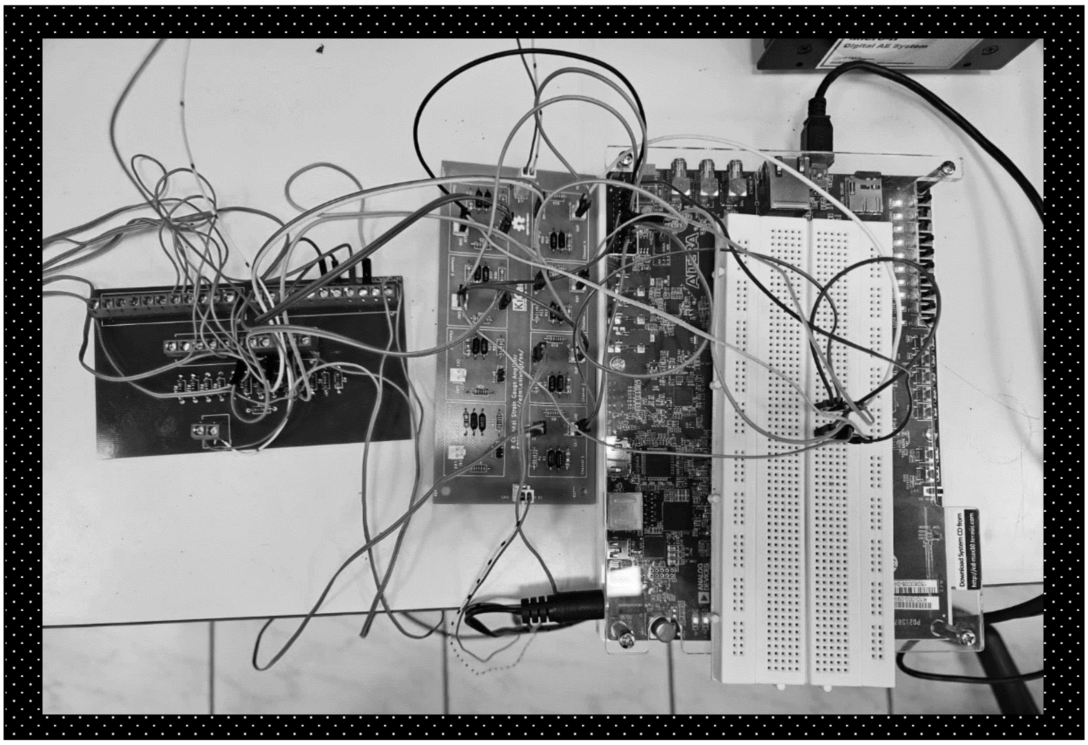
UNIVERSITY OF WEST ATTICA
FACULTY OF ENGINEERING
DEPARTMENT OF ELECTRICAL & ELECTRONICS ENGINEERING

Diploma Thesis

Embedded System for the Recording and Visualization of Voltage Sensor Data



Student: **Evangelia-Agni Alexopoulou**
Registration Number: **18387107**

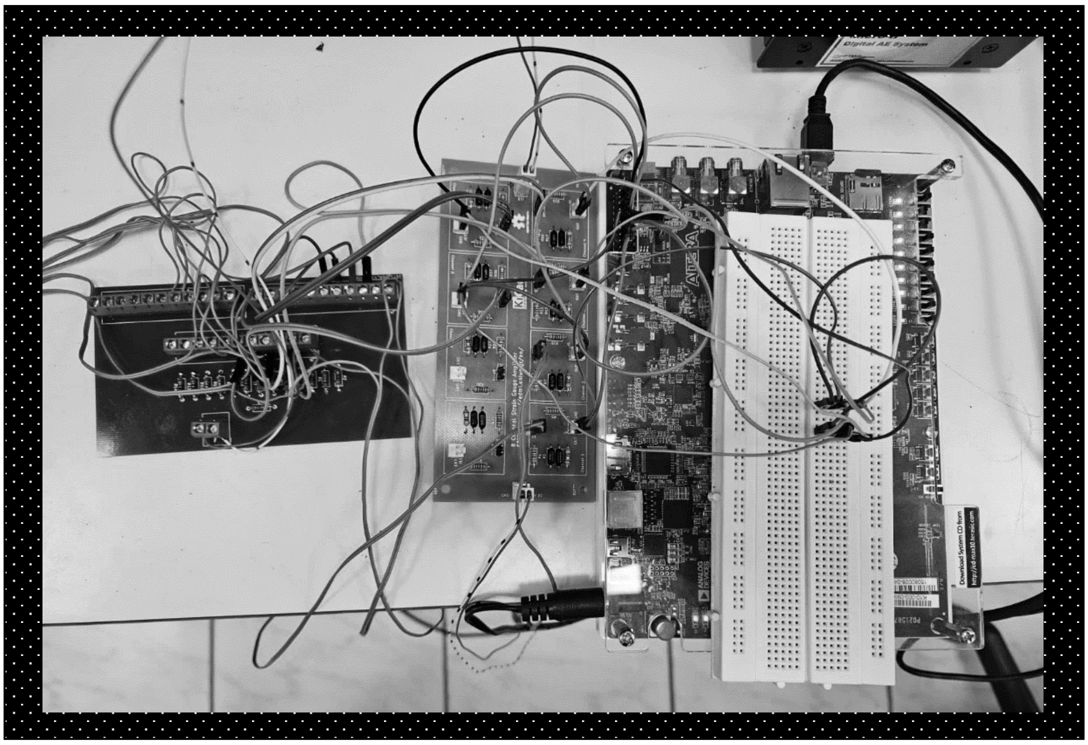
Supervisor: **Prof. Ilias Stavrakas**

ATHENS-EGALEO, JULY 2024

ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ
ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ & ΗΛΕΚΤΡΟΝΙΚΩΝ ΜΗΧΑΝΙΚΩΝ

Διπλωματική Εργασία

Ενσωματωμένο Σύστημα Καταγραφής και Απεικόνισης Δεδομένων Αισθητήρων Τάσης



Φοιτήτρια: **Ευαγγελία-Αγνή Αλεξοπούλου**
Αριθμός Μητρώου: **18387107**

Επιβλέπων Καθηγητής: **Ηλίας Σταύρακας, Καθηγητής**

ΑΘΗΝΑ-ΑΙΓΑΛΕΩ, ΙΟΥΛΙΟΣ 2024

Η Διπλωματική Εργασία έγινε αποδεκτή και βαθμολογήθηκε από την εξής τριμελή επιτροπή:

