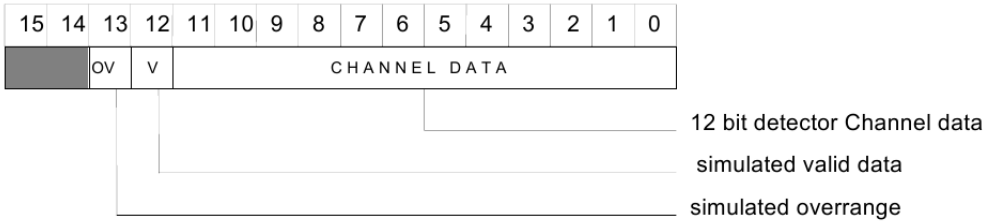
This register contains three 16bit words that are used to identify a module located at the address Base+%FA, Base+%FC, Base+%FE. At the address Base+%FA the two particular bytes allow the au-

tomatic localization of the module. For the Mod. V550 the word address Base+%FC has the following configuration :

Manufacturer Number = 000010 b
Type of module = 00000110100

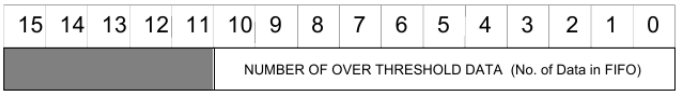
The word located at the address Base+%FE identifies the single module via a serial number and any change in the hardware (for example the use of a faster Conversion Logic) will be shown by the Version number.

Test Pattern Register



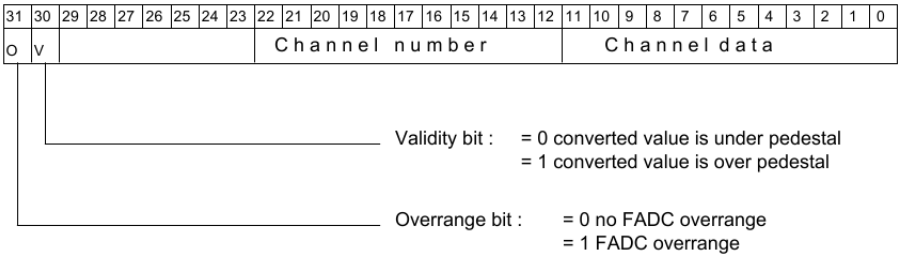
This 16bit register is located at the address Base+%16 for Channel 1 and Base+%14 for Channel 0 and contains the values of a simulation pattern returned by Mod. V550.

Word Counter Register



This 16bit register is located at the address Base+%12 for Channel 1 and Base+%10 for Channel 0 and contains the number of words that are over threshold, that is of course the number of data in FIFO.

FIFO Register

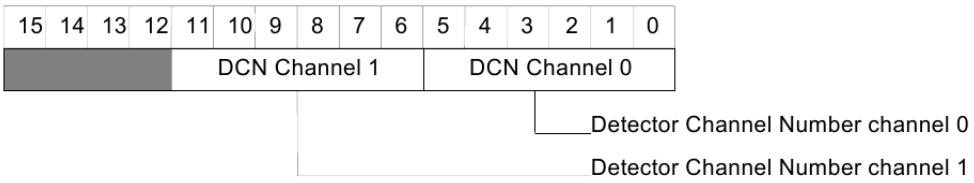


This 32bit register is located at the address Base+%0C for Channel 1 and Base+%08 for Channel 0. It contains the values of the ADC channel number that holds the value of channel data when these data are valid or over range. The two FIFOs are also accessible in block transfer mode.

Clear Module Register

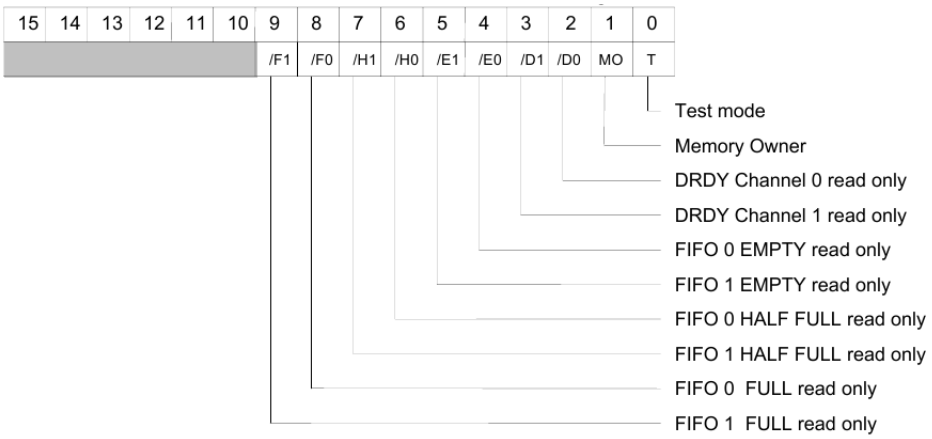
This register is located at the address Base+%06 and one can access it only in Write Mode. A VME write access to this location aborts the conversion process, if one is active, clears the FIFOs and clears the word counters.

Number of Channels Register



This 16bit register is located at the address Base+%04 and allows to program the number of detector channels to be read out in step of 32. This number ranges from 32 ($DCN = 1$) to 2016 ($DCN = 63$). If DCN equals zero then only one detector channel will be read out.

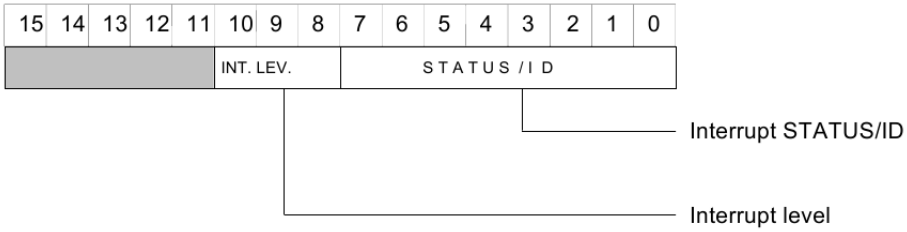
Status Register



This 16bit register is located at the address Base+%02 and offers information about the status of the module at the moment of access. In incrementing order the bits indicate if a test mode is activated, the memory owner of the module (VME or Conversion Logic), if there are ready data on channels 0 or 1 and if the FIFOs of channels 0 and 1 are empty, half full or full.

Interrupt Register

This 16bit register is located at the address Base+%00 and contains the value of the Interrupt Level and the STATUS/ID that the V550 Interrupter places on the VME data bus during the Interrupt Acknowledge cycle.



Pedestal and Threshold Register

CHANNEL 0

31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16 15 14 13 12	11 10 9 8 7 6 5 4 3 2 1 0	ADDRESS
	Pedestal detector channel 0	Threshold detector channel 0	Base + % 2000
	Pedestal detector channel 1	Threshold detector channel 1	Base + % 2004
	Pedestal detector channel 2047	Threshold detector channel 2047	Base + % 3FFC

CHANNEL 1

31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16 15 14 13 12	11 10 9 8 7 6 5 4 3 2 1 0	ADDRESS
	Pedestal detector channel 0	Threshold detector channel 0	Base + % 4000
	Pedestal detector channel 1	Threshold detector channel 1	Base + % 4004
	Pedestal detector channel 2047	Threshold detector channel 2047	Base + % 5FFC

These 32bit registers are located at the addresses Base+%2000 until Base+%5FFE for channel 0 and at Base+%4000 until Base+%5FFC for channel 1. These registers are accessible via VME only if the Memory Owner bit is set to zero (default value). The registers' size is 2048 words but only 2016 are actually used by the conversion logic as 2016 is the maximum number of channels.

Operation Sequence

Each channel of the unit accepts two analog signals via front panel connectors. The difference between the two signals is amplified and fed to the FADC. The sensitivity (mV/bit) can be selected among four different values (with relative ratios of 1, 2, 5 and 10) by means of internal jumpers.

The channel needs an external `CONVERT` pulse whose leading edge indicates that the analog signal must be sampled. The signal (ECL level) can be provided via pins of the VME backplane or via the front panel.

The number N of detector channels, which is between 32 and 2016, can be programmed via VME. In addition, the pedestal and threshold registers must be filled via VME with the chosen values. This memory can be accessed by VME only when the acquisition is stopped and the switching can be performed by means of the `CONTROL REGISTER`.

With the occurence of the leading edge of the `CONVERT` signal the analog signal is sampled by the FADC and its digital value is compared to the threshold of the current channel. If the channel is over threshold,

the pedestal is subtracted and the result is stored in the FIFO. The word in the FIFO has the following format

d<31>	d<30>	d<29..23>	d<22..12>	d<11..0>
Overrange	Data Valid	Reserved	Channel #	Channel pulse height

The D31 bit indicates a FADC overrange, while the D30 bit indicates that the field `PULSE HEIGHT` is valid, which means that is positive after pedestal subtraction. The bits from D23 to D29 are not specified. After N `CONVERT` pulses the data readout of an event is over. When the last `CONVERT` pulse has been processed and the FIFO is not empty, the card channel goes into `DATA READY` state, signaling that the data must be read. The `DATA READY` state is signaled by a bit of the status word (`DRDY`) and a positive open collector signal `DRDY` supplied via two front panel bridged connectors.

The daisy chain connection performs the wired-OR of the `DRDY` signals of different channels. When a channel is in `DATA READY` state the signal `CONVERT` has no effect on the card. After the last VME read from the FIFO, the `DRDY` signal goes low and the channel is ready for other acquisitions.

The readout from the FIFO can be performed either in a random VME read for each FIFO, or a block transfer for each FIFO. For this mode a word counter for each FIFO is available, in order to know the number of words stored in FIFO.

The beginning of the reading phase is triggered either by software polling of the `DRDY` bit, or by interrupt raised by the card on the condition that at least one of the `DRDY` of the two channels goes high, or even by an interrupt raised by the control card (CAEN Mod. V551B Sequencer) on the condition that the wired-OR of the `DRDY` signals goes high.

The system housed in a single crate, that is one VME CPU, one control card and M acquisitions cards) can handle up to $4032 \times M$ detector channels.

Interrupt Generation

The operations of the V550 VME RORA Interrupter are fully programmable; via VME it is possible

- to set the VME Interrupt Level
- to program the VME Interrupt Vector (STATUS/ID)

The interrupt generated on the logical OR of the two `DRDY` signals (at least one channel has ended the programmed N conversion cycles and its FIFO is not empty) and released when the two `DRDY` signals are low (the two FIFOs have been completely read out).

2.2.5 CAEN V462 Dual Gate Generator

The CAEN Model V462 is a Dual Gate Generator housed in a 1-unit wide VME module. Each module consists of two Gate Generators, one per channel, independently programmable between $100ns$ and $9.9999999s$. These generate three standard NIM signals: a Gate, a Begin Marker of fixed $100ns$ width, simultaneous to the beginning of the Gate, and an End Marker of fixed $100ns$ width, simultaneous to the end of the Gate.

The module is based on the use of Programmable Gate Arrays containing almost all the logic of each generator's operations. There are two working registers per channel, one for Local Mode, the other for VME Mode.

In Local Mode, the content of the Local Register is transferred to eight 4bit multiplexers that allow displaying of the gate width and is presented to some bit-to-bit comparators. The module contains eight 4bit BDC[1] counters that are positioned to zero if the gate is not triggered. When the Start signal takes place, the counters start counting at $10MHz$ frequency and the output of these counters is presented to the other input of the comparators.

[1] Binary Deciman Counter.

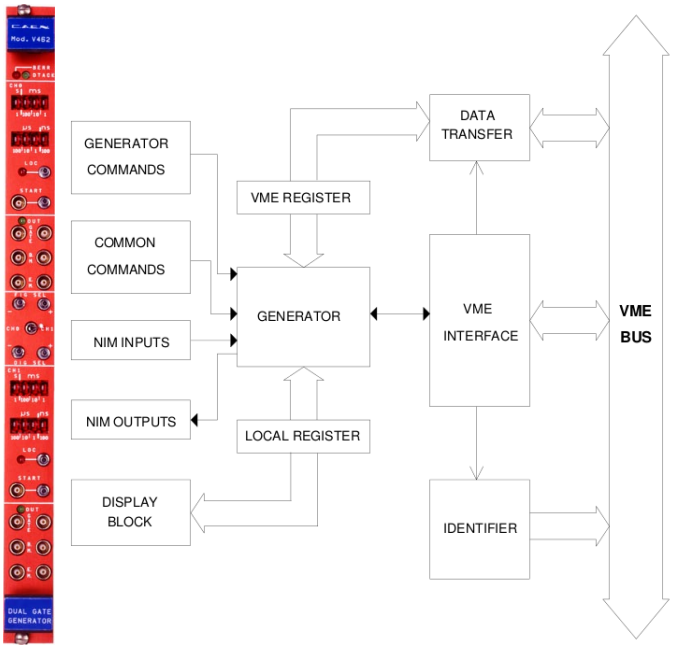


Figure 2.6: Front view and Block Diagram of CAEN Mod. V462 Dual Gate Generator.

When the 32bit word of the counters reaches the one of the selected register, the counting stops and the gate signal ends. A new cycle is possible right after the End of Gate Marker signal.

The Local Mode is characterized by the display of the content on the Local Register, which becomes the register to compare with the counters. Moreover, in Local Mode, the user can modify the content of the register via four push buttons and set a different width of the Gate. A digit (unit) that is selected via the `DIG SEL` +/− push buttons starts blinking and the value of the digit can be incremented or decremented with the `FIG SEL` +/− push buttons.

The VME Mode is characterized by the display of the content of the VME Register, which becomes the register to compare with the counters.The gate triggering and width are programmable via VME. The selection switches and push buttons for the width setting become totally ineffective on the selected register.

Whichever operating mode is selected, the Gate can always be triggered by the NIM signal or the push button "START" for each channel. When the Gate signal is present and for all the Gate duration, all the "START" signals are ignored ("non-updating" operation).

The Model V462 is an A24, D16 VME slave. Its base address is fixed by four internal rotary switches. A front panel LED (DTACK) lights up each time the module generates the VME signal DTACK. The main registers of this module are to be presented in short.

Module Identifier Words Register

These 16bit registers are located at the addresses Base+%FA, Base+%FC, Base+%FE and contain three words that are used to identify the module. At the address Base+%FA the two particular bytes allow the automatic localization of the module. For the mod. V462 the word at address Base+%FC has the following configuration
Manufacturer Number = 000010 b
Type of module = 0000001010 b
The word located at the address Base+%FE identifies the single module through the module's serial number and any change in the hardware will be shown by the Version number.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Address
Version				Module's serial number												Base + % FE
Manufacturer number						Module type										Base + % FC
%FA Fixed code								%F5 Fixed code								Base + % FA

Control and Status Register

D15	D14	D13	D12	D11	D10	D9	D8	...	D0
ERR	VME/ LOC 1	VME/ LOC 0	GATE CH. 1	GATE CH. 0	START CH. 1	START CH 0		...	

The Control and Status register is a 16bit register located at the address Base+%00 and contains some information on the status of the module in the five most significant bits:

- the bit in D15 is an error bit and indicates that one of the two generators is either in local mode or that its gate is open
- the bits D14 and D13 indicate that, respectively, channel 1 or channel 0 are in VME or Local Mode (bits are high if the channels are in VME mode)
- the bits D12 and D11 indicate that, respectively, the gate of channel 1 or channel 0 is open.
- Every attempt to write into one of the aforementioned bits generates a Bus Error and a LED on the front panel (`BERR`) lights on.
- The two bits of the Control and Status Register that can be used in write mode to trigger via VME the gate generators are D10 and D09 that trigger channel 1 and channel 0 gate generators respectively. Any attempt to read these two bits is ineffective.
- The bits in D8 through D0 are meaningless.