Ηλίας Σταύρακας, καθηγητής	Ευστάθιος Κυριάκης Μπιτζάρος, καθηγητής	Αικατερίνη Ζαχαριάδου, καθηγήτρια
[illegible] Digitally signed by ILIAS STAVRAKAS Date: 2024.07.26 11:41:05 +03'00' (Υπογραφή)	[illegible] Efstathios Kyriakis-Bitzaros 2024.07.26 11:58:08 +03'00' (Υπογραφή)	Aikaterini-Styliani Zachariadou Digitally signed by Aikaterini-Styliani Zachariadou Date: 2024.07.29 12:46:44 +03'00' (Υπογραφή)

ΔΗΛΩΣΗ ΣΥΓΓΡΑΦΕΑ ΔΙΠΛΩΜΑΤΙΚΗΣ ΕΡΓΑΣΙΑΣ

Η κάτωθι υπογεγραμμένη Ευαγγελία-Αγνή Αλεξοπούλου του Παναγιώτη με αριθμό μητρώου 18387107 φοιτήτρια του Πανεπιστημίου Δυτικής Αττικής της Σχολής ΜΗΧΑΝΙΚΩΝ του Τμήματος ΗΛΕΚΤΡΟΛΟΓΩΝ ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΜΗΧΑΝΙΚΩΝ,

δηλώνω υπεύθυνα ότι:

«Είμαι συγγραφέας αυτής της διπλωματικής εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, οι όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε ακριβώς είτε παραφρασμένες, αναφέρονται στο σύνολό τους, με πλήρη αναφορά στους συγγραφείς, τον εκδοτικό οίκο ή το περιοδικό, συμπεριλαμβανομένων και των πηγών που ενδεχομένως χρησιμοποιήθηκαν από το διαδίκτυο. Επίσης, βεβαιώνω ότι αυτή η εργασία έχει συγγραφεί από μένα αποκλειστικά και αποτελεί προϊόν πνευματικής ιδιοκτησίας τόσο δικής μου, όσο και του Ιδρύματος.
Παράβαση της ανωτέρω ακαδημαϊκής μου ευθύνης αποτελεί ουσιώδη λόγο για την ανάκληση του διπλώματός μου.»

Η Δηλούσα

ΕΑλεξοπούλου

Acknowledgements

Αφιερώνω αυτή τη διπλωματική εργασία στους γονείς μου, των οποίων η αμέριστη αγάπη, υποστήριξη και ενθάρρυνση ήταν το θεμέλιο για την εκπλήρωση αυτού του σημαντικού στόχου στη ζωή μου. Χωρίς τη συνεχή τους στήριξη και πίστη σε εμένα, τίποτα από αυτά δεν θα ήταν δυνατό.

Θα ήθελα επίσης να εκφράσω τις βαθύτατες ευχαριστίες μου στους καθηγητές μου Ευστάθιο Κυριάκη-Μπιτζάρο και Ηλία Σταύρακα, για την πολύτιμη καθοδήγησή τους κατά τη διάρκεια της διπλωματικής μου εργασίας και όχι μόνο. Η εμπειρία τους, οι γνώσεις τους και η αφοσίωσή τους υπήρξαν πηγή έμπνευσης σε κάθε βήμα αυτής της πορείας. Η συνεισφορά τους ήταν ανεκτίμητη και θα παραμείνει πάντα χαραγμένη στη μνήμη μου.

Επίσης, θα ήθελα να εκφράσω την ειλικρινή μου ευγνωμοσύνη στον κ. Ηλία Δαραδήμο Ηλεκτρονικό Μηχανικό Διαστημικών Συστημάτων για τη σημαντική του συμβολή στην κατασκευή των πλακετών της διπλωματικής μου. Η υποστήριξη και η τεχνογνωσία του υπήρξαν καταλυτικές για την ολοκλήρωση αυτού του έργου.

Σας ευχαριστώ όλους από καρδιάς για τη στήριξη και τη βοήθειά σας.

Περίληψη

Η παρούσα διπλωματική εργασία επικεντρώνεται στη σχεδίαση, ανάπτυξη και υλοποίηση ενός προηγμένου συστήματος παρακολούθησης παραμορφώσεων, το οποίο χρησιμοποιεί την τεχνολογία FPGA (Field Programmable Gate Array). Το κύριο αντικείμενο του συστήματος είναι η ακριβής συλλογή, επεξεργασία και απεικόνιση δεδομένων σε πραγματικό χρόνο, κάτι που είναι ζωτικής σημασίας για τη διατήρηση της ακεραιότητας και της αποτελεσματικότητας των ηλεκτρικών συστημάτων, καθώς και για την ασφάλεια και την ανθεκτικότητα των υποδομών.

Το σύστημα αυτό βασίζεται στη χρήση αισθητήρων παραμόρφωσης KYOWA KFG-5-120-C1-11L3M2R, μιας ειδικά κατασκευασμένης μονάδας προσαρμογής σήματος και του NEEK MAX10 FPGA Development Kit για την καταγραφή και ανάλυση των δεδομένων. Το FPGA επιτρέπει την ταχεία συλλογή δεδομένων και την επεξεργασία τους σε πραγματικό χρόνο, προσφέροντας ακριβή και αξιόπιστη παρακολούθηση παραμορφώσεων. Η επιλογή του NEEK MAX10 FPGA Development Kit έγινε λόγω της υψηλής απόδοσης, της ευελιξίας και της εκτεταμένης υποστήριξης για πολύπλοκες εργασίες επεξεργασίας σήματος.

Η εργασία αναπτύσσει δύο λειτουργικές εκδόσεις του συστήματος. Η πρώτη έκδοση, της σειριακής επικοινωνίας, επιτρέπει την αποστολή δεδομένων από το FPGA στον υπολογιστή, όπου το λογισμικό MATLAB αναλαμβάνει την επεξεργασία και απεικόνιση των δεδομένων. Αυτός ο τρόπος λειτουργίας παρέχει ευελιξία και δυνατότητα λεπτομερούς ανάλυσης δεδομένων σε ένα ισχυρό υπολογιστικό περιβάλλον. Η δεύτερη έκδοση, εκείνη της αυτονομίας, λειτουργεί ανεξάρτητα, αξιοποιώντας τον επεξεργαστή Nios II που είναι ενσωματωμένος στο FPGA και το αναπτυξιακό περιβάλλον Eclipse για την επεξεργασία δεδομένων και την απεικόνισή τους σε πραγματικό χρόνο στην οθόνη LCD της NEEK MAX10. Η έκδοση αυτή είναι προτιμότερη για περιπτώσεις που απαιτείται συνεχής παρακολούθηση σε απομακρυσμένες ή δυσπρόσιτες τοποθεσίες.