Generators Register

	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
BASE + %08	LEAST SIGNIFICANT DIGITS CHANNEL 1
BASE + %06	MOST SIGNIFICANT DIGITS CHANNEL 1
BASE + %04	LEAST SIGNIFICANT DIGITS CHANNEL 0
BASE + %02	MOST SIGNIFICANT DIGITS CHANNEL 0

These 16bit registers are located at the addresses Base+%08, Base+%06, Base+%04, Base+%02. The generator registers (gate width duration) are composed of two 16bit registers per channel. Address Base+%08 containes the four least significant digits of the gate width for channel 1, whereas Base+%06 the four most significant. The same pattern is for channel 0 at Base+%04 and Base+%02 for the least and most significant digits accordingly.

Front Panel Signals

To trigger externally the Gate Generators, two `START` inputs, one per channel, can be sent to the module. These should be standard NIM signals of $20ns$ minimum width, otherwise the Gate could not be triggered. A gate is triggered by the leading edge of the NIM `START` signal.

There are three output signals with a fan-out of two for each channel: the Gate, the Begin Marker and the End Marker. The Begin Marker is simultaneously with the Gate ending, and also its width is $100ns$.

The delay between the trigger signal and the leading edge of the Gate or Begin Marker signals is $(140 \pm 10)ns$.

2.3 The NIM Crate

For a readout trigger two plastic scintillators were used, whose signals were fed into a coincidence unit and the output was input for the V551B Sequencer `TRIGGER` signal. This section describes the modules of the NIM create on the rack that was used.

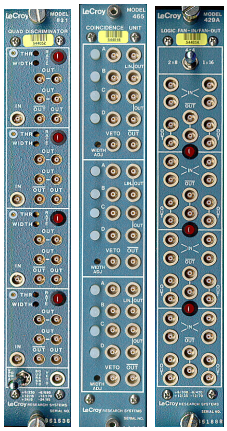


Figure 2.7: The NIM crate modules used. From left to right: LeCroy quad discriminator 821CS, LeCroy coincidence 465, LeCroy fan-in-fan-out logic 429A, and a custom-made unit from LAPP that is missing from the picture.

2.3.1 Quad Discriminator LeCroy 821CS

The discriminator is a device that responds only to input signals with a pulse height greater than a certain threshold value. If this criterion is satisfied, the discriminator responds by issuing a standard logic signal; if not, no response ins made. The value of the threshold can be adjusted by a helipot or screw on the front panel. As well, an adjustment of the width of the logic signal is usually possible via similar controls.

The LeCroy Model 821 is a high performance Quad Discriminator incorporating the features requested by experimenters throughout the world. Its hybrid front ends afford high sensitivity and greater than $100MHz$ counting rate capability. The input signal can be from $-30mV$ to $-1V$ and the outputs are six differential-type current source. The module offers a double-pulse resolution less than $9ns$.

The signals coming from the plastic scintillators were fed into the two channels of the discriminator. If the input pulse is higher than this threshold a logic output signal is produced by each channel and is fed into the coincidence unit.

2.3.2 Coincidence Unit LeCroy 465

The LeCroy Model 465 contains three independent high-speed general-purpose coincidence units in a single width NIM module. Each channel has four coincidence inputs and a separate veto input which accepts standard negative NIM logic levels. The logic inputs may be individually enabled or disabled without altering input cabling or termination by means of front-panel pushbutton switches. With all inputs enabled, four inputs are required. Disabling the logic inputs is equivalent to reducing the number of simultaneous negative inputs required for an output. Thus, each channel may be programmed for 4-fold, 3-fold or 2-fold logic decisions. With only one input enabled, each channel of the 465 operates a a logic fan-out.

Once triggered by singls satisfying the input coincidence requirements, the module generates five NIM fast logic outputs: one pair of $-32mA$ negative preset outputs, one $-16mA$ preset complementary output and one pair of $-32mA$ overlap outputs. The preset outputs are continuously adjustable from less than $5ns$ to greater than $500ns$ by means of a front-panel multiturn potentiometer and are independent of input overlap time, amplitude and rate. Because it is updating, it may be retriggered even before the end of an output pulse that is already present. The overlap outputs are equal in duration to the coincidence overlap and produce outputs up to the maximum input rate capability.

The logic signals of the discriminator unit were fed into the coincidence unit. The module compares the input signals and if a coincidence is occurred a logic pulse is generated that is input for the V551B VME Sequencer module as a `TRIGGER` signal.

2.3.3 LeCroy Fan-In-Fan-Out Logic 429A

Fan-outs are active circuits which allow the distribution of one signal to several signals of the same height and shape. This should be distinguished from the passive pulse splitter which divides both the signal and the amplitude. The fan-on, on the other hand, accepts several input signals and delivers the algebraic sum at the output. These modules may be bipolar, i.e. accepting signals of both polarities, or of the same polarity, i.e. accepting signals of one polarity only. Fan-ins are particularly useful for summing the outputs of the several detectors or the signals from a large detector with many outputs. Both fan-ins and fan-outs come in two varieties: linear and logic. The linear modules accept both analog and logic signals, whereas logic fan-outs and fan-ins are designed for logic signals only. In the case of a logic fan-in, the algebraic sum is replaced by a logical sum (i.e. `OR`).

Both fan-in and fan-out modules have the same basic function of combining the inputs and distributing multiple outputs. The Model 429A is a multi-functional fast logic module designed to fulfill a wide variety of signals handling needs. It combines the operations of TTL-to-NIM level translation, logic fan-in, logic fan-out and polarity inversion in one low-cost module. Each of the 4 channels has four inputs which accept both NIM and TTL levels. This is particularly usefull for test setups and experiments where digital triggers and/or control logic may use both signal standards.

Each channel of the 429A includes four independent logic inputs, four normal logic outputs and two complementary logic ouputs. Channels may be paralleled to provide up to 16 inputs and 24 outputs by means of a front-panel switch. An efficient circuit design holds power dissipation of the entire module to within the NIM standard.

The 429A eliminates the extra cabling and time delay involved when conventional fan-ins and fan-outs must be cascaded. In addition, it eliminates the common use of expensive logic units to perform logical `OR` with adequate fan-out. The ability to conveniently parallel channels permts a degree of flexibility and efficiency heretofore unavailable.

Inputs are $50Ohm$ impedance for NIM or TTL signals. Unused inputs need not be terminated. Inputs may be driven with single or double amplitude NIM signals or TTL signals without affecting output amplitude. The three pairs of bridged outputs are direct-coupled current sources which deliver $-32mA$ into two $50Ohm$ loads. Output duration is equal to the logical `OR` of the input durations.

2.4 The Front-End Chip

The Front-End Chip that was used for the data acquisition was a 64-channel multiplexing frontend that uses an integrated circuit called Gassiplex. Gassiplex was developed at CERN as an improved version of the well known AMPLEX chip.

The Gassiplex chip has 96 input channels that consist of a charge sensitive amplifier (CSA), a switchable filter, a shaper and a track and hold stage. These channels can be multiplexed to one output allowing a sequential readout of all channels. In computer networks, multiplexing (also known as muxing) is a method by which multiple analog message signals or digital data streams are combined into one signal over a shared medium. The aim is to share an expensive resource.

2.5 The MicroMeGAS Detector

Micromegas (stands for MicroMesh GAseous Structure) detector which falls into the category of MicroPattern Gaseous Detectors (MPGD) was created by J. Collar and G. Giomataris in the middle '90s. At first, it was proposed to detect low energy photons ($1 - 10keV$). Because of its high gain nature, Micromegas can stand up alone without the need of an additional preamplification. The fact that its operational principle follows the conventional one from MPGD provides the end user with a variety of benefits such as

- stability
- quick response
- great energy and spatial resolution
- high efficiency
- granularity
- radiation hardness
- usability in experiments with rare events if the background is relatively low.

The detector has already been used in numerous experiments such as COMPASS (COmmon Muon-Proton Apparatus for Structure and Spectroscopy, CERN), NA48 (CP symmetry violation studies, CERN), CAST (CERN Axion Solar Telescope, CERN) n-TOF (Neutron Time Of Flight, CERN) and it has been proposed as a candidate for the upgrade of the ATLAS Muon Spectrometer and for the forthcoming ILC as part of the hadronic calorimeter.

Micromegas is a gaseous parallel plate detector in which several innovative properties rely on a narrow amplification space (typically $50 - 100\mu m$) between two parallel electrodes: the micromesh (cathode) and the strips (anode). The challenge when constructing a Micromegas detector is to keep the gap between the electrodes constant over the whole active area. This consistency of the gap is achieved by the means of insulating pillars deposited on the cathode.

The volume of the detector is divided into two areas. The first area that a charged particle cross is the drift gap passing through the drift electrode. After the electrode the particle is in the conversion region which has a width of a few mm until the micromesh is reached. In this area, the voltage is around $1keV/cm$ and is the region that the first interaction of the particle with the detector takes place and electron-ion pairs are produced. The electrons are driven by the electric field to cross the micromesh and proceed to the next region whereas the ions are collected by the micromesh.

The region between the micromesh and the anode is called amplification gap and its width is about $100\mu m$. The anode consists of conductive microelements (strips) which are printed on a printed circuit board (PCB).

In order to get the maximum efficiency from the detector one should know the structure of the electric field close to the mesh. The homogenous field that is required for the amplification gap is easily created despite the small size of the area and consequently the desired electronic avalanche is generated. The main issue for one to be aware is the distortion of the electric field around the gaps of the micromesh. The potential lines starting from the drift electrode traverse through the mesh and end at the strips. However, their density is massively distorted near the gaps, as it is seen in figure 2.9. What this actually reveals is

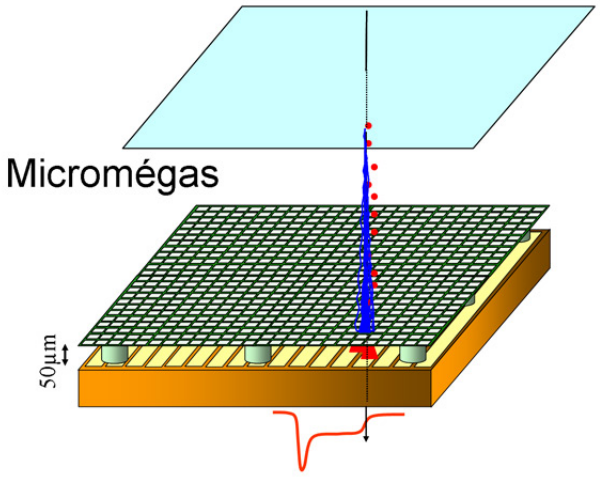


Figure 2.8: Graphical representation of an interaction of a charged particle with the detector. The interior division into two regions is apparent.

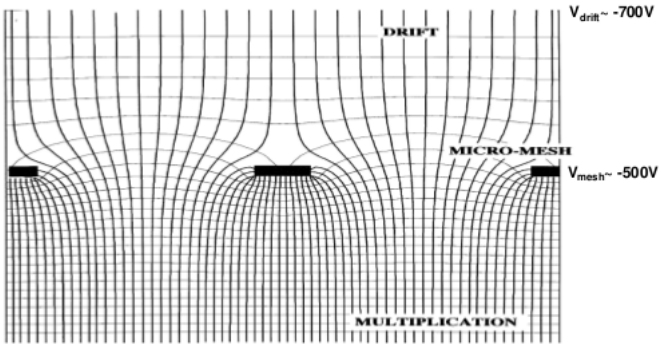


Figure 2.9: The potential lines of the electric field when passing through a GEM foil.

the difference between the intensities of the field between the two areas. The amplification when a particle crosses the mesh is negligible compared to the total one and is not taken into consideration, which is the main difference between the MicroMeGAS detector and the rest of MicroStrip Gaseous Detectors. Finally, in some cases it is convenient for the electric field in the conversion area to have increased intensity in order to generate a kind of preamplification.

The fact that the distance between the gaps of the mesh is relatively small is the reason that the longitudinal dispersion of a particle is small. Consequently, MicroMeGAS offers excellent time resolution. Furthermore, in a non-vertical track event, because of the fact that the development of the electronic avalanche is nominal to the strips and therefore the signal is conducted in a small region of readout channels resulting to an excellent spatial resolution. The latter characteristic of the detector gives the end-user the ability to use MicroMeGAS as a Time Projection Chamber by reading the strips that fired and the time the signal was collected.

The signal at the anode or the cathode is generated by the relative motion of the electrons towards the anode and of the ions towards the cathode. A typical ion signal has a response time of $100ns$ depending on the amplification gap and the gas mixture. However, the electronic signal at the anode is a lot faster

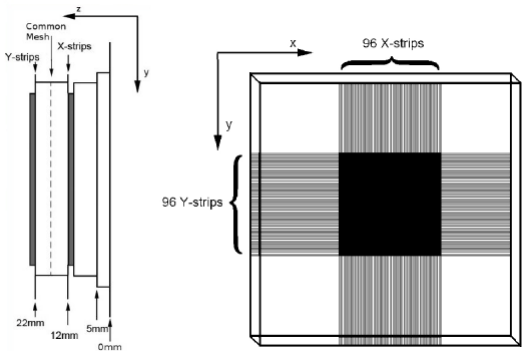


Figure 2.10: The dual set of MicroMeGAS detectors (left) and a schematic of the active area (right) of each station.

because of the high mobility μ of the electrons. A MicroMeGAS detector, because of the fast collection of ions, nullifies the effect of the space charge effect that reduces the amplification in most detectors when in operation with high frequency interactions.

The MicroMeGAS detector and MPGD in general are resistant to radiation. As the electric field is mostly homogenous in the amplification area, the development of unfortunate phenomenae, such as the polymerism during the avalanche, have a small effect.

2.5.1 The RD51 MicroMeGAS Telescope

The RD51 collaboration aims at the development of advanced gas detector technologies and the associated electronic readout system for applications in either basic or applied research.

The MicroMeGAS tracker consists of three identical double MicroMeGAS detectors that in each pair both X and Y axis are read out. When mounted on the custom made table the distance between the first and second station along beam is $338mm$ and between the second and the third $403mm$. In each station the two component detectors are mounted back to back on a common frame sharing a common drift high voltage plane but having different mesh planes. Hence each station is in need of three high voltage lines. In addition, each station is equipped with two PCB baring $10 \times 10cm$ dimentions, one to provide data for the X axis and the other, rotated by 90^o, the Y axis. Each PCB accommodate 96 strips with $250\mu m$ pitch that translates into an active area of $2.4 \times 10cm$ at each PCB and $2.4 \times 2.4cm^2$ in total for each station. As a particle travels through each station, it first crosses the X-plane and then the Y-planes.

2.6 Trigger Logic

Having presented all the parts of which the hardware consists of, as the last part of this chapter the trigger logic is repeated in short.

The experimental setup consists of the rack that accommodates the NIM and VMEbus crates, two sets of two plastic scintillators, a MicroMeGAS telescope station and two Gassiplex cards. The beam hits the first set of the plastic scintillators which produce a logic signal. Both these signals pass through the discriminator unit. This unit produces a logic pulse signal if the input signal in each line is greater than the preset threshold. If both scintillator signals are over the threshold the logic signals produced by the discriminator are fed into a coincidence unit. If the two signals from the discriminator are in timing considence the LeCroy 465 Coincidence Unit outputs the first reference logic signal.

[2] 96 strips $\times 250\mu m = 2.4cm$.