Η διπλωματική περιλαμβάνει λεπτομερή ανάλυση της αρχιτεκτονικής του συστήματος, τονίζοντας τα βασικά του στοιχεία: το FPGA, τη μονάδα ενίσχυσης σήματος, τη γέφυρα Wheatstone και τους αισθητήρες παραμόρφωσης. Το FPGA αποτελεί την καρδιά του συστήματος, προσφέροντας τις απαιτούμενες δυνατότητες για την ταχεία συλλογή και επεξεργασία δεδομένων. Η μονάδα προσαρμογής σήματος είναι υπεύθυνη για την ενίσχυση και το φιλτράρισμα των σημάτων από τους αισθητήρες παραμόρφωσης, εξασφαλίζοντας ότι τα σήματα είναι κατάλληλα για επεξεργασία από το FPGA. Η γέφυρα, βασισμένη στην αρχή της γέφυρας Wheatstone, χρησιμοποιείται για την ακριβή μέτρηση των μικρών αλλαγών στην αντίσταση που προκαλούνται από τις παραμορφώσεις.

Η εργασία παρουσιάζει επίσης την ανάπτυξη λογισμικού για τη διαχείριση της επικοινωνίας και της επεξεργασίας των δεδομένων. Αναλυτικότερα, για την έκδοση σειριακής επικοινωνίας, αναπτύχθηκε λογισμικό σε MATLAB που επιτρέπει την αποστολή δεδομένων από το FPGA στον υπολογιστή και το αντίστροφο, την επεξεργασία των δεδομένων που λαμβάνει, την απεικόνισή τους σε γραφήματα, καθώς και για την αποθήκευση των δεδομένων για μελλοντική ανάλυση. Για την αυτόνομη έκδοση, αναπτύχθηκε κώδικας σε γλώσσα C

χρησιμοποιώντας το περιβάλλον ανάπτυξης Eclipse, το οποίο επιτρέπει την επεξεργασία των δεδομένων στο FPGA και την απεικόνισή τους στην οθόνη LCD.

Η απόδοση του συστήματος επικυρώθηκε μέσω μιας σειράς πειραμάτων, τα οποία περιλάμβαναν την παρακολούθηση παραμορφώσεων υπό διάφορες συνθήκες φόρτισης, όπως συμπιεστική φόρτιση, προφορτωμένα δείγματα και δοκιμές κάμψης τριών σημείων. Τα αποτελέσματα αυτών των πειραμάτων έδειξαν ότι το σύστημα είναι ανθεκτικό και ακριβές στην καταγραφή δεδομένων παραμόρφωσης, παρέχοντας πολύτιμες πληροφορίες για τη συμπεριφορά των υλικών.

Η εργασία προτείνει αρκετές βελτιώσεις για μελλοντική ανάπτυξη του συστήματος. Αυτές περιλαμβάνουν την ασύρματη επικοινωνία, την αυτοματοποιημένη ανάλυση δεδομένων, την ενεργειακή αποδοτικότητα, τη βελτίωση της διεπαφής χρήστη, την εισαγωγή μηχανισμών άμεσης ανατροφοδότησης, την επέκταση της αποθήκευσης δεδομένων και την ενσωμάτωση με το cloud για την απομακρυσμένη πρόσβαση και ανάλυση των δεδομένων.

Συμπερασματικά, αυτή η διπλωματική εργασία συμβάλλει στον τομέα της παρακολούθησης της δομικής υγείας, προσφέροντας μια αξιόπιστη και ευέλικτη λύση για τη μέτρηση παραμορφώσεων σε πραγματικό χρόνο. Αναδεικνύει τη δυναμική της τεχνολογίας FPGA για ευρύτερες εφαρμογές στη μηχανική και τη βιομηχανία, επισημαίνοντας τις δυνατότητες για περαιτέρω βελτιώσεις και καινοτομίες στον τομέα της παρακολούθησης δομικών παραμορφώσεων. Η ενσωμάτωση προηγμένων τεχνολογιών σε συστήματα παρακολούθησης παραμορφώσεων προσφέρει σημαντικές δυνατότητες για την έγκαιρη ανίχνευση προβλημάτων και τη λήψη προληπτικών μέτρων, βελτιώνοντας την ασφάλεια και την αξιοπιστία των κρίσιμων υποδομών.

Abstract

This thesis presents the design, development and implementation of an advanced strain monitoring system utilizing FPGA technology, aimed at providing accurate, real-time data acquisition, processing, and visualization. The system employs KYOWA KFG-5-120-C1-11L3M2R strain gauges, a custom-built signal conditioning unit, and the NEEK MAX10 FPGA Development Kit to capture and analyze strain data. Two operational versions are explored: the Serial Communication Version, which interfaces with MATLAB for detailed data analysis on a computer, and the Autonomous Version, which leverages the Nios II processor for on-board data processing and real-time display on the NEEK MAX10's LCD screen.

Pilot experiments were conducted to validate the system's performance under various loading conditions, including static loadings, namely compressive loading, preloaded specimen, and 3-point bending tests. The results demonstrated the system's robustness and accuracy in capturing strain data, providing valuable insights into material behavior. Future improvements are proposed, including enhanced sensor integration, wireless communication, automated data analysis, miniaturization, power efficiency, improved user interface, real-time feedback mechanisms, extended data storage, cloud integration, and saving data on the SD card for the autonomous version.

This research contributes to the field of structural health monitoring by offering a reliable and versatile solution for real-time strain measurement, highlighting its potential for broader applications in engineering and industrial settings.

"Fracture will be one of the last phenomena comprehended by the scientists"
(M. Marder and J. Fineberg ,1996)

Keywords

1. Strain Monitoring
2. FPGA Technology
3. Real-Time Data Acquisition
4. Signal Conditioning
5. NEEK MAX10 Development Kit
6. MATLAB Integration
7. Nios II Processor
8. Structural Health Monitoring
9. 3-Point Bending
10. Data Visualization

Table of Contents

List of Figures

1 Chapter 1st : Introduction

1.1 Project Overview

In the contemporary engineering landscape, precise monitoring and recording of voltage fluctuations are crucial for maintaining the integrity and efficiency of electrical systems and ensuring the safety and durability of infrastructures. This thesis, titled "Embedded System for the Recording and Visualization of Voltage Sensor Data," proposes a solution to enhance these capabilities through the development of an advanced embedded system utilizing Field Programmable Gate Array (FPGA) technology. This system aims to provide a sophisticated yet cost-effective approach for capturing, managing, and visualizing voltage data, integrating high-speed processing capabilities and the flexibility of FPGAs. The use of FPGA technology enables real-time analytics and high-resolution visualization, offering the integration of soft processor core. This approach significantly improves operational efficiencies and enhances the protection and sustainability of electrical infrastructures, thus contributing to advancements in structural health monitoring and electrical engineering.

The rapid advancement of technology and the expansion of infrastructures across urban, industrial, and natural environments have increased the demands on structural integrity monitoring and ensure safety. Continual exposure to dynamic loads, environmental stresses, and operational demands can precipitate structural degradation over time, making early detection of potential issues through strain surveillance a critical priority. This form of observation is vital in various sectors including civil engineering, aerospace, automotive, and large-scale industrial machinery, where early detection of structural deformations prevents failures that could lead to catastrophic consequences and economic losses [1]. Strain monitoring systems provide essential data for maintaining and extending the lifespan of structures and machinery. Indicatively, in civil engineering, real-time observation of bridges and historical buildings can prevent unexpected breakdowns and secure safety. In aerospace and automotive industries, detecting stress and strain in components may avoid catastrophic failures that could result in loss of life and significant financial costs [2].

The system developed in this thesis comprises four main modules: the FPGA unit, the signal conditioning amplifier unit, the resistor bridge and the strain gauges. The heart of the system is the NEEK MAX10 FPGA Development Kit, chosen for its high performance, flexibility, and extensive support for complex signal processing tasks. The MAX10 FPGA handles high-speed data acquisition and real-time processing, which is essential for accurate and reliable strain supervision. [3]

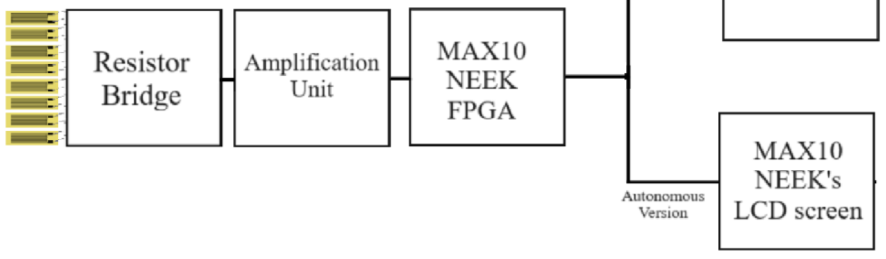


Figure 1: Block diagram of the system

Strain gauges capture mechanical deformation, which is converted into electrical signals by the resistor bridge, and then processed by the conditioning unit, which amplifies and filters them. This combination of components provides a comprehensive and efficient solution for real-time strain monitoring, enabling detailed analysis and visualization of strain data.

1.1.1 Hardware and Software Integration

The core data processing for this thesis leverages a combination of high-performance hardware and advanced software tools to ensure precise and efficient strain surveillance. The primary hardware platform is the NEEK MAX10 FPGA Development Kit, which interfaces with the strain gauges and a custom-built signal conditioning unit. The strain gauges, KYOWA KFG-5-120-C1-11L3M2R, measure the strain, while a 12-channel high precision and thermal stability custom resistor bridge converts these measurements into readable electrical signals. These signals are then amplified and conditioned by the signal conditioning unit, utilizing the Texas Instruments TLV9002D dual operational amplifiers.

For software integration, the NEEK board is programmed using Quartus Prime Lite Standard Edition, which provides the necessary tools for FPGA design and implementation. The system has the option to switch between stand-alone operation, visualizing data on the LCD display of the NEEK board and connected mode, where data is transmitted to a PC via serial communication and visualized using MATLAB. MATLAB's advanced graphical capabilities allow for detailed plots and charts, facilitating the interpretation of strain measurements and identifying significant patterns or anomalies. This method ensures that the strain monitoring system not only collects and processes data efficiently but also presents it in an easily interpretable format. For the autonomous version, Eclipse, part of the NIOS II Software Build Tools (SBT), is used to develop software for the NIOS II processor, enabling real-time data visualization on the onboard LCD screen.

Effective data management and visualization are crucial for comprehensive analysis and decision-making. The FPGA systematically organizes and manages raw data, which is then transmitted and stored in a text file on the user’s PC in the serial communication version. This approach to data management and visualization ensures that the strain monitoring system not only collects and processes data efficiently but also presents it in a way that is easily interpretable and actionable. The integration of high-precision hardware and sophisticated software tools provides a comprehensive and reliable system for real-time strain observation.

1.1.2 Significance and Potential Impact

The development of an advanced deformation surveillance system using FPGA technology has broad implications across various engineering fields. This thesis enhances the accuracy, efficiency, and reliability of structural health monitoring systems. Leveraging the high-speed data acquisition and real-time processing capabilities of the NEEK MAX10 FPGA Development Kit ensures early detection of structural deformations and stress concentrations. This capability is critical for preventing failures in civil infrastructure, such as bridges, buildings, and dams, thereby enhancing public safety and reducing maintenance costs.

The system integrates cost-effective components, such as strain gauges and a custom signal conditioning unit, ensuring affordability without sacrificing performance. The use of open-source software further reduces costs compared to traditional monitoring systems, making it viable for widespread adoption across various industries. This leads to substantial economic savings by preventing structural failures and extending the lifespan of critical infrastructure.

Utilizing the NIOS II processor and advanced software tools like Quartus Prime Lite and Eclipse for FPGA programming and data visualization demonstrates the potential of integrating modern computing technologies in engineering applications. This approach sets a new standard for real-time deformation observation and opens avenues for further research and development. The system's design supports both autonomous and non-autonomous operation modes, providing flexibility in data visualization and analysis. The ability to visualize data in real-time on the NEEK board's LCD screen or transmit it to MATLAB for detailed analysis caters to different operational needs, enhancing the system's applicability across various fields.

This thesis aims to serve as a valuable resource for researchers and professionals in civil engineering, aerospace, automotive, and industrial machinery. The insights gained from this work can inform future developments in deformation surveillance technologies and contribute to the broader field of structural health monitoring. Overall, integrating FPGA technology in deformation monitoring systems addresses current challenges, offering improved protection, economic benefits, and technological advancement.

1.2 Importance of Strain Monitoring

Strain monitoring is a critical aspect of maintaining the protection, reliability, and longevity of various structures and mechanical systems. It involves measuring the deformation or strain in materials and structures subjected to dynamic or static loads, and can provide valuable insights into their structural health and performance. The importance of strain monitoring spans several key areas.

- **Impact on Safety and Economic Factors**

The primary purpose of deformation observation is to ensure the protection of structures by detecting signs of stress and deformation that could lead to catastrophic failures if left unaddressed. For example, bridges, buildings, and dams are subjected to continuous stress from loads and environmental factors. By supervising these stresses in real-time, potential issues can be identified early, preventing accidents and saving lives. Early detection of structural problems can also lead to significant economic savings by reducing the need for extensive repairs and avoiding the costs associated with structural failures. Real-time structural health surveillance can assess the status and integrity of structures, preventing costly repairs and ensuring protection. [4].