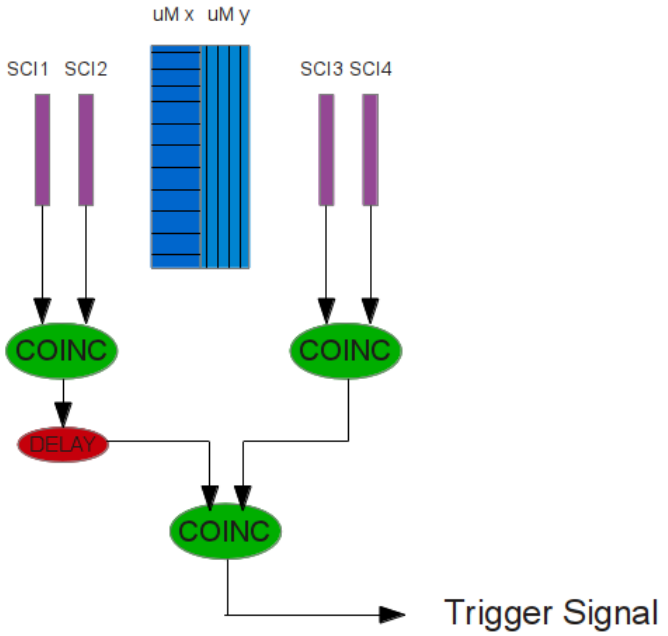


Figure 2.11: Trigger logic.

After crossing the two MicroMeGAS detectors, the beam collides with the second set of plastic scintillators. Following the above procedure the second reference signal is produced. The two reference signals are fed into the coincidence unit and the logic outcome of the unit is passed to the V551 Sequencer module as the `TRIGGER` signal.

When fed with the `TRIGGER` signal the Sequencer module sends out the `HOLD` control signal to the front-end chip after a preset t_1 delay and raises the `BUSY` signal. While the `BUSY` signal is raised the module starts the `CLOCK` and `CONVERT` sequence of control signals storing the read values into the C-RAMS module buffers. These control signals are part of the flat cable lines that starting from the sequencer module are passed to the custom-made LAPP module that translates the flat cable lines into LEMO cable outputs. These outputs are then passed to a Fan-In-Fan-Out unit that multiplies the number of outputs from one (the one coming from the sequencer) to the desired number of output signals. The LEMO cables from the Fan-In-Fan-Out unit are driven into the Gassiplex front-end chips. When the sequence ends, the `HOLD` signal goes down and the `DRDY` control signal is raised which is the trigger for the read out system to start reading the buffers of the V550 modules. Finally, after the buffers are read, they are cleared and the `BUSY` signal is lowered in order to be ready for the next `TRIGGER` signal.

CHAPTER 3

The Software

3.1 Introduction

The main tools that used throughout the software that was developed are presented in this chapter. The core of the software is a pure $C++$ code. This core includes the drivers for the modules as well as the communication with the VME crate via the optical link of the CAEN Controller, for which the Interface of the CAEN library of VME Bridges (`CAENVMELib`) is needed. The user interface of the software was written in *Qt*4 and for the histograms in the online display *ROOT* was used.

Figure 3.1: Logos of the Framework tools that were used throughout the development of the software.

3.2 CAENVME Library

The interface of CAEN library for VME Bridges is free to download from the website of CAEN[1] and it includes the main types and functions provided via VME and the modules of CAEN. The available software contains a folder in which three C headers (*./include/CAENVMElib.h*, *./include/CAENVMEoslib.h* and *CAENVMEtypes.h*) are included, as well as the shared library file (*libCAENVME.so*). Finally, some samples and demo programs are included to be used for reference.

[1] http://www.caen.it/csite/CaenProd.jsp?parent=43&idmod=689

3.2.1 CAENVMElib.h

This header file defines integer and unsigned integer types of 8, 16, 32 and 64 bits long as well as a new pre-compiler macro in order for the developer to define if C or C++ will be used. Also the main CAEN functions are defined in this file. The some of the functions included in this header file are presented in the table below, the complete list of functions can be found in appendix 1002.

Function Prototype	**Description**
CAENVME_DecodeError(CVErrorCodes Code)	Decodes the error returned by CAEN functions.
CAENVME_SWRelease(char *SwRel)	Permits to read the software release of the library.
CAENVME_BoardFWRelease(int32_t Handle, char *FWRel)	Permits to read the firmware release loaded into the device.
CAENVME_DriverRelease(int32_t Handle, char *Rel)	Permits to read the software release of the device driver.
CAENVME_DeviceReset(int32_t dev)	Permits to reset the device.
CAENVME_Init(CVBoardTypes BdType, short Link, short BdNum, int32_t *Handle	The function generates an opaque handle to identify a module attached to the PC.
CAENVME_End(int32_t Handle)	Notifies the library the end of work and free the allocated resources.
CAENVME_ReadCycle(int32_t Handle, uint32_t Address, void *Data, CVAddressModifier AM, CVDataWidth DW)	The function performs a single VME read cycle.
CAENVME_WriteCycle(int32_t Handle, uint32_t Address, void *Data, CVAddressModifier AM, CVDataWidth DW)	The function performs a single VME write cycle.
CAENVME_FIFOBLTReadCycle(int32_t Handle, uint32_t Address, void *Buffer, int Size, CVAddressModifier AM, CVDataWidth DW, int *count)	The function performs a VME block transfer read cycle. The Address is not incremented on the VMEBus during the cycle.
CAENVME_FIFOBLTWriteCycle(int32_t Handle, uint32_t Address, void *Buffer, int size, CVAddressModifier AM, CVDataWidth DW, int *count)	THe function performs a VME block transfer write cycle. The address is not incremented during the cycle.
CAENVME_IRQCheck(int32_t Handle, CAEN_BYTE *Mask)	The function returns a bit mask indicating the active IRQ Lines.
CAENVME_IRQEnable(int32_t Handle, uint32_t Mask)	The function enables the IRQ lines specified by Mask
CAENVME_IRQDisable(int32_t Handle, uint32_t Mask)	The function disables the IRQ lines specified by Mask.
CAEVME_IRQWait(int32_t Handle, uint32_t Mask, uint_t Timeout)	The function wait the IRQ lines specified by Mask until one of them raise or timeout expires.

Table 3.1: The functions included in the CAENVMElib.h file and are used throughout the software.

This header file is used in the definition and declaration of the drivers of the modules that were written.

3.2.2 CAENVMEoslib.h

This header file is a simple pre-compiler command which if the programmer defines that the operating system is Windows include a couple of headers that contain Windows-specific declaration for all the functions in the Windows API[2] are included.

3.2.3 CAENVMEtypes.h

The last header file of the CAENVME library is a set of definitions of enumerated types[3] that are used throughout the software. The enumerations that are defined are presented in the table below.

Enumeration	**Description**
CVBoardTypes	Selection of CAEN boards models.
CVDataWidth	Masking for data modifier.
CVAddressModifier	Masking for the address.
CVErrorCodes	The return value of CAEN functions.
CVPulserSelect	Selection of pulser A or B.
CVOutputSelect	Selection of output line.
CVInputSelect	Selection of input line.
CVIOSources	Selection of I/O source (button, Input lines, coincidence, VMEbus signals, internal signals).
CVTimeUnits	Time units.
CVLEDPolarity	Selection of level in which the LED emits signal.
CVIOPolarity	Selection between normal or inverted polarity.
CVRegisters	Masking of the addresses of registers.
CVStatusRegisterBits	Masking of Status register bits.
CVInputRegisterBits	Masking of Input register bits.
CVOutputRegisterBits	Masking of Output register bits.
CVArbiterTypes	Selection between Priority and Round-Robit Arbiter.
CVRequesterTypes	Selection between Fair and On demand bus requester.
CVReleaseTypes	Selection between Realese-When-Done and Release-on-Request requester.
CVBusReqLevels	Masking of the bus request levels.
CVIRQLevels	Masking of the Interrupt Request Levels.
CVVMETimeouts	Selection between $50\mu s$ or $400\mu s$ timeout.
CVDisplay	Masking of the front display.

Table 3.2: Enumerations in CAENVMEtypes.h header file.

3.3 Trolltech's Qt

Qt is a comprehensive $C++$ application developmnent framework for creating cross-platform Graphical User Interface (GUI) applications using a "write once, compile anywhere" approach. Qt let programmers

[2]The Windows API is a Microsoft's core set of application programming interfaces (APIs) available in the Microsoft Windows Operating Systems. It was formely called the Win32 API, however the name "Windows API" more accurately reflects its roots in 16bit Windows and its support on 64bit Windows.

[3]In computer programming, an enumerated type is a data type consisting of a set of named values called enumerators. The enumerator names are usually identifiers that behave as constants in the language. It is commonly used as a mean to provide a user friendly mask for a variable of "unfriendly" type (i.e. bit masks).

use a single source tree for applications that will run on Windows 98 to 7, Max OS X, Linux, Solaris, HP-UX and many other versions of Unix with X11. The Qt libraries and tools are also part of the Qt-Embedded Linux, a product that provides its own window system on top of the embedded Linux.

The Meta-Object Compiler (moc) is the program that handles Qt's $C++$ extensions. Each Qt class should has in its definition the `Q_OBJECT` macro, as the moc tool reads a $C++$ header file and if it finds one or more class declarations that contain the `Q_OBJECT` macro, it produces a $C++$ source file containing the meta-object code for those classes. Among other things, meta-object code is required for the signals and slots mechanism, the run-time type information and the dynamic property system.

A graphical module of Qt is called widget. Each widget has its own header file, which should be included in the source code and the name of the header is the letter Q followed by the name of the widget. For example, if one would like to add a push button to a GUI, one should include the header `QPushButton.h`.

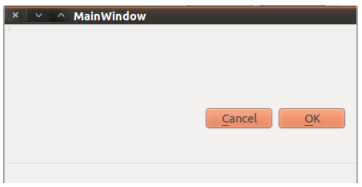


Figure 3.2: A simple main window. The widgets that are used are: QPushButons, QMainWindow, QMenuBar, QToolBar and QStatusBar.

3.4 Graphical User Interface

For the software that was developed and it is presented in this thesis, Qt handles the graphical implementation of the configuration handler of the core program, as well as it provides the framework on which the online display is build. The class responsible for the GUI is called Interface (`Interface.h`, `Interface.cpp`, `ui_interface.h`) and the source code can be found under the `src/` folder of the accompanying software file tree.

Displayed in 3.4 is the main window of the GUI. Currently active tab is the configuration tab. One can see that the main window seems plain and clean and most importantly the main widgets are self-explanatory. It consists of a QTabWidget which is the tabbed environment that houses the four tabs of the software (Online Display, Configuration, Log Viewer and About) and a set of QPushButtons that signal a variety of functions to be called and communicate with the crate or close the program. All over the GUI there are widgets whose name has an underlined letter. This would imply that there is a keyboard shortcut to quickly use this widget without making use of the mouse. The current template of the keyboard shortcuts mechanism is that the widget is activated (clicked(), selected(), etc...) when the key sequence `ALT+(Underlined Letter)` is pressed. For example, the Initialize button has underlined the letter "i" which means that it can be clicked using the sequence `ALT+i`.

3.4.1 Display Tab

Under the display tab a ROOT Canvas is embedded and if selected, a number of hitmap histograms that equals the number of the detectors that are available [4]. In each data acquisition cycle (DAQ cycle) the end-user can choose a number with which the histograms would be refreshed. This number is called `Display Refresh Rate` and it is counted in events. In other words, if the end-user defines the configuration variable `Display Refresh Rate` to be equal to 1000, at the first event the histograms would present themselves and then every 1000 events it would be refreshed to the values of the current event.

[4] Actually, the number of histograms is double the number of C-RAMS

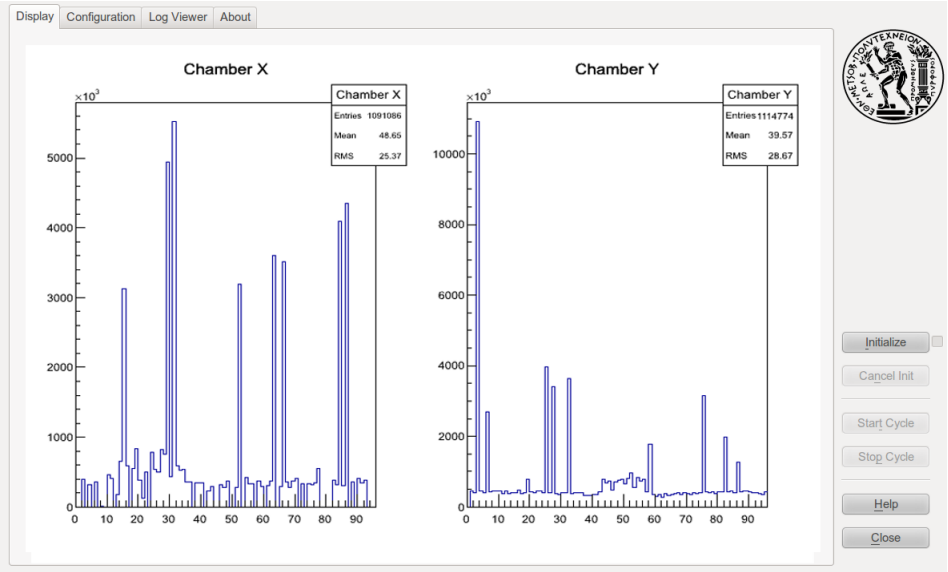


Figure 3.3: Snapshot of the display tab.

The embedded canvas is dynamically divided into pads whose number is equal to twice the number of the C-RAMS available. This happens because each C-RAMS has two FIFOs, so the software assumes that the number of the detectors is the same as the number of the FIFOs. The titles of the histograms are also set dynamically according to the relevant placing into the FIFO channels. For example a detector connected to the second FIFO of the third C-RAMS would be labelled as `"Chamber 6"`.

3.4.2 Configuration Tab

Much attention was drawn to the fact that the end user should easily change the configuration of the readout software according to the modules at hand, the way that the readout cycle would conclude or even the markers of the output file in order to much their own framework, without messing with complex configuration files or writing extra code.

The Configuration tab is divided into seven parts. Starting from the top left corner, the user can verify or change the base address of the Sequencer and the time delays and duration for the multiplexing cycle. Right next to the Sequencer configuration is the C-RAMS configuration. Here the end-user can verify or change the number of C-RAMS that he is going to include, making use of the spin-box, as well as set their base addresses. Next is the Gate Generator group box. If a Gate Generator is included in the VME setup.

Just below of these, there is the configuration of the readout markers. These special words specify the start and the end of each block of readout format. Then the run number and the run type boxes are present to configure the name of the run in a numerical way. At last, there is a misc section where the number of readout channels of the front-end chip can be modified, select if the run would end when a predefined number of events is reached, change the time in milliseconds that the VME bus will wait for an interrupt during trigger awaiting, set the paths for the configuration and mapping file, set the refresh rate of the display tab and finally if the end user wants a root file with the total hitmaps of his run.

The use and the meaning of each widget will be now explained separately.

♢ **V551 Sequencer:**

- *Sequencer Base Address:* This line edit is filled with the base address of the sequencer module. The base address can be set via four rotary switches on the board of the module. The master module of the VME crate, which has its own base address, will recognize the sequencer with the set base

address as its slave and send commands directly to it. Also, as presented in the previous chapter each register of the module is recognized via a preset value added to its base address. It is strongly recommended to use the UNIX format for hexadecimal numbers to set the base address and it is necessary the input to be in TCP/IP endianess (Big Endianess).

- *Trigger to Hold Delay:* This spinbox sets the $T1$ register of the sequencer module. The value that is set does not evaluate to nanoseconds directly. It corresponds to the time between the rise of the trigger pulse until the hold signal is raised. The spinbox is locked by default and only expert users should edit it as it changes the timing configuration between your detector signals and the VME control signals.
- *Hold to Sequence Delay:* This is the delay between the fall of the trigger pulse until the first rise of the clock pulses sequence. This spinbox is also locked.
- *Active Clock Duration:* The $T3$ register corresponds to the width of the `CLOCK` pulses. This of course means the time between the rise and the fall of the pulse. The spinbox is locked.
- *Period if Clock and Convert Sequence:* This is the time interval between the rise of one `CLOCK` pulse and the rise of the following. The spinbox is by default locked.
- *Clock and Convert Delay:* This spinbox configures the $T5$, which is the delay between the rise of a `CLOCK` pulse and a rise of a `CONVERT` pulse. The widget is locked.

♢ **Readout Markers:**

- *Header Begin Marker:* This is the hexadecimal representation of a specific number that represents the beginning of the header block in the output binary file. By default the value is set to 0x90000000, which in decimal representation stands for 2415919104. The line edit is locked by default and only expert users should modify its value.
- *Header End Marker:* The value that marks the end of the header block in the output file. By default it is 0x9ffff000 which is 2684350464 in decimal. The widget is also locked.
- *Event Data Begin Marker:* Each event data block in the output file starts with this word. Its default value is 0x80000000 or 2147483648 in decimal. The line edit is by default locked.
- *Event Data End Marker:* The end of each event data block is marked by this word. Its default value is 0x8ffff000 or 2415915008 in decimal. The modification of the value is locked.
- *Footer Begin Marker:* Each file closes with the last block being the footer block. The footer block starts with this word, whose default value is 0xa0000000 or 2684354560 decimal. The widget is locked.
- *Footer End Marker:* The end of the footer is marked by the footer end marker. The default value for this is 0xaffff000 or 2952785920 in decimal. The widget is also locked.

♢ **Run Information:**

- *Run Number:* The spinbox contains the number of the run that is about to start. It will mark also the name of the file that is produced during the DAQ cycle. The spinbox is updated when the program is launched and when a DAQ cycle is ended, when the number in the spinbox is incremented by one.
- *Type of Run:* This combo box defines the type of the run that is about to start. There are three possibilities: Physics, Pedestal and Test. The type of the run is also apparent in the name of the output file. Each data file that is produced is named as :

$$run[RunNumber][RunType].dat$$

If the run type is physics then the file name will be marked with a "D" character. If the type is pedestal then that character is a "p". Finally if the end-user is about to start a test run the special character is "T". This means that a physics run with number 9011 will produce an output file named `run9011D.dat`.

♢ **CRAMS:**

- *Number of CRAMS:* The spinbox holds the value of the number of C-RAMS the end user desires to use. Whenever the spinbox value is changed a line edit from the list below is either enabled or disabled accordingly.
- *C-RAMS i:* The line edit below each C-RAMS label is used to set the base address for the corresponding C-RAMS module. It is recommended to use UNIX format when inputting the address and use TCP/IP endianess. The number of the enabled (white background and editable) line edits is equal to the number in the number of C-RAMS spinbox and this property is modified dynamically via it.

♢ **Gate Generator:**

- *Is there a Gate Generator? :* If the state of this checkbox is checked then the software will recognize a V462 Dual Gate Generator module on the VME crate and will initialize it and use it properly.[5]
- *Base Address:* This line edit is the modules base address, in order to be recognized as a slave to the VME controller Master. It is strongly recommended UNIX format and TCP/IP endianess input. The widget is enabled only if the checkbox is checked.
- *Channel 0: Upper Display:* This is to set the lower display of the first channel on the gate generator that corresponds to the gate width of the first channel. The widget is enabled only if the checkbox is checked.
- *Channel 0: Lower Display:* This is to set the lower display of the first channel on the gate generator that corresponds to the gate width of the first channel. The widget is enabled only if the checkbox is checked.
- *Channel 1: Upper Display:* This is to set the upper display of the second channel on the gate generator that corresponds to the gate width of the first channel. The widget is enabled only if the checkbox is checked.
- *Channel 1: Lower Display:* This is to set the lower display of the second channel on the gate generator that corresponds to the gate width of the first channel. The widget is enabled only if the checkbox is checked.

♢ **Misc:**

- *Number of Readout Channels:* The value of the spinbox corresponds to the number of the detector readout channels. It is used as an upper limit on the size of the readout vectors throughout the software. The widget is by default locked, as the software was designed to work with Gassiplex front end chip which houses 96 readout channels.
- *Maximum Number of Events:* This checkbox defines the conditions under which the current run will end. If it is checked then the spinbox next to it should be modified to point to the maximum number of events that should be collected in order for the run to stop. For example, if the checkbox is checked and the spinbox value is 5000, then the DAQ cycle will start and after 5000 events will stop. If the checkbox is unchecked then the cycle will start and will not be ended until the Stop Cycle button is pressed.
- *Configuration File Path:* This line edit holds the full path to the configuration file. The default configuration file is placed inside the `inputFiles` folder in the installation folder tree and it is named `daq.conf`.
- *Mapping File Path:* This line edit holds the full path to the mapping file. The mapping file contains two columns of numbers. The first column represents the ADC Channel number, while the second the strip number. The default mapping file is saved under the `inputFiles` folder in the installation directory and is named `Mapping.txt`[6]

[5] A word of warning: If a gate generator is used, the timings of the Sequencer module should probably be modified.

[6] The default mapping file was provided by the group working on micromegas at LAPP (*Laboratoire d' Annecy-le-Vieux de Physique des Particules*).

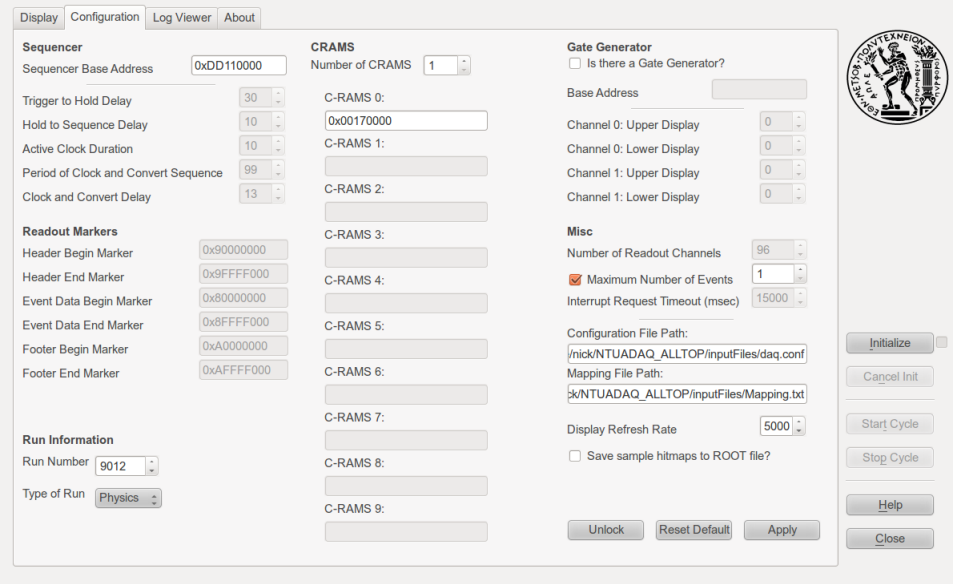


Figure 3.4: Snapshot of the configuration tab.

- *Display Refresh Rate:* The number in the spinbox is the rate that the display of the histograms under the display tab will be refreshed. This should be modified according to the current event rate. A small number of refresh rate and a huge number of events will force the Graphical User Interface to crash, as the thread controlling the display would fail to alter the graphics of the main thread too fast as the ROOT plotter is an external library.
- *Save sample hitmap to ROOT file? :* If the checkbox is checked when the run will end a root file containing some histograms with a total hitmap on the events should be produced.

♢ **Buttons:**

- *Unlock/Lock:* If `Unlock` is pressed the locked widgets under the configuration tab unlock, becoming enabled and editable, while the label of the button changes to `Lock`. If pressed again the same widgets will lock and the label will change back to `Unlock`.
- *Reset Defaults:* If this button is pressed the software will read again the default configuration file and reset all values of the widgets to their default settings.
- *Apply:* If any change to any widget under the configuration tab is made will not be applied if the `Apply` button is not pressed. When this button is pressed the software sets the user-defined values to the parameters needed by the DAQ cycle.

3.4.3 Log Viewer

The log viewer tab holds tracking information of the conditions and the warnings of the software. Each entry in the log is tagged by a timestamp and a qualifier to assert the importance of it. There are three types of qualifiers: `INFO`, which is information about the process of the graphical interface, `WARNING`, that holds information of issues that may or may not be crucial to the execution of the software and `FATAL`, which is the most serious type of qualifier and an entry of this type usually means that there is a problem with the connection to the hardware or the hard disk of the system.

To put it simply, an `INFO` entry is set when the software starts called

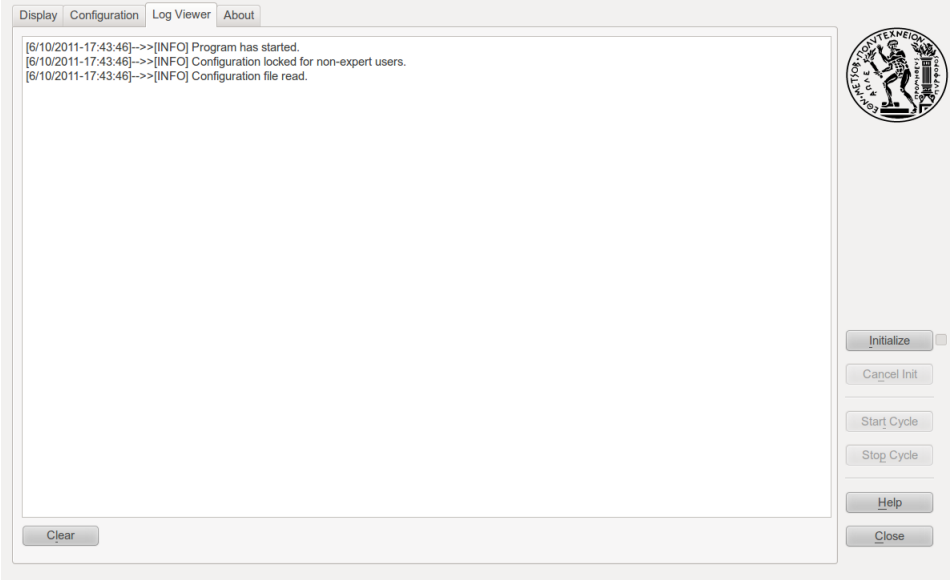


Figure 3.5: Snapshot of the log viewer tab.

`[INFO] Configuration locked for non-expert users.`

A possible warning entry is

`[WARNING] Error while generating Event Data Bottom Header. Possibility of corrupted file.`

while a fatal entry is

`[FATAL] Error while initializing controller.`

Finally, the widget houses a `Clear` button that clears the history off the log viewer if pressed.

3.4.4 About

The About tab provides information on the creator of the software and its version.

3.4.5 Cycle Buttons

The cycle buttons are the six buttons placed at the right end of the graphical interface, used to control the operations of a DAQ cycle.

- **Initialize:** When the `Initialize` button is pressed the connection between the hardware on the crate is established, the input files are read and processed and the output file is opened for writing. If the initialization is successful the checkbox next to the button changes state from unchecked to checked, a log entry is set to inform the user of the success of the operation and the `Start Cycle` button is enabled. If the initialization is unsuccessful the checkbox remains unchecked and a fatal error appears in the log viewer. There is a keyboard shortcut for this operation and it is the sequence `ALT+I`.

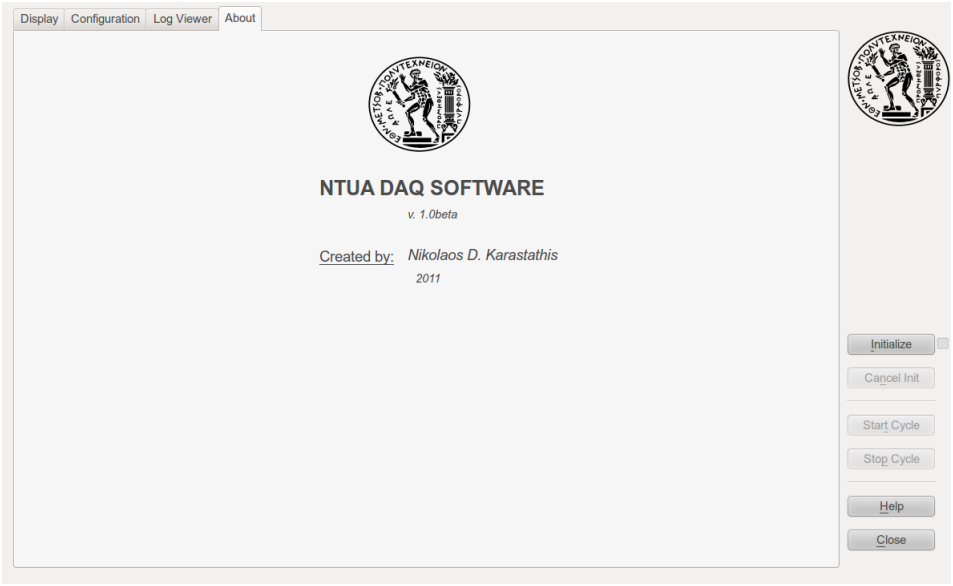


Figure 3.6: Snapshot of the about tab.

- **Cancel Init:** If for any reason the end-user would like to cancel the initialization of the modules and files, for example if there was a typographic mistake in a widget under the configuration tab, the cancel init button could be pressed. This would release the system memory of the allocated space for the modules and close every open file. If the operation is successful the checkbox should be unchecked and the `Start Cycle` button should be disabled. The shortcut sequence for this button is `ALT+N`.
- **Start Cycle:** This is the button to press in order to start the DAQ cycle. The cycle spawns a new thread to the software and proceed to start the readout sequence for the current run. Various information messages are printed in the log viewer while the run is in progress. When the `Start Cycle` button is pressed the `Stop Cycle` button becomes enabled. The shortcut sequence for this operation is `ALT+T`.
- **Stop Cycle:** If the maximum number of events is checked the run will stop when this number is reached, otherwise the end-user must use the `Stop Cycle` button to end the run. The `Stop Cycle` button enables the `Start Cycle` button. The keyboard sequence for this operation is `ALT+P`.
- **Help:** The `Help` button opens the default web browser of the operating system presenting the user to a quick documentation that provides information about the run process, the widgets of the graphical interface and some common error messages and how to face them. It is a short and practical version of this thesis available to all platforms as it is written in HTML. The main window of the documentation can be seen in Figure 3.7. The shortcut sequence for this operation is `ALT+H`.
- **Close:** This button closes the program. The keyboard shortcut sequence is `ALT+C`.

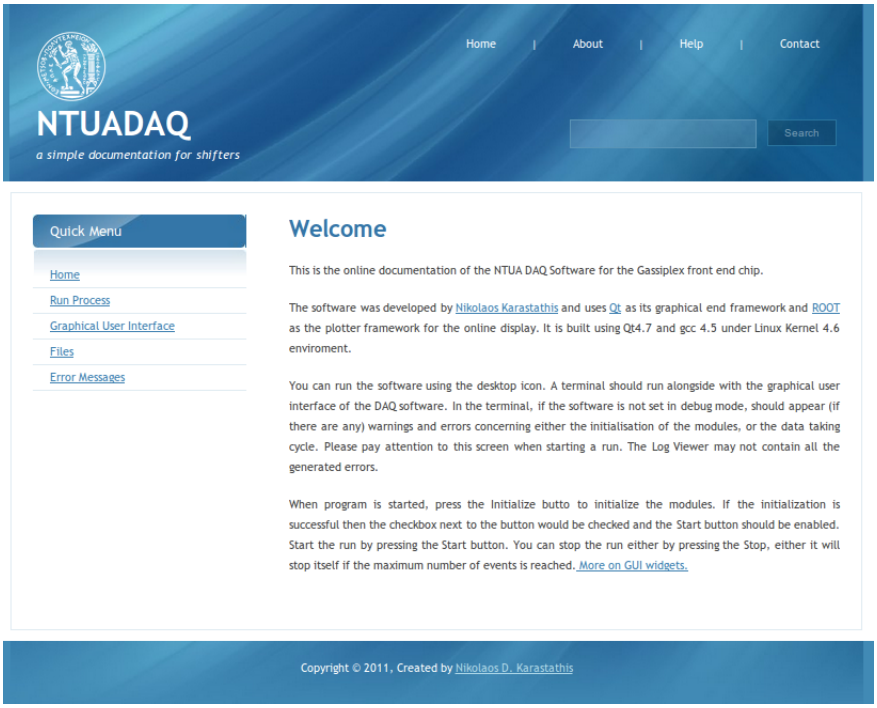


Figure 3.7: Snapshot of the documentation page.