- **Strain Gauge Applications and Improvements**

Sensing gauges are widely used in various applications to measure deformation. In civil engineering, they observe the structural health of bridges, buildings, and other infrastructure. In the aerospace and automotive industries, deformation gauges help in testing and validating the design of components, ensuring they can withstand operational stresses. Recent advancements in deformation gauge technology and the integration of high-performance information acquisition systems, such as FPGA-based platforms, have significantly improved the accuracy and reliability of deformation measurements. These advancements enable more precise supervision and better-informed decision-making. [5].

- **Perception and Social Impact**

The implementation of advanced strain surveillance systems has a profound social impact by enhancing public confidence in the safety and reliability of critical infrastructure. By demonstrating a commitment to maintaining structural health and safety, authorities can foster trust and reduce public anxiety about potential structural failures. Additionally, the availability of accurate and real-time data on structural performance can inform policy decisions and maintenance strategies, leading to more efficient and effective use of resources. [6].

1.3 Objectives of an FPGA-Based Strain Monitoring System

The development of an FPGA-based strain monitoring system aims to address the challenges associated with traditional surveillance methods by leveraging the advanced capabilities of Field-Programmable Gate Arrays (FPGAs).

- **Systematic and Continuous Monitoring**

The primary objective is to establish a robust system capable of systematic and continuous supervision of structural strains. This includes the ability to operate autonomously for extended periods, ensuring real-time data acquisition and processing without significant downtime. Continuous surveillance is essential for early detection of structural issues and timely intervention, thereby preventing potential failures [5]. Additionally, continuous monitoring helps in understanding the long-term behavior of structures under various loads and environmental conditions, which is crucial for maintenance planning and risk management [4].

- **Accurate Strain Measurement and Analysis**

The system aims to achieve high accuracy in measuring and analyzing deformation information. By utilizing high-precision strain gauges and advanced signal processing techniques within the FPGA, the system is expected to deliver reliable and detailed strain measurements. This accuracy is crucial for assessing the structural integrity and making reliable decisions regarding maintenance and safety [4]. Accurate strain measurement is essential for detecting subtle changes in structural performance that could indicate early signs of damage or degradation [6].

- **Enhancing Data Handling and Processing**

An important objective is to enhance the data handling and processing capabilities of the monitoring system. The integration of FPGAs allows for high-speed data acquisition and real-time processing, significantly improving the efficiency of data management. This includes the ability to process large volumes of data swiftly and accurately, facilitating immediate analysis and response [6]. Enhanced data handling capabilities ensure that the system can manage complex datasets from multiple sensors, providing a comprehensive overview of structural health [5].

- **Use of Advanced Technologies for Enhanced Monitoring**

The system incorporates advanced technologies such as the NEEK MAX10 FPGA Development Kit, NIOS II processor, and high-precision sensors to enhance monitoring capabilities. These technologies provide the necessary computational power and flexibility to handle complex monitoring tasks, ensuring a reliable monitoring solution. The use of these advanced technologies also facilitates the integration of new features and improvements as they become available, keeping the monitoring system at the forefront of technological innovation.

- **Real-Time Data Analysis and Reporting**

A key objective is to enable real-time data analysis and reporting. This involves using advanced software tools to program and control the data acquisition systems, as well as to process the collected data. Real-time analysis allows for the immediate detection of anomalies, facilitating timely interventions and corrective actions. This capability supports proactive maintenance strategies, reduces the risk of unexpected failures, and extends the lifespan of structures by allowing for timely repairs and maintenance.

2 Chapter 2nd : Essential Background

2.1 Introduction

This chapter provides an overview of the fundamental concepts and technologies essential for understanding strain monitoring systems. It begins with an introduction to strain monitoring, explaining its importance in engineering applications. The types of strain gauges and their working principles are then discussed, highlighting their role in measuring deformation.

Sequentially, the signal conditioning background, which includes amplifying and filtering signals from strain gauges, is covered. The chapter also explores data acquisition systems, focusing on their role in converting analog signals to digital data for analysis. The use of Field-Programmable Gate Arrays (FPGAs) in strain monitoring systems is examined, emphasizing their high-speed processing capabilities.

Real-time data processing and visualization techniques are addressed, detailing how raw data is converted into interpretable graphical representations. The chapter concludes with a summary, encapsulating the key points discussed, and providing a foundation for the advanced topics in the subsequent chapters.

2.2 Strain Monitoring

Strain monitoring involves measuring the deformation that a material or structure experiences under static or dynamic loading, providing critical insights into its structural integrity and performance. Strain is defined as the deformation per unit length of a material when subjected to an external force and is typically expressed as microstrain ($\mu\varepsilon$), where one microstrain equals 10^{-6} of a strain. The primary components of a strain monitoring system involve strain gauges, signal conditioning circuits, data acquisition systems, and data processing and visualization tools. Strain gauges, which convert mechanical deformation into electrical resistance variations, are the most commonly used sensors for this purpose.

The history of strain monitoring dates back to the early 20th century when engineers and scientists began exploring methods to quantify material deformation. A significant milestone was the development of the electrical resistance strain gauge in the 1930s by Edward E. Simmons and Arthur C. Ruge. This invention marked a turning point, leading to numerous advancements that have improved the accuracy, sensitivity, and durability of strain gauges. In recent decades, the integration of digital electronics, microprocessors, and data communication technologies has further revolutionized strain monitoring systems, allowing for real-time data acquisition, processing, and visualization.

Strain monitoring is employed across various industries, each with its specific requirements and challenges. In civil engineering, strain monitoring is crucial for assessing the structural health of bridges, buildings, dams, and other infrastructure, helping to identify stress areas and potential failure points, thus enabling timely maintenance and repairs. In the aerospace industry, it ensures the integrity of aircraft components by detecting fatigue and structural damage, thereby enhancing safety and performance [7]. In the automotive sector, strain gauges test and validate the strength and durability of components such as chassis, suspension systems, and engines. Monitoring strain in industrial machinery helps prevent catastrophic failures and extend the operational lifespan of heavy equipment [8]. Additionally, strain monitoring is used in marine engineering to assess the structural health of ships and offshore structures under harsh marine conditions.

2.3 Data Acquisition Systems

Data acquisition systems (DAS) are essential components in strain monitoring, enabling the collection, processing, and storage of data from strain gauges and other sensors. These systems play a critical role in converting the analog signals generated by sensors into digital data that can be analyzed and interpreted.

A typical data acquisition system consists of several key components, including sensors, signal conditioners, analog-to-digital converters (ADCs), and data storage or processing units. The process begins with the sensors, such as strain gauges, which detect mechanical deformation and convert it into an electrical signal. This signal is then passed through the signal conditioning unit, where it is amplified and filtered to enhance its quality and ensure it is within the appropriate range for the ADC. The ADC is a crucial component that converts the conditioned analog signal into a digital format. This conversion is necessary because digital data is easier to process, analyze, and store using modern computing systems. The resolution of an ADC, typically measured in bits, determines the precision of the conversion. Higher resolution ADCs provide more accurate digital representations of the analog signals.