3.5 The Source Code

The installation directory of the software has the structure that is presented in Figure 3.8.

3.5.1 src/V550_CRAMS

The two files (`V550_CRAMS.h` and`V550_CRAMS.cpp`) are the definition and the declaration of the driver functions written for CAEN V550 C-RAMS module.

- **V550_CRAMS(int32_t ctrlHandle, uint32_t bAddress)**
  This is the constructor function for a V550 module. It takes as arguments the controller ID to recognize it as its own Master and the base address of the module.
- **CVErrorCodes ReadModuleFixedCode(uint16_t &moduleFixedCode)**
  This function reads the Fixed Code register of the V550 module and stores it in a 16bit word. Its return type is CVErrorCodes which is an enumeration provided by CAEN VME library.
- **CVErrorCodes ReadModuleManufacturerSpecs(uint16_t &moduleManufacturerSpecs)**
  It is used to store in a 16bit word the specifications of the manufacturer as they are imprinted in the register of the module.
- **CVErrorCodes ReadModuleVersionSerial(uint16_t &moduleVersionSerial)**
  It reads the version and serial number of the module and stores it in a 16bit word.
- **CVErrorCodes WriteTestPatternRegister(bool whichFIFO, uint16_t testPattern)**
  A write function to the module's Test Pattern Register. Each C-RAMS V550 module houses two separate FIFOs named `FIFO 0` and `FIFO 1`. We pulled a programming trick by defining the value that points to a FIFO as a boolean type parameter. C++ evaluates every number except integer zero to `TRUE`, so the programmer can easily separate the two FIFOs via an if-else statement.

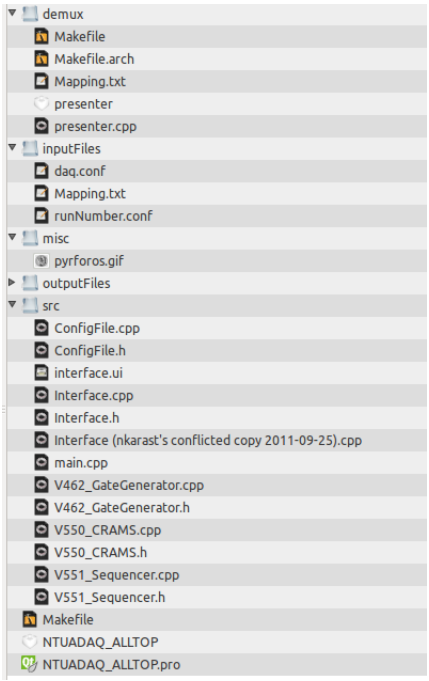


Figure 3.8: Snapshot of the file tree of the installation directory.

- **CVErrorCodes ReadWordCounterRegister(bool whichFIFO, int &words)**
  Reads the number of words that exist in each FIFO. This function is used during readout to define the number of words that are to be read from each buffer.
- **CVErrorCodes ReadFIFOBLT(bool whichFIFO, uint32_t *data, int size, int *count)**
  Reads a FIFO in Block Transfer Mode and stores the block of data in a vector of unsigned 32bit integers. The address that is to be read during a BLT read access increments automatically and its upper limit is set via the size argument.
- **CVErrorCodes ClearModule()**
  A call for this function clears the module returning each of its registers to its default set up.
- **CVErrorCodes ReadNumberOfChannels(unsigned int &numOfChansADC0, unsigned int &numOfChansADC1)**
  A read access to Number of Channels register of the module. The values returned are the number of channels divided by 32. So if both channels return that the number of channels is 3 it means that the actual number of readout channels is 96, as it is in the case of the Gassiplex front-end chip.
- **CVErrorCodes WriteNumberOfChannels(unsigned int numOfChansADC0, unsigned int numOfChansADC1)**
  Write the number of readout channels of the module. The number provided must be the number of readout channels divided by 32.
- **CVErrorCodes ReadStatusRegister(bool &testMode, bool &memOwner, bool &DRDYChan0, bool &DRDYChan1, FIFOStatus &fifo0status, FIFOStatus &fifo1status)**
  Reads the status register of the V550 module. `FIFOStatus` is an enumeration that the writer created having the values of 0, 1, 2, 3, −1 for the cases that the current FIFO is empty, not empty, half-full, full or if it is in an undefined state, accordingly.
- **CVErrorCodes WriteStatusRegister(bool testMode, bool memOwner)**

Write access to the status register of the module. Only the first two bits can be accessed in write mode.

- **CVErrorCodes WriteInterruptRegister(unsigned int statusId, unsigned int interruptLevel)**
  Writes the interrupt register of the module providing its own `STATUS/ID` code and the level of the interrupt that it is able to raise.
- **CVErrorCodes ReadPedestalThreshold(bool whichFIFO, unsigned int channel, unsigned int &pedestal, unsigned int &threshold)**
  Reads in sequential mode the buffer provided for storing the pedestal and threshold values of each channel.
- **CVErrorCodes WritePedestalThreshold(bool whichFIFO, unsigned int channel, unsigned int pedestal, unsigned int threshold)**
  Writes the pedestal and threshold values in the buffer of the module.

3.5.2 src/V551_Sequencer

The two files `V551_Sequencer.h` and `V551_Sequencer.cpp` are the driver routines for creating and controlling a CAEN V551 Sequencer module.

- **V551_Sequencer(int32_t ctrlHandle, uint32_t bAddress)**
  This is the constructor for a V551_Sequencer object. It takes as arguments the controller ID in order to be its Master and the base address of the module on the crate.
- **CVErrorCodes ReadModuleFixedCode(uint16_t &moduleFixedCode)**
  Read access to the fixed code of the module which is stored to the second argument .
- **CVErrorCodes ReadModuleManufacturerSpecs(uint16_t &moduleManufacturerSpecs)**
  Reads the manufacturer specifications imprinted in the respective register of the module and stores it in a 16bit long integer passed as argument.
- **CVErrorCodes ReadModuleVersionSerial(uint16_t &moduleVersionSerial)**
  Reads the version and the serial number of the module and stores both of them in a 16bit long integer passed as argument.
- **CVErrorCodes WriteInterruptVectorRegister(unsigned int statusId)**
  Performs a VME write cycle, setting the `STATUS/ID` value of the interrupt register of the module.
- **CVErrorCodes WriteInterruptLevelRegister(unsigned int interruptLevel)**
  Sets the interrupt level that the module would generate.
- **CVErrorCodes ClearModule()**
  A call of this function clears the module.
- **CVErrorCodes SoftwareTrigger()**
  The V551 module is capable of generating by software request a trigger signal and this function gives to the end user this capability. It is mostly used for testing and debugging purposes.
- **CVErrorCodes ReadStatusRegister(bool &internalDelay, bool &veto, bool &autoTrigger, bool &dataReady, bool &busy, bool &activeSequence)**
  Reads the pattern of the status register of the module. Then assigns each bit of the pattern to its corresponding attribute and stores them in the arguments passed during calling.
- **CVErrorCodes WriteStatusRegister(bool internalDelay, bool veto, bool autoTrigger)**
  Sets the values of the status register of the module to the ones passed through its arguments.
- **CVErrorCodes ReadTestRegister(bool &testMode, bool &clockLevel, bool &shiftInLevel, bool &testPulseLevel)**
  Reads the pattern of the test register of the module. Then assigns each bit of the pattern to its corresponding attribute and stores them in the arguments passed during calling.

- **CVErrorCodes WriteTestRegister(bool testMode, bool clockLevel, bool shiftInLevel, bool testPulseLevel)**
  Sets the values of the test register of the module to the ones passed through its arguments.
- **CVErrorCodes ReadNumberOfChannels(unsigned int &numberOfChannels)**
  Reads the number of channels assigned to the sequencer and stores the output to the variable passed into the function.
- **CVErrorCodes WriteNumberOfChannels (unsigned int numberOfChannels)**
  Writes the number of channels in the according register.
- **CVErrorCodes ReadT1Register(unsigned int &T1)**
  Reads the $T1$ register that corresponds to the delay between `TRIGGER` signal and `HOLD` control signal and stores it to the parameter passed into the function.
- **CVErrorCodes WriteT1Register(unsigned int T1)**
  Writes the $T1$ parameter of the time delay between the `TRIGGER` and `HOLD` signals. $T1$ must be a positive integer less than 255.
- **CVErrorCodes ReadT2Register (unsigned int &T2)**
  Reads the $T2$ register that corresponds to the delay between `HOLD` signal and the sequence of `CLOCK` control signals and stores it to the parameter passed into the function.
- **CVErrorCodes WriteT2Register(unsigned int T2)**
  Writes the $T2$ parameter of the time delay between the `HOLD` and the sequence of `CLOCK` control signals. $T2$ must be a positive integer between 10 and 511.
- **CVErrorCodes ReadT3Register(unsigned int &T3)**
  Reads the $T3$ register that corresponds to the width of the `CLOCK` pulses and stores it to the parameter passed into the function.
- **CVErrorCodes WriteT3Register(unsigned int T3)**
  Writes the $T3$ parameter of the width of the `CLOCK` pulses. $T3$ must be a positive integer between 1 and $T4$.
- **CVErrorCodes ReadT4Register(unsigned int &T4)**
  Reads the $T4$ register that corresponds to the period of the `CLOCK` control signals and stores it to the parameter passed into the function.
- **CVErrorCodes WriteT4Register(unsigned int T4)**
  Writes the $T4$ parameter of the time period of the `CLOCK` control signals. $T4$ must be a positive integer between 1 and 511.
- **CVErrorCodes ReadT5Register(unsigned int &T5)**
  Reads the $T5$ register that corresponds to the time delay between `CLOCK` and `CONVERT` control signals and stores it to the parameter passed into the function.
- **CVErrorCodes WriteT5Register(unsigned int T5)**
  Writes the $T5$ parameter of the time delay between `CLOCK` and `CONVERT` control signals. $T5$ must be a positive integer between 2 and 511.

3.5.3 src/V462_GateGenerator

The two files `V462_GateGenerator.h` and `V462_GateGenerator.cpp` are the driver routines for creating and controlling a CAEN V462 Dual Gate Generator module.

- **V462_GateGenerator(int32_t ctrlHandle, uint32_t bAddress)**
  This function is the constructor of the V462_GateGenerator class, creating an object requesting as arguments the controller ID and the base address of the module on the VME crate.
- **CVErrorCodes ReadModuleFixedCode(uint16_t &moduleFixedCode)**
  Reads the fixed code of the module and stores it to the 16bit-wide integer variable passed through calling.

- **CVErrorCodes ReadModuleManufacturerSpecs(uint16_t &moduleManufacturerSpecs)**
  Reads the specifications of the module as stored by the manufacturer in the respective register of the module and stores it in the variable passed through calling.
- **CVErrorCodes ReadModuleVersionSerial(uint16_t &moduleVersionSerial)**
  Reads the version and the serial number of the module as stored by the manufacturer in the respective register of the module and stores them in a pattern form in the variable passed through calling.
- **CVErrorCodes SetChannel0UpperDisplay(unsigned int upperDisplayCh0)**
  Sets the upper display of the first channel (channel 0) of a V462 Dual Gate Generator module to the integer value passed through calling.
- **CVErrorCodes SetChannel0LowerDisplay(unsigned int lowerDisplayCh0)**
  Sets the lower display of the first channel (channel 0) of a V462 Dual Gate Generator module to the integer value passed through calling.
- **CVErrorCodes SetChannel1UpperDisplay(unsigned int upperDisplayCh1)**
  Sets the upper display of the second channel (channel 1) of a V462 Dual Gate Generator module to the integer value passed through calling.
- **CVErrorCodes SetChannel1LowerDisplay(unsigned int lowerDisplayCh1)**
  Sets the lower display of the second channel (channel 1) of a V462 Dual Gate Generator module to the integer value passed through calling.
- **CVErrorCodes GenerateTestPulse()**
  A call to this function generates a test pulse that is most commonly used for test purposes.

3.5.4 src/ConfigFile

The two files `ConfigFile.h` and `ConfigFile.cpp` are the definition and declaration of the functions that read the configuration file of the software and sets the predefined values to most of the variables needed by the DAQ cycle. The class has as its member variables (see `ConfigFile.h`) all the variables configured in the configuration file and two functions that create a configuration file parser object and read the file.

- **ConfigFile(const char* filename= 0)**
  The constructor of the class. Creates a configuration file parser using as input source the filename provided through calling.
- **$\sim$ ConfigFile()**
  The destructor of the class. Frees the memory of the parser created and its variables and closes the input file.
- **int loadConfiguration()**
  Creates a `TEnv` object (ROOT class) and reads the configuration file that was set through the constructor of the class passing the values from the file to the member variables of the class.

3.5.5 src/Interface

The two files `Interface.h` and `Interface.cpp` consist the core of the software. They implement the class that creates and presents the graphical user interface and executes a DAQ cycle binding all the features of the software under one robust interface. One can easily see that in the header file of the `Interface` class there is also another class called `QRootCanvas` which creates an embedded ROOT canvas in the application and handles the events to and from it. Therefore, as far as the `QRootCanvas` class is concerned, it contains the following functions.

- **QRootCanvas(QWidget *parent = 0)**
  This is the constructor of a `QRootCanvas` object. It takes as an argument a `QWidget` of the Qt application, where the `TCanvas` of ROOT would be embedded. In particular, it gets the window id of the `QWidget` passed through calling and registers it to the `VirtualX`, which is ROOT's generic interface to the underlying, low level graphics system (X11, Win32, MacOS). Then a `TCanvas` object is created having as parent the `QWidget` that was just registered as a child to `VirtualX`.

- **virtual QRootCanvas()**
  The destructor of the class. Deletes the objects created by the constructor and frees the respective heap memory.
- **mouseMoveEvent(QMouseEvent *e)**
  Handles the mouse movement events used on the embedded `TCanvas`. It passes the Qt events concerning the mouse movements to the ROOT event handler.
- **mousePressEvent(QMouseEvent *e)**
  Handles the mouse clicks used on the embedded `TCanvas`. It passes the Qt events concerning the pressed state of the mouse buttons to the ROOT event handler.
- **mouseReleaseEvent(QMouseEvent *e)**
  Handles the mouse clicks used on the embedded `TCanvas`. It passes the Qt events concerning the released state of the mouse buttons to the ROOT event handler.
- **resizeEvent(QResizeEvent *e)**
  Handles the resize events of the embedded `TCanvas`. Upon a raised event via Qt the ROOT event handler resizes the canvas and then updates it.
- **paintEvent(QPaintEvent *e)**
  Handles the paint events of the embedded `TCanvas`. Upon a raised event via Qt the ROOT event handler updates the canvas.

Then the main class description in those two files concern the `Interface` class. As it is a GUI class it contains both conventional functions and slot function for the widget it contains. The list of the function members of the class is the following.

- **explicit Interface(QWidget *parent = 0)**
  This is the constructor of the class. It builds and sets up the graphical interface, reading the configuration file and loading its values to the widgets of the main window. Additionally, it sets the `TCanvas` and its timer, initializes a variety of global variables to their respective values and connects the various signals of the graphical widgets to their respective slot functions.
- **Interface()**
  The destructor of the class. It deletes the user interface in its whole freeing the memory allocated for it.
- **void demux(uint32_t, int &debugChan, int &debugData)**
  Takes a 32bit wide unsigned integer that corresponds to the FIFO register pattern and decyphers the channel number (ADC channel number) and its respective ADC value storing them in the two last variables passed through calling.
- **void lockExpertMode()**
  Locks the sensitive configuration parameters disabling the respective modules. The modules are enabled when the Unlock button is pressed.
- **void lockGateGeneratorConf()**
  Locks (disables) the graphical widgets that correspond to the Dual Gate Generator module, if the checkbox is unchecked. The modules are enabled when the state of the checkbox is `Qt::Checked()`.
- **void readConfFile()**
  Reads the configuration file creating an object of the `ConfigFile` class. The default configuration file is called `daq.conf` and is under the `inputFiles/` subfolder. This function is called when the UI sets up, so all the widgets will be filled with the values from the default configuration file at first. In order to change the configuration file, the user may edit the file or choose another path to the configuration file.
- **void setUpConfigurationTab()**
  Fills the configuration widgets with the values read from the configuration file.
- **void setUpCRAMSLineEdits()**
  Sets dynamically the line edits for the base addresses of the C-RAMS modules.

- **std::string getTimeStamp()**
  Returns a string with the current date and time in the format `DD/MM/YYYY−hh:mm:ss`.
- **void setLogEntry(const char* entry)**
  Appends to the log viewer the timestamp given by the above funtion followed by the argument of the current function.
- **int initialize()**
  Performs the initialization of the DAQ cycle. In detail, it commences the connection with the controller (Master of the VME crate). Afterwards, if the connection was established successfully it opens the input files (configuration and mapping), creates the output data file, initializes the rest of the hardware. During these operations the log viewer is updated with information or error messages and the flow of the program is either continued or stopped according to the existance and the severity of the errors. The function is an integer type in the sense that a return integer of zero value signs a successful operation where a negative value signs an occured error. This notation is used throughout the software.
- **int initializeHardware()**
  Initializes the hardware (C-RAMS, Sequencer and Gate Generator) according to the values and options selected via the configuration file or the graphical interface. The function is called by the `initialize()` function and during its execution a variety of validity checks on either the connection or the successful initialization of the registers is performed.
- **int initializeOutputFiles()**
  Opens the output file in binary mode. If the output file cannot be opened an error is flagged and the run stops.
- **int initializeInputFiles(const char* mapFilename)**
  Initializes the input (mapping) file. Mapping file is a two-column ASCI file in which the first column represents the ADC channel number and the second the respective strip number. This function creates a map structure between ADC channel values and the strip numbers.
- **void startDAQCycle()**
  Starts a DAQ cycle waiting for a trigger to arrive. If the trigger arrive before the predeterminded timeout interval it starts the readout cycle reading the components of the FIFOs of the C-RAMS.
- **int waitForTrigger(command &curCommand)**
  Masks a VME interrupt line and sets a timer on it. If an interrupt signal is raised before the timeout of the interval then the variable passed through calling changes into `TriggerArrived` and the readout cycle starts. Otherwise the variable is changed to `Timeout` and the cycle of masking a line and waiting starts again.
- **int readout()**
  Reads dynamically the FIFOs of all the C-RAMS on the setup. The function uses a BLT mode for each FIFO and stores their data in a temporary vector that is written into the output file in hexadecimal form. Furthermore, this function fills the histograms in the Display tab with the hitmaps of selected events. Finally, if selected by the user, this function also produces the ROOT files of total hitmaps.
- **int finalize()**
  Finalizes the DAQ cycle by stopping the loop, writing and closing the output files, incrementing run number index and refreshing the GUI with the new values.
- **int generateHeader()**
  Writes the header of the output file. The header consists of six 4-byte words:
  - Header begin, a preset unique bit sequence that marks the start of the header.
  - Run number
  - Date the run started in integer form (e.g. if the date was 12th of December 2011 the number would be 12122011 in decimal base).

  - Time the run started in integer form (e.g. if the time was 12:13:14 the number would be 121314 in decimal base).
  - Type of run. This is an integer identifier that specifies if the run is Physics run, Pedestal run or Test Run.
  - Header end, a preset unique bit sequence that marks the end of the header.

- **int generateEventDataTop()**
  Writes the top part of the Event Data block of the output file. The event data blocks in a file should be numerically equal to the total number of events, meaning that there is one event data block per event. The top part of the Event Data block consists of three 4-byte words:
  - Event begin, a preset unique bit sequence that marks the start of the event data block.
  - Event number, an integer that identifies the event in the run.
  - Start time of the event in the usual integer form as described above.

  The block of Event Data consists of the top part and then for each FIFO a number identifying the C-RAMS module, one identifying the FIFO in the C-RAMS, one stating the number of 32bit words the FIFO consists of and finally the vector of multiplexed words in the fifo. Then the Event Data block is finalized by the Event Data end marker.
- **int generateEventDataBot()**
  Writes the final part of the Event Data block in the output file. This part consists of the Event Data end marker, a preset unique bit sequence that marks the end of the Event Data block.
- **int generateTrailer()**
  Writes the trailer block in the output file. This block consists of five 4-byte words:
  - Trailer begin, a preset unique bit sequence that marks the start of the trailer block.
  - Number of event in the run.
  - End date of the run in the usual integer form.
  - End time of the run in the usual integer form.
  - Trailer end, a preset unique bit sequence that marks the end of the trailer block.
- **void handleRunNumber()**
  A utility routine that reads the current run number and displays it on the GUI of the software and when each run is concluded the run number is increased. The need of this function is apparent if one wonder how one could keep track of a running number during the whole time that the software is installed on the system.
- **void handleOutputFilename()**
  Another utility function that generates the complete name of the output file. As mentioned above, the filename consists of the constant string `run`, the number of the run and a character identifier discriminating between physics, pedestal and test runs.
- **void getReadoutVectorValues(uint32_t inputVector[])**
  Gets the values of the readout vector for the current event and for each FIFO. This function is used in order to pass the information of the current event to the display functions.
- **void HistoFill()**
  Creates and fills the histograms displayed in the Display tab with the information passed via getReadoutVectorValues() function.
- **void displayCycle(std::vector<uint32_t> vec, int whichFifo)**
  Runs a parallel thread (QtConcurrent::Run(), asynchronous thread) for the display cycle getting as input the vector of the data for the current event and an identifier for the current FIFO.
- **void startDisplayThread(std::vector<uint32_t> vec, int whichFifo)**
  Spawns the parallel thread (QtConcurrent::Run(), asynchronous thread) for the display cycle.

- **SLOT: void unlockExpertMode()**
  Public slot that responds to the signal that the `Unlock` push button emits. Enables the disabled widgets on the configuration tab and changes the name of the button from Unlock to Lock, so that if the button is pressed again the widgets would restore their previous state.
- **SLOT: void unlockGateGeneratorConf(int state)**
  Public slot that responds to the signal that the `Is there a Gate Generator` checkbox emits. If the state of the checkbox is `Qt::Checked()` then the widgets that correnspond to the Gate Generator module become unlocked and the driver routines for the module are loaded.
- **SLOT: void resetDefaultConfig()**
  Public slot that responds to the signal that the `Reset Default` push button emits. If the button is pressed the GUI reads again the default configuration file and restores the values of the widgets.
- **SLOT: void enableMaxNumberEvents(int state)**
  Public slot that responds to the signal that the `Maximum Number of Events` checkbox emits. If the state of the checkbox is `Qt::Checked()` then the spinbox next to it is enabled and the future run will conclude when the predefined number of events is reached.
- **SLOT: void applyConfiguration()**
  Public slot that responds to the signal that the `Apply` push button emits. If the push button is pressed the values set to the widgets of the configuration tab are stored to the variables of the DAQ cycle. This step is required in order to initialize the modules and start a run.
- **SLOT: void loadConfifuration()**
  A slot function that is called throughout various cases during the software GUI setup. This is actually the function that fills the widgets of the configuration tab.
- **SLOT: void changeNumberOfCrams(int newVal)**
  Public slot that responds to the signal that the `Number of C-RAMS` spinbox emits. If the value of the spinbox is changed into a new value the equal number of C-RAMS addresses widgets are only enabled, protecting the validity of usage of pointers towards object of the classes of the modules.
- **SLOT: void startDAQThread()**
  Public slot that responds to the signal that the `Start Cycle` push button emits. The slot spawns a parallel asynchronous thread that handles the DAQ cycle in such a way that the GUI would not freeze when the data acquisition algorithm is in progress. Furthermore, this slot disables the `Initialize` and enables the `Stop Cycle` push button.
- **SLOT: void initializeDAQ()**
  Public slot that responds to the signal that the `Initialize` push button emits. The slot initializes the connection of the system with the VMEbus crate, the modules of the crate and the input and output files. In addition, it enables the `Cancel Init` and `Start Cycle` push buttons. If the initialization is successful the checkbox next to the push buton changes its state to `Qt::Checked()`.
- **SLOT: void cancelInitialization()**
  Public slot that responds to the signal that the `Cancel Init` push button emits. The slot cancels the initialization of the modules and output files by deleting the pointers assigned to the modules and the newly created output data files.
- **SLOT: void stopDAQCycle()**

3.5.6 src/main.cpp

This is the main function of the software. It creates both a TApplication for the event handler of ROOT and a QApplication for the one of Qt's. It finally creates an object of the `Interface` class and presents it to the end user.

3.5.7 inputFiles/daq.conf

This is the default configuration file that holds the values needed by the DAQ cycle algorithm. The values are set in an easy to read form for the end user separating a key from its coresponding value with the use of a colon. This file is recommended to be edited only by adapt users and only if the default configuration is changed permanently to another.

3.5.8 inputFiles/runNumber.conf

This file holds the run number for the current system. By default the first run is identified as run 1. If for any reason, one desires to change the run number, one should change this number from the GUI and not from the specific file. A manual change to the file does not produce any logical errors for the software as it is set for the archiving purposes of each end-user.

3.5.9 inputFiles/Mapping.txt

This file holds a mapping table between strip number and ADC channel number in the ADC module. The channels in the FIFO of the C-RAMS do not correspond one-to-one the readout channels (strip) and therefore a transformation between them should be performed.

3.5.10 demux/

This folder contains a short script that reads the output file in a serial way and plots some basic histograms containing the ADC values per strip with or without a user-set cut, as well as some strip efficiency plots.

3.5.11 outputFiles/

This is the folder that the produced output files of raw data are saved.

3.6 Typical Run Process

The typical run procedure to acquire the raw data of a run should be described in short in this section. First of all, the end-user should start the software by executing the command

```
./NTUADAQ
```

Then the configuration of the run should be processed. Check the validity of the default configuration, mainly the base addresses of the modules, the type of run and the way that the run should end (either by reaching a maximum number of events or by pressing the `Stop Cycle` push button.

If the configuration is valid, the user should press the `Apply` button to store the values of the widgets to the variables of the DAQ cycle. Next in line is the connection with the modules by pressing the `Initialize` button. When pressed the connection with the controller module (master) of the VMEbus crate is established, the modules are initialized and the output files are opened. In each and every of the afforementioned steps validation checks are performed. If the initialization is successful the checkbox next to the push button should change state into `Qt::Checked` and the two buttons below should be enabled. The first one if pressed would cancel the current initialization and load the configuration without making any changes whatsoever. The second would start the DAQ cycle. When the cycle is running the VMEbus waits for an interrupt generated by the C-RAMS signaling that there are data converted and ready to be read out. When the interrupt is assessed the readout procedure begins, writing an event data block for each FIFO of each C-RAMS module in the output raw data file. The run should end either when the maximum number of events is reached or when the `Stop Cycle` button is pressed. Whichever the case might be, when the run ends the output file is closed, the modules are reset back to their default settings and the configuration tab updates itself to the new configuration (as the run number changes) being ready for a new run.

CHAPTER 4

Results

In order to test the functionality and validate the smoothness of the data flow of the software, a real conditions stress test should have taken place. In this consensus, the software was tested in the test beam of RD-51 set up to read out a sole station (two MicroMeGaS detectors) of the whole MicroMeGaS telescope.

4.1 The Set-Up

The double MicroMeGaS station was accommondated on the solid frame of GEM Tracker and used a shared `TRIGGER` signal with it. The station was placed down-beam, near the end of the solid table. After further analysis, it was proved that the station was a bit off of the center of the beam, so the wanted beam profile would be shifted in a direction off center.

In detail, the setup was using a VMEbus crate as was described in a previous chapter, accomondating a CAEN V2718 Controller Unit as the Master of the VMEbus crate, a CAEN V551B Sequencer and a CAEN C-RAMS V550 module. The Master module of the crate was connected to a pc via an optical link. As far as the trigger signal is concerned, for the experiment at hand, it was taken externally as the whole telescope was using a shared trigger with two scintillators at the beginning of the set-up and two scintillators at the end, while using coincidence logic. A NIM crate with a Dual Timer used as test trigger signal, a Fan-In-Fan-Out unit and the custom made LAPP module translating the control signals of the sequencer through its flat cable to LEMO cables that are the inputs of the Gassiplex front-end chip, was also used. Finally, a main frame was used housing power supply units for powering the MicroMeGaS station and the GEM tracker was used.

During the test beam two sets of Gassiplex front-end chips were tested. One coming from LAPP, being the latest series of this chip and one coming from CEA Saclay that belonged in the previous series. A table containing the runs taken at the test beam area is presented in table 3.1.

The goal of the test beam was to reproduce using the new software the beam profile. Using the double MicroMeGaS, one can reproduce the profile of the oncoming beam on both axis, X and Y, since the strips on the two chambers are rotated 90^0 with respect to eachother. Since the goal of the software is not to produce a full analysis ROOT file but merely the raw data of the run taken, the only plots that are going to be presented are hitmaps of the two chambers constructed by a script written in the software, and actually is a feature of it (see <<Save sample hitmap to ROOT file?>> checkbox). For that matter, two runs where chosen to be presented: one taken with a pion beam and one with a muon beam. The difference between those two is the spread of the beam in the vertical axis. While a pion beam is well focused on both axes,

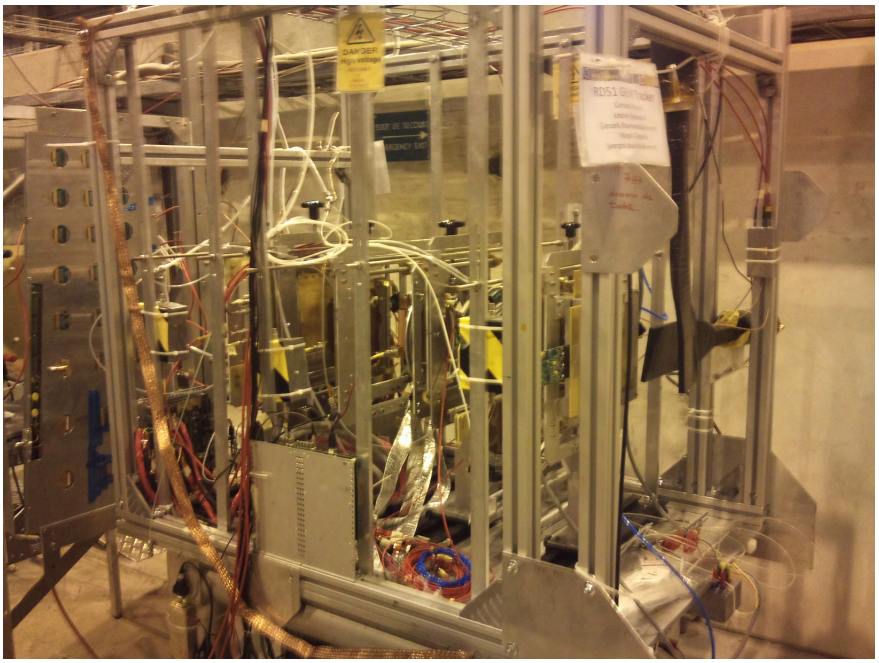


Figure 4.1: A long shoot of the whole set up consisting of the movable table, the solid frame, the GEM tracker and at the end (far right) the double MicroMeGaS station.