Once converted to digital form, the data can be processed in real-time or stored for later analysis. Real-time processing involves immediate analysis and visualization of the data, allowing for timely decision-making and intervention if necessary. For instance, in this thesis, the FPGA on the NEEK MAX10 Development Kit is used for real-time data processing and visualization, leveraging its high-speed processing capabilities. Data storage is another important aspect of data acquisition systems. The collected data can be stored locally on devices such as SD cards or transmitted to remote servers for centralized storage and analysis. Effective data management ensures that the information remains organized and accessible for future reference and detailed analysis.

Modern data acquisition systems integrate various features to enhance their performance and usability. These include high-speed data sampling, multi-channel inputs, and advanced signal processing capabilities. Furthermore, the integration of wireless technology has enabled remote monitoring and control, significantly expanding the applications of DAS in various fields [9]. Advancements in data acquisition technology have significantly enhanced the capabilities of strain monitoring systems. Modern DAS offer higher resolution, faster sampling rates, and more robust data management features, making them indispensable tools for engineers and researchers aiming to ensure the safety and reliability of critical structures and components.

2.4 Strain Gauges

Strain gauges are critical components in the field of structural health monitoring, playing a pivotal role in measuring the deformation or strain of materials under various loads. These devices convert mechanical deformation into electrical resistance variation, which can then be accurately measured and analyzed. The fundamental principle of a strain gauge is based on the property that the electrical resistance of a conductor changes when it is stretched or compressed.

Strain gauges come in several types, including resistive, capacitive, and optical. However, resistive strain gauges are the most commonly used due to their simplicity, accuracy, and cost-effectiveness. These gauges consist of an insulating flexible backing that supports a metallic foil pattern. When the material to which the gauge is attached deforms, the foil deforms as well, causing its electrical resistance to change. This change in resistance is proportional to the strain and can be measured using a Wheatstone bridge circuit.

The image provides a detailed view of a typical resistive strain gauge. The key components include the grid, which is the active element, usually made of metallic foil, that undergoes deformation and the carrier, an insulating layer that supports the metallic grid. The terminals are, the connections that allow the strain gauge to be connected to external circuits for signal measurement. Another important part is the protective coating, it is a layer that protects the gauge from environmental factors like moisture and mechanical damage. Last, but not least, the backing provides additional support and stability to the entire gauge structure.

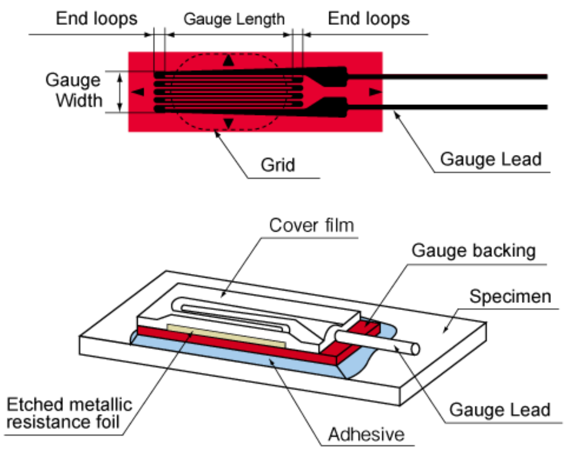


Figure 2: Resistive Strain Gauge

The key parameters of strain gauges include gauge factor, linearity, hysteresis, and temperature sensitivity. The gauge factor is a measure of the sensitivity of the gauge and is defined as the ratio of relative change in electrical resistance to the mechanical strain. Linearity refers to the degree to which the output of the strain gauge is directly proportional to the strain. Hysteresis is the difference in the output signal when the strain is increased and then decreased, while temperature sensitivity addresses how the gauge's output varies with temperature changes.

Selecting the appropriate strain gauge for a specific application involves considering factors such as the material of the structure being monitored, the expected range of strain, environmental conditions, and the required accuracy. For instance, in high-temperature environments, strain gauges with better temperature stability are preferred.

The KYOWA KFG-5-120-C1-11L3M2R is an example of a resistive foil strain gauge used in this dissertation. This specific model has a gauge factor of approximately 2.1, which indicates its sensitivity. It has a resistance of 120Ω and is constructed with a polyimide resin backing for durability and flexibility. Additionally, it features temperature compensation, making it suitable for various materials and ensuring accurate readings over a range of temperatures. The strain gauge typically comes with pre-attached lead wires, simplifying the installation process. [10]

2.5 Wheatston (Resistor) Bridge

A Wheatstone bridge is an electrical circuit designed to measure an unknown electrical resistance by balancing two legs of a bridge circuit. It is named after Sir Charles Wheatstone, although it was actually invented by Samuel Hunter Christie in 1833. This bridge circuit consists of four resistors arranged in a diamond shape, with a voltage source applied across the bridge.

In a typical Wheatstone bridge configuration, two resistors are known and fixed, one is adjustable, and the fourth is the unknown resistor whose value needs to be determined. The bridge operates on the principle of null measurement, where the goal is to adjust the known resistors until the voltage difference between the two legs of the bridge is zero. At this point of balance, the ratio of the two known resistances equals the ratio of the unknown resistor and the adjustable resistor. By using this ratio, the value of the unknown resistance can be calculated with high precision.

The Wheatstone bridge is particularly useful in applications where small changes in resistance need to be measured accurately. One common application is in strain gauge sensors, which are used to measure strain (deformation) in materials. In this setup, the strain gauge, which changes resistance when deformed, forms part of the bridge. Even minor changes in resistance due to strain can be detected accurately using the Wheatstone bridge configuration. [11]

The sensitivity and accuracy of the Wheatstone bridge make it an essential tool in various fields such as electrical engineering, physics, and materials science. It is widely used in laboratories and industrial settings for precise measurements of electrical components and for calibration purposes. Additionally, the Wheatstone bridge is foundational in the development of other measurement devices and circuits, underpinning technologies that require precise resistance measurements. Its ability to detect minute resistance changes has made it a crucial component in modern sensor technology and instrumentation.

2.6 Signal Conditioning

Signal conditioning is a crucial step in the process of strain monitoring, ensuring that the signals obtained from the Wheatstone bridge are accurately amplified and filtered for subsequent data acquisition and analysis. This process involves several key functions, including amplification, filtering, and sometimes isolation, to enhance the quality and usability of the electrical signals generated by strain gauges.