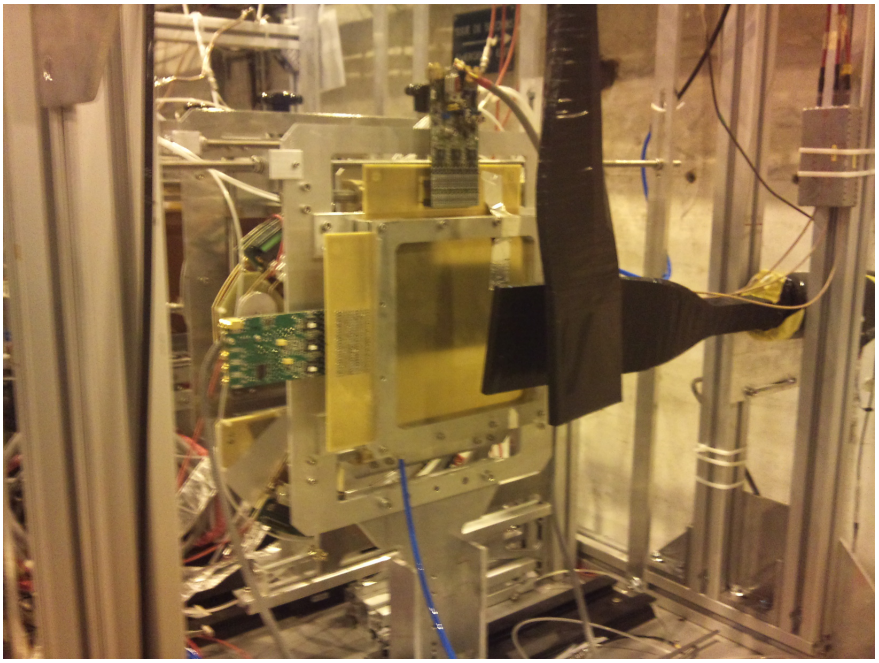


Figure 4.2: A closer shoot of the part of the set up where the double MicroMeGaS was placed. In picture the scintillators formation for the trigger is also apparent.

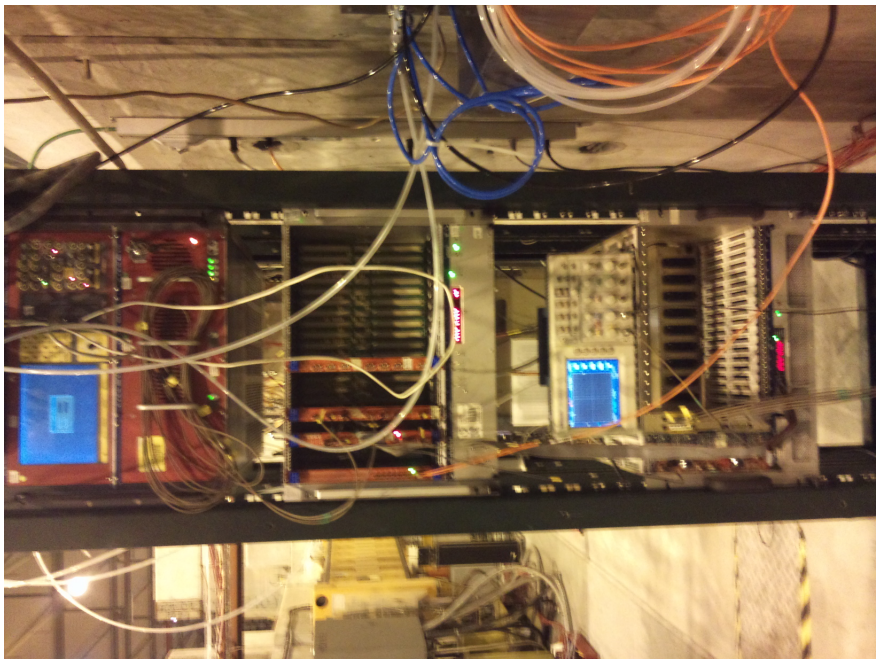


Figure 4.3: The complete rack of the Test Beam setup. The trigger signal is external so no units are used for it.

Run Number	**Type**	**Events**	**Beam**	**Chip**
5000	Physics	20000	Muons	LAPP
5001	Physics	50000	Pions	LAPP
5002	Physics	100000	Muons	LAPP
6001	Test	1000	–	Saclay
6002	Test	1000	–	Saclay
6003	Test	1000	–	Saclay
6004	Test	1000	–	Saclay
6005	Test	1000	–	LAPP
8000	Physics	10000	Muons	Saclay
8001	Test	1000	Muons	Saclay
8002	Physics	10000	Muons	Saclay
8200	Physics	50000	Muons	Saclay
8201	Physics	20000	Muons	Saclay
9003	Physics	50000	Muons	Saclay
9004	Physics	20000	Muons	Saclay
9005	Physics	50000	Muons	Saclay
9006	Physics	20000	Pions	Saclay
9010	Physics	50000	Pions	Saclay
9011	Physics	20000	Pions	Saclay
9012	Physics	100000	Muons	Saclay

Table 4.1: Runs taken during test beam.

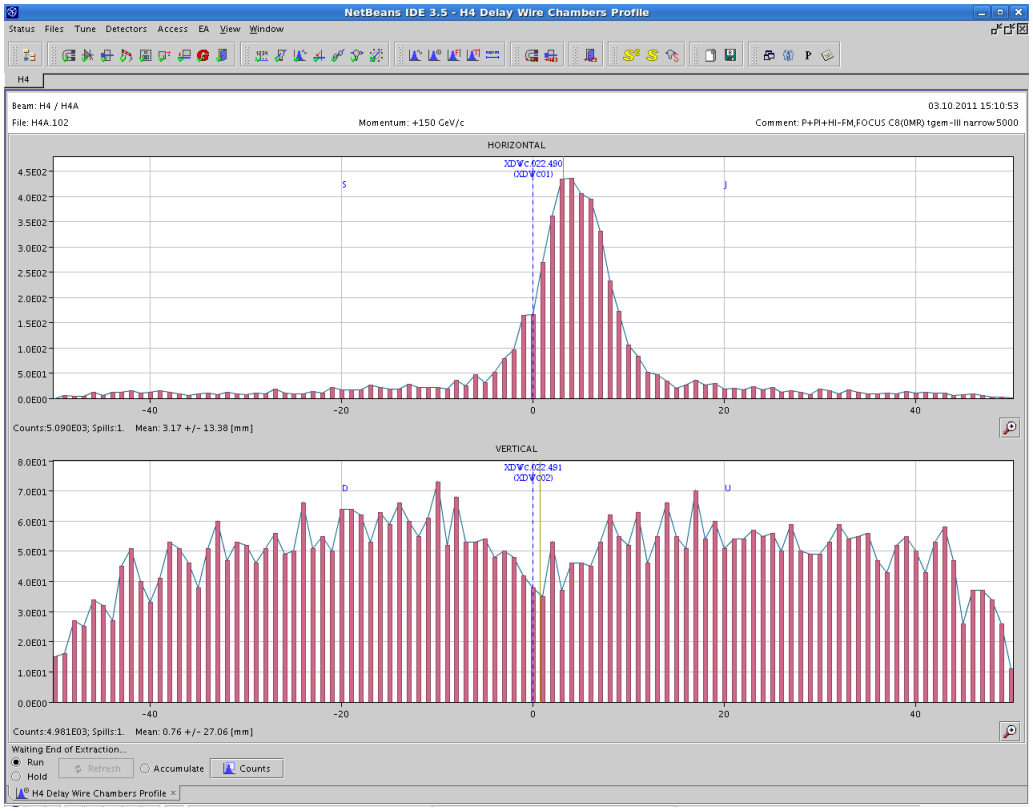


Figure 4.4: Beam profile for muons run as provided by SPS.

a muon beam is focused on the horizontal axis, but widely spread on the vertical one. A data taking event rate maxing at around 7000 events per spill was recorded.

4.2 Muon Beam: run9005P.dat

As the filename reveals by itself the run with run number 9005, a physics run, is the first to be presented. This is a muon beam run and in figure 3.4 is the beam profile as provided by SPS. The voltages on the chamber were $560V$ and $820V$ for the drift and mesh accordingly. The gas mixture used was $Ar : CO_2 - (85 : 15)$.

A script was created that takes as input the raw data file of the run and creates a histogram, on which the horizontal axis is the strip number, and fills it according to how many times each strip fired. In addition, in order to clear up some background a threshold was manually implemented. The result is presented in figure 3.5. One can easily see that there are some noisy strips (e.g. strip 16, 28, 32 etc) which give signal almost in each event as well as some dead strips that generate no signal at all (or the signal is below th e threshold, e.g. strip 8, 42, 95 etc.). In figure 3.6 is presented a closer look on the area of the beam profile, though only for X axis, as the profile on the vertical axis is widely spread and there is no need for closer look

Taken into consideration that the alignment of the chamber was a bit off center, the two beam profiles match, however the precision would be even more apparent in the pion beam results. In addition, if one fills the plot with a unit block for each strip that fires without a threshold, one may acquire a way to see graphically the efficiency of the chamber at these voltages. In other words, if there is no threshold and

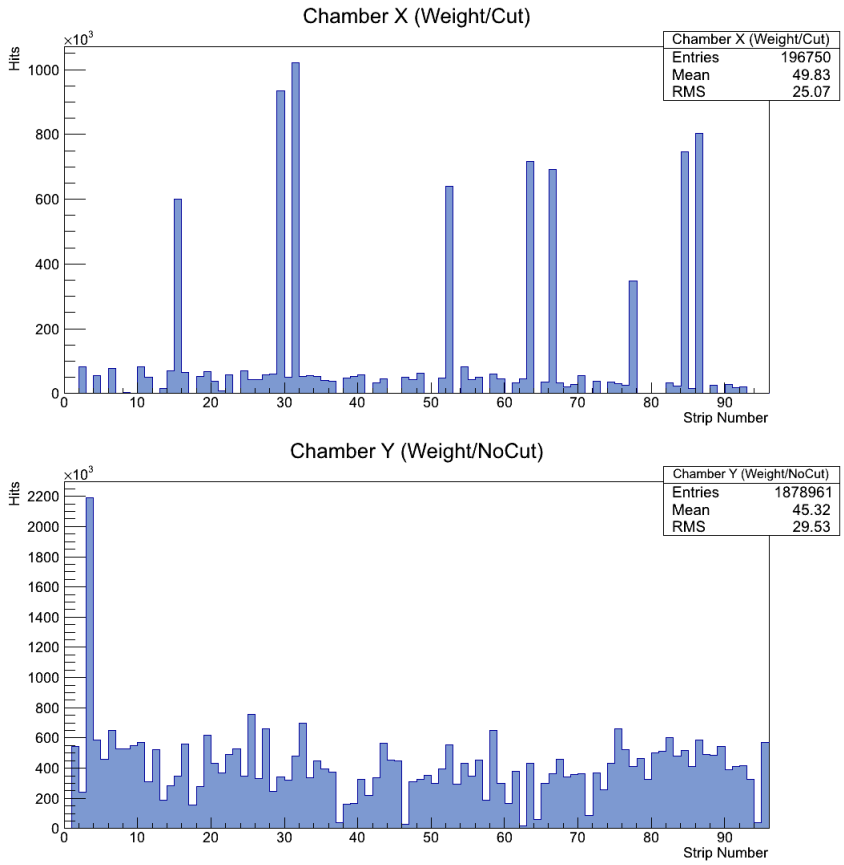


Figure 4.5: Beam profile for muons run as taken by the software.

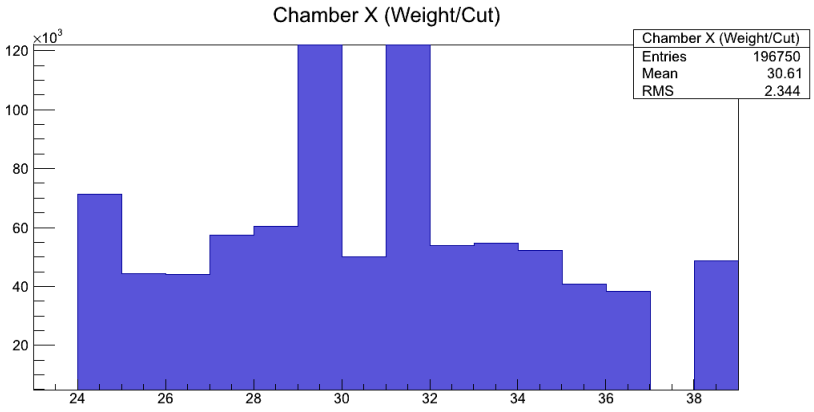


Figure 4.6: Closer look on beam profile on X axis for muons run as taken by the software.

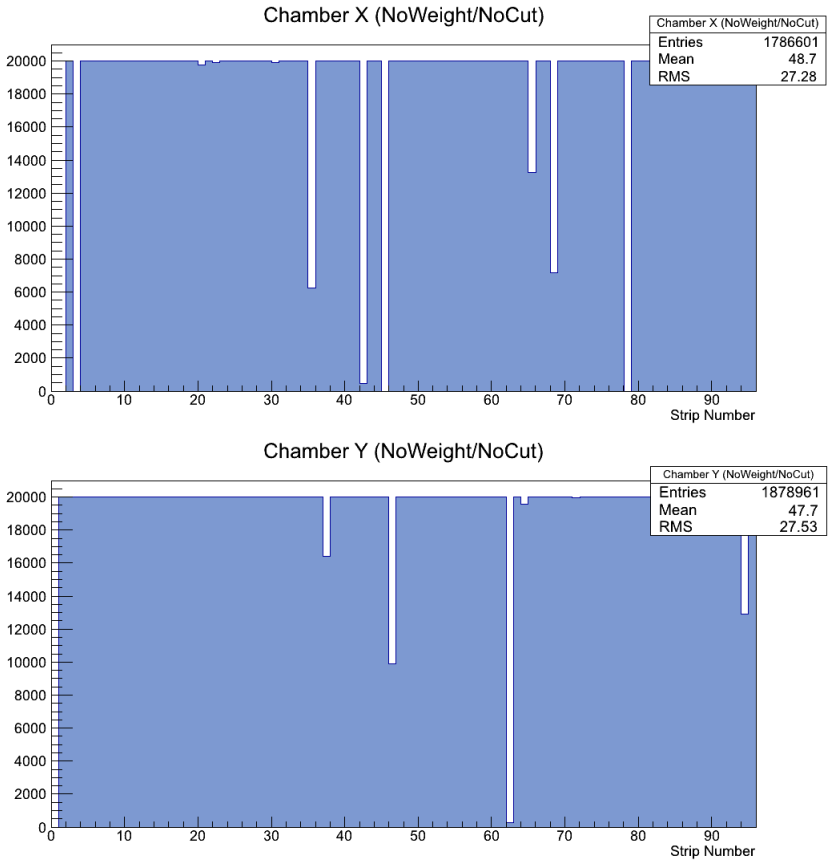


Figure 4.7: Efficiency plot for the chambers.

taken into consideration that all channels (strips) have a pedestal value, using a 100% efficient chip for a 10000 events run one should expect a plot with a flat distribution at the point of 10000 hits. Creating this plot for the current run one acquires the plots in the figure 3.7.