A Wheatsone bridge outputs low electrical signals that are not suitable for direct processing by data acquisition systems. These signals typically require amplification to increase their amplitude to a level that can be accurately digitized. Amplification is achieved using operational amplifiers, which are designed to boost the signal strength without significantly altering its characteristics. For instance, in this thesis, the signal conditioning unit incorporates Texas Instruments TLV9002D dual operational amplifiers to ensure that the small voltage changes produced by the strain gauges are sufficiently amplified.

Filtering is another essential function of signal conditioning. It involves removing unwanted noise and interference from the signal, which could otherwise obscure the strain measurements. Filters can be designed to pass certain frequency ranges while attenuating others, depending on the specific requirements of the application. For example, low-pass filters are commonly used to eliminate high-frequency noise that is not relevant to the strain measurements [12]. In some cases, signal conditioning may also include isolation to protect both the sensors and the data acquisition system from high voltages or other electrical hazards. Isolation techniques ensure that the signal path is electrically separated from potential sources of interference or damage.

The design and implementation of the signal conditioning circuit are critical for achieving accurate and reliable strain measurements. In this thesis, the signal conditioning unit is custom-built to meet the specific requirements of the strain monitoring system. It includes precision resistors and capacitors for fine-tuning the amplification and filtering processes, ensuring that the conditioned signals accurately represent the strain experienced by the monitored structure.

There are several types of signal conditioners, each designed for specific applications and requirements [13]. Amplifiers increase the amplitude of low-level signals to match the input range of data acquisition systems. They are essential for boosting weak signals from sensors like strain gauges. Filters are used to remove unwanted noise from the signal. They can be designed as low-pass, high-pass, band-pass, or band-stop filters, depending on the frequency components that need to be attenuated or passed. Isolators protect both the sensors and the data acquisition system from high voltages or electrical noise. They ensure that the signal path is electrically separated, preventing damage and improving measurement accuracy. Converters change the signals from analog to digital, current to voltage, or frequency to voltage. Converters are vital in systems where the signal type must be changed to match the data acquisition requirements. Linearizers are used to correct non-linear signals from sensors, providing a linear output that is easier to analyze and interpret. This is particularly useful for sensors with non-linear characteristics. These types of signal conditioners are integral to the performance and reliability of strain monitoring systems, ensuring accurate, noise-free, and properly scaled signals for analysis.

2.7 Field-Programmable Gate Arrays (FPGAs)

Field-Programmable Gate Arrays (FPGAs) are integrated circuits that can be programmed by the user after manufacturing, offering a high degree of flexibility and reconfigurability. These devices are particularly well-suited for applications requiring parallel processing and high-speed data handling, making them ideal for real-time strain monitoring systems. FPGAs consist of an array of programmable logic blocks and interconnects that can be configured to perform complex computational tasks. Unlike traditional microcontrollers or processors, which execute instructions sequentially, FPGAs can execute multiple operations in parallel, significantly enhancing processing speed and efficiency. This parallel processing capability is crucial for applications involving high-speed data acquisition and real-time analysis, such as those in strain monitoring systems. [14]

The architecture of an FPGA includes several key components: Logic Blocks, which are the basic building units of an FPGA, capable of implementing simple logic functions. Each block contains a few logic gates and a small amount of memory, typically in the form of look-up tables (LUTs) and flip-flops. Programmable interconnects connect the logic blocks, allowing for the creation of complex circuits. The flexibility of the interconnects enables the user to design custom pathways for signal routing. Input/Output blocks interface the FPGA with external components, such as sensors, actuators, and other peripherals. These blocks are essential for integrating the FPGA into larger systems. FPGAs often include embedded memory blocks that provide temporary storage for data and instructions. This feature is particularly useful for applications requiring large amounts of data to be processed quickly. [15]

The use of FPGAs in strain monitoring systems offers several advantages. Their high-speed processing capabilities enable real-time data acquisition and analysis, which is essential for monitoring dynamic structural changes. Additionally, the reconfigurable nature of FPGAs allows for system updates and modifications without the need for hardware changes, providing a flexible platform for developing and improving monitoring systems.

- **NEEK MAX10 FPGA Development Kit**

In this thesis, the NEEK MAX10 FPGA Development Kit is utilized for real-time strain monitoring. The NEEK MAX10 FPGA Development Kit includes several notable features. It is based on the Altera MAX 10 FPGA, which offers non-volatile storage, dual-configuration flash, and analog-to-digital converters (ADCs) integrated directly into the chip. The development kit also includes a rich set of peripherals such as an LCD touchscreen, USB ports, and various I/O interfaces, making it an ideal platform for developing embedded systems with advanced user interfaces and connectivity options. The integrated ADCs in the MAX 10 FPGA provide high-speed data conversion capabilities, crucial for real-time monitoring applications. The on-board flash memory allows for the storage of configuration data and user applications, ensuring that the system can be reconfigured as needed without requiring additional hardware modifications. These features make the NEEK MAX10 FPGA Development Kit a versatile and powerful tool for implementing sophisticated strain monitoring systems. [3]

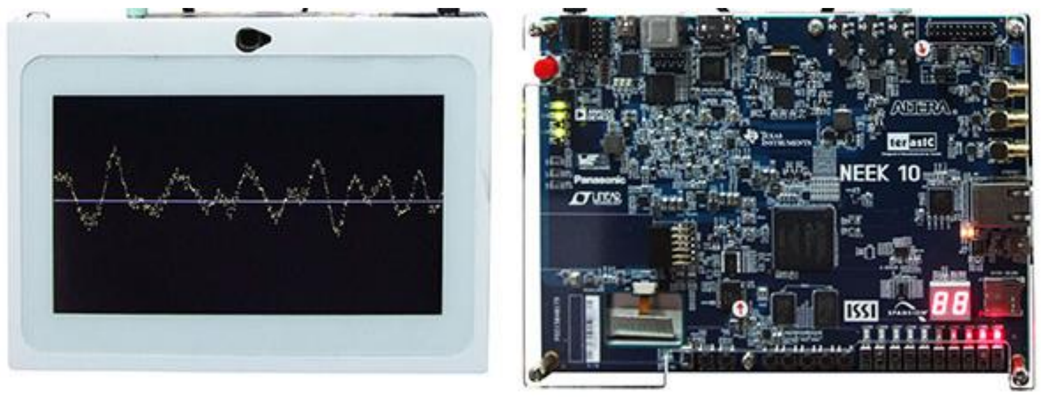


Figure 3: NEEK MAX10 FPGA Development Kit

FPGAs are widely used across various industries due to their versatility and performance. In aerospace, they are employed for tasks such as avionics systems, radar processing, and satellite communications. The automotive industry uses FPGAs for advanced driver-assistance systems (ADAS), infotainment, and engine control. In telecommunications, FPGAs are used for high-speed data processing and network infrastructure. The flexibility and reconfigurability of FPGAs also make them suitable for research and development in fields such as artificial intelligence and machine learning.