4.3 Pion Beam: run9012P.dat

Run 9012 is a physics run taken with a pion beam. The pion beam was well focused on both axes having a size of $4mm$ on the horizontal axis and $6mm$ on the vertical axis. This information was provided by SPS and can be seen in figure 3.8. The voltages and the gas mixture were the same as the muon beam test.

The result taken by software can be seen in figure 3.9. By taking the strip number difference of the start and the end of the bumps and multiplying by the strip pitch ($250\mu m$), one can find exactly the size of the beam spread. Furthermore, in figure 3.10 it is presented a closer look on the strips consisting the beam profile on both axes.

In addition the efficiency plots for the pion beam run are presented in figure 3.11. By efficiency here one means the successful strip fire rate. Since in this plot one would expect for a 100% efficient front-end chip to raise for each strip to the height of the total events, meaning that all the readout channels function well. However, one can easily see that some strips are inefficent.

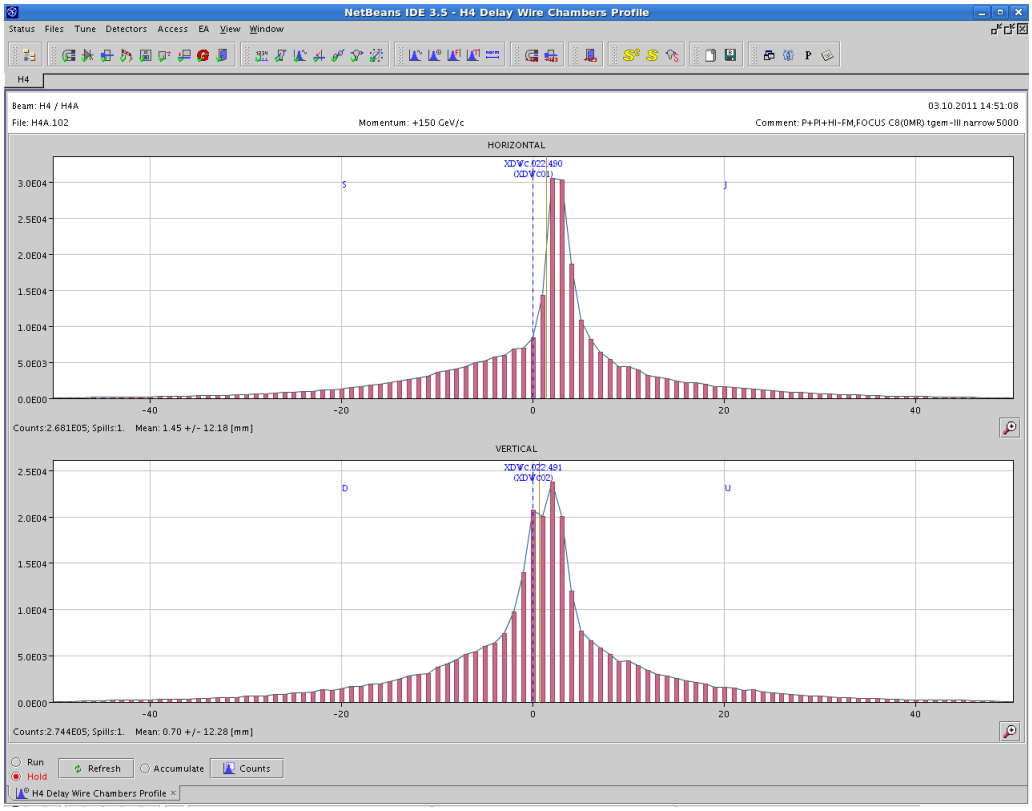


Figure 4.8: Beam profile for pions run as provided by SPS.

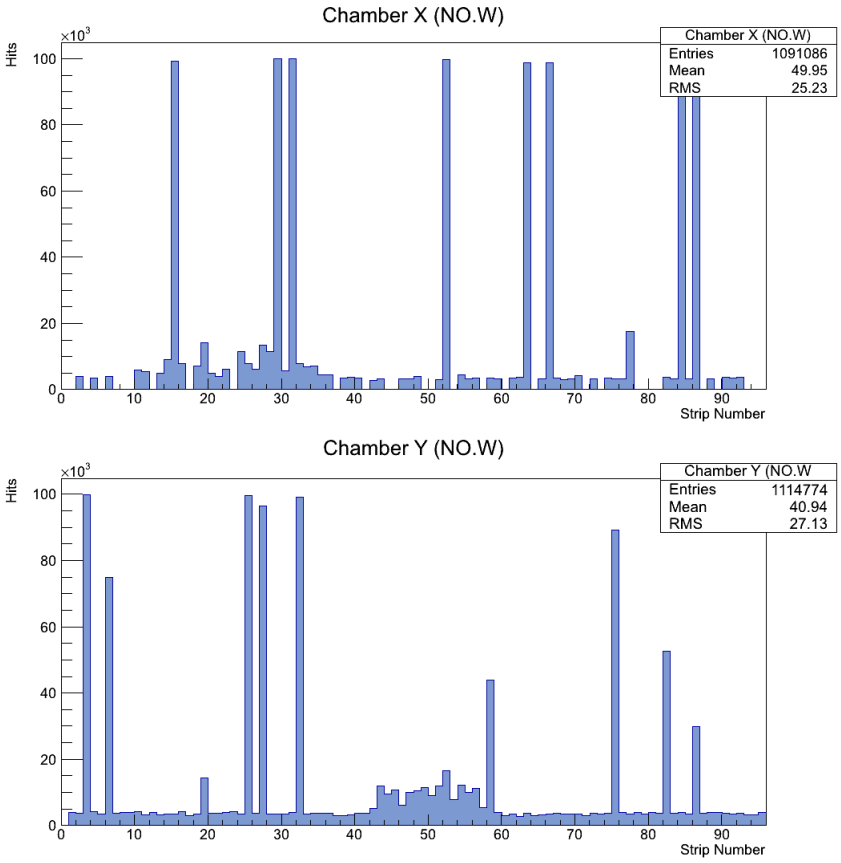


Figure 4.9: Beam profile for pions run as taken by the software.

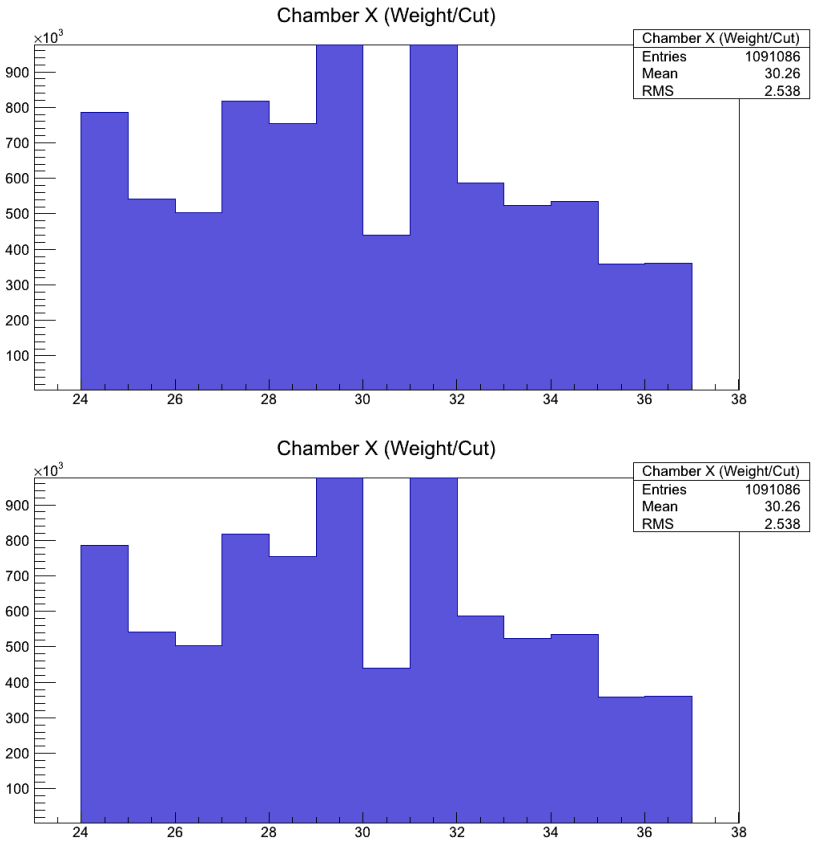


Figure 4.10: Closer look on the beam profile for pions run as taken by the software.

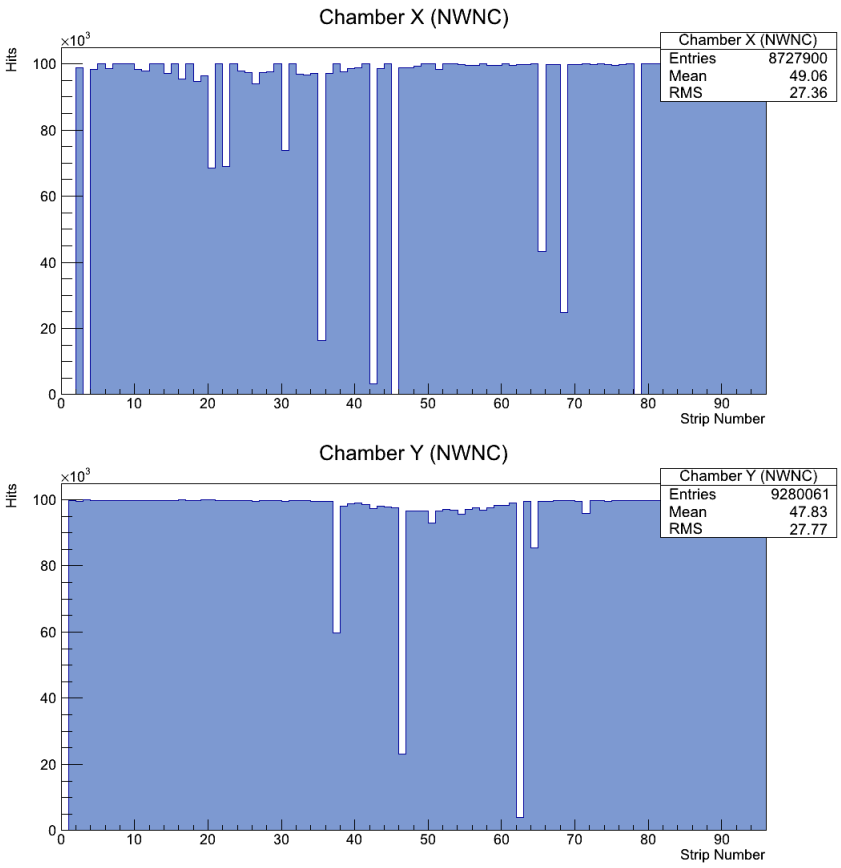


Figure 4.11: Efficiency plots for the pion beam test.

4.4 Conclusion

As a conclusion to this thesis, one should recapitulate the main features and advancements of the software that was developed in relation to its predecessor.

First of all, the core of the software handling the DAQ cycle is written in pure C++ exploiting its virtues concerning the memory handling and thread capabilities instead of a control language such as LabView, PVSS, etc. This provides the total system to handle data flow at a speed closer to the one that VME is capable of, alleviate the pain of the constant polling of control languages. During the test beam the total system reached a data rate of $700Hz$ which is already 10 times higher than its predecessor. However, through meticulous editing of the code the total rate can easily exceed the barrier of $1kHz$.

In addition, the new software is easy to be used by the end-user. The graphical interface includes all the information needed by a shifter and are essential for the expert to proceed to a successful run. It can be easily customized to the needs of each researcher via the interface, without having to swim to an endless sea of code or edit complex files endangering the final output. Finally, as an extra touch of facilitation, the user is able to open a short, categorized documentation by just pressing a button on the interface of the software, even when the DAQ cycle is active.

And last but not least, it just works. As prooved by the tests in the area, the new software was able to reconstruct the beam profile as provided by SPS. Also, the log viewer provides the expert user with some hints in case of failure or data corruption in order to minimize any possible data loss. Moreover, the output file is reduced in its size by almost an order of magnitude in relation to the previous software making easier the process of saving multiple run files of raw data even in smaller hard disks.

Bibliography

[1] *American National Standard for VME64.*

[2] *Technical Information Manual Mod. V2718/VX2718/VN2738 VME-PCI Optical Link Bridge.*

[3] *Technical Information Manual Mod. V462 Dual Gate Generator.*

[4] *Technical Information Manual Mod. V550/V550B 2 Channel C-RAMS.*

[5] *Technical Information Manual Mod. V551 C-RAMS Sequencer.*

[6] Ιστοσελίδες όπως:

- http://public.web.cern.ch/public/
- https://twiki.cern.ch/twiki/bin/view/atlas/webhome
- https://twiki.cern.ch/twiki/bin/view/atlas//muonmicromegas
- http://rd51-public.web.cern.ch/rd51-public/
- http://mpgd.web.cern.ch/mpgd/
- http://www.caen.it

.

[7] Samuel A Andriamonje, S Aune, H Bräuninger, and T Papaevangelou. A new micromegas line for the cast experiment. *Nucl. Instrum. Methods Phys. Res., A*, 581(1-2):217--220, 2007.

[8] Elena Aprile, Aleksey E Bolotnikov, Alexander I Bolozdynya, and Tadayoshi Doke. *Noble Gas Detectors.* Wiley, Weinheim, 2006.

[9] G. Charpak, J. Derré, Y. Giomataris, and Ph. Rebourgeard. Micromegas, a multipurpose gaseous detector. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 478(1-2):26 -- 36, 2002.

[10] R C Fernow. *Introduction to experimental particle physics.* Cambridge Univ. Press, Cambridge, 1986.

[11] P Fonte, P Martinengo, E Nappi, R Oliveira, V Peskov, F Pietropaolo, and P Picchi. Advances in the development of micropattern gaseous detectors with resistive electrodes. Technical Report arXiv:1005.1477, May 2010.

[12] I. Giomataris, R. De Oliveira, S. Andriamonje, S. Aune, G. Charpak, P. Colas, G. Fanourakis, E. Ferrer, A. Giganon, Ph. Rebourgeard, and P. Salin. Micromegas in a bulk. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 560(2):405 -- 408, 2006.

[13] Y. Giomataris, Ph. Rebourgeard, J. P. Robert, and G. Charpak. Micromegas: a high-granularity position-sensitive gaseous detector for high particle-flux environments. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 376(1):29 -- 35, 1996.

[14] Claus Grupen, Armin Böhrer, and Ludek Smolik. *Particle detectors*. Cambridge monographs on particle physics, nuclear physics, and cosmology. Cambridge Univ. Press, Cambridge, 1996.

[15] Mark Summerfield Jasmin Blanchette. *C++ GUI Programming with Qt4; 2nd edition*. Prentice Hall, Upper Saddle River, USA, 2008.

[16] Scott J. Kleper Nicholas A. Solter. *Professional C++*. Wiley Publishing, Inc, USA, 2005.

[17] R Oliveira, V Peskov, F Pietropaolo, and P Picchi. First tests of gaseous detectors made of a resistive mesh. Technical Report arXiv:1002.1415, Feb 2010.

[18] Herbert Schildt. *C++ The Complete Reference; 2nd ed.* McGraw-Hill, USA, 2003.

[19] Mark Summerfield. *Advanced Qt Programming*. Addison-Wesley, Upper Saddle River, USA, 2010.

[20] Nicholas Tsoulfanidis. *Measurement and detection of radiation; 2nd ed.* Taylor and Francis, USA, 1995.