2.8 Real-Time Data Processing and Visualization

Real-time data processing and visualization enable the immediate analysis and interpretation of data collected from sensors. This capability is essential for applications where timely decision-making is crucial. Real-time data processing involves the continuous collection and analysis of data as it is acquired, without significant delays. This is typically achieved using high-performance hardware and optimized software algorithms that can handle large volumes of data at high speeds. In strain monitoring systems, real-time processing allows for the detection of anomalies or significant changes in the structural integrity of monitored assets, enabling prompt corrective actions.

Visualization is the process of converting raw data into graphical representations that are easy to interpret. Effective visualization techniques can reveal patterns, trends, and anomalies in the data that might not be immediately apparent from raw numerical data alone. Common visualization methods in strain monitoring include time-series plots, stress-strain curves.

The system developed in this thesis has two versions: the Serial Communication Version and the Autonomous Version. In the Serial Communication Version, the data visualization is managed through MATLAB. The FPGA collects and processes the strain data, which is then transmitted to a computer via a serial communication interface. This setup allows for flexible and detailed data analysis on a powerful computing platform, making it suitable for environments where direct interaction with the system is required.

In contrast, the Autonomous Version operates independently, leveraging the Nios II processor integrated within the FPGA and the Eclipse development environment for on-board data processing and display. In this version, the processed data is visualized directly on the NEEK MAX10's LCD screen, enabling real-time monitoring without the need for an external computer. This setup is ideal for scenarios requiring continuous monitoring in remote or inaccessible locations. The real-time visualization on the LCD screen provides immediate feedback to the user, allowing for quick assessment of the structural health of the monitored system.

The combination of real-time data processing and visualization enhances the overall effectiveness of strain monitoring systems. By providing immediate and clear insights into the structural health of assets, these technologies support proactive maintenance and timely interventions, ultimately improving safety and reliability. The integration of real-time processing and visualization in strain monitoring systems is also beneficial for long-term data analysis. Historical data can be stored and later reviewed to identify trends over time, assess the impact of various stressors, and predict future performance. This comprehensive approach to data management and analysis ensures that all relevant information is considered in maintaining and improving structural health.

2.9 Summary of Essential Background

The essential background for strain monitoring systems encompasses various components and concepts, each contributing to the overall effectiveness and accuracy of the monitoring process. This chapter has provided an overview of the foundational elements necessary for understanding and implementing an advanced strain monitoring system.

Specifically, introduction is made for the:

- ✓ Strain sensors (strain gauges)
- ✓ Wheatstone (resistor) bridge
- ✓ Signal conditioning unit
- ✓ Analogue to Digital conversion
- ✓ Data storage
- ✓ Strain translation visualization

3 Chapter 3rd : Hardware Prototyping

This chapter details the process of developing and prototyping critical components for precise strain measurement. It encompasses the design and refinement of the 8-channel strain gauge amplifier PCB and the complementary Resistor Bridge board. Starting with initial single-channel prototypes, the chapter outlines the design iterations and extensive testing that led to the final multi-channel amplifier. Advanced design software and a combination of automated and manual assembly techniques were utilized in the Electronic Devices and Materials Laboratory (EDML) within the Sector of Electronics and Materials, Department of Electrical and Electronic Engineering, School of Engineering, University of West Attica. Additionally, the chapter details the Resistor Bridge board's role in signal conditioning, covering its design, assembly, and integration with the amplifier circuit.

3.1 Initial Prototyping and Development of the 8-Channel Strain Gauge Amplifier PCB

The initial design phase involved creating a single-channel strain gauge amplifier to validate the concept and ensure accurate signal processing. The final design evolved into an 8-channel amplifier PCB, accommodating multiple strain gauges for comprehensive strain measurement.

The operational principles of the amplifier circuit are based on the Wheatstone bridge configuration. Below are the key equations and calculations that outline the functionality of the amplifier circuit, using the following schematic:

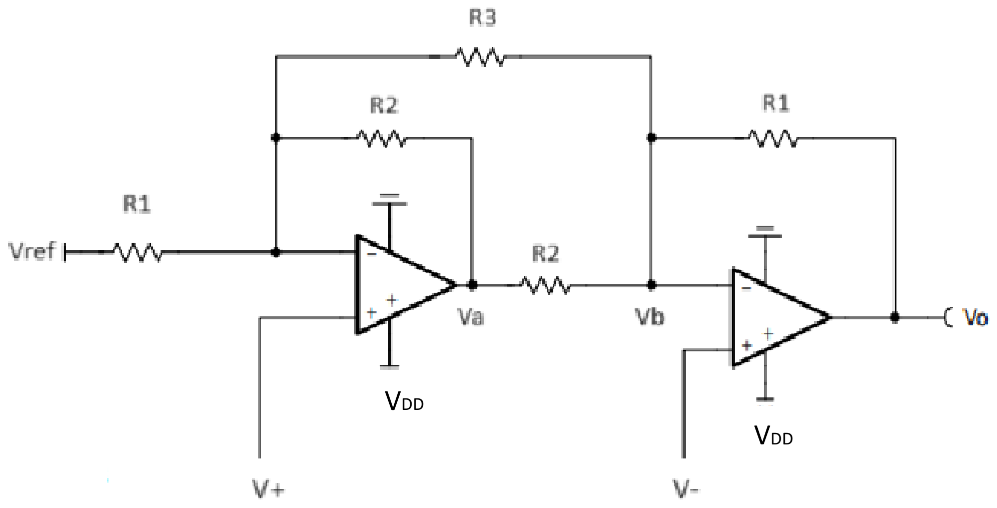


Figure 4: Schematic of the amplifier circuit

$$V^{+} * (G1 + G2 + G3) - Vref * G1 - Va * G2 - Vb * G3 = 0 \quad (1)$$

$$-V^{-} * (G1 + G2 + G3) + Va * G2 + V^{+} * G3 + V0 * G1 = 0 \quad (2)$$

So, (1) +(2) $\rightarrow$

$$(V^{+} - V^{-}) * (G1 + G2 + G3) - Vref * G1 - Vb * G3 + V^{+} * G3 + V0 * G1 = 0 \quad (3)$$

But since $V^{-} = Vb$

(3)→ $(V^{+} - V^{-}) * (G1 + G2 + G3 + G3) - Vref * G1 = -V0 * G1$

$$(V^{+} - V^{-}) * \frac{(G1 + G2 + 2G3)}{G1} - Vref = -V0$$

Given the resistor values: R1 = 20kΩ, R2= 5.1kΩ and R3 = 40.15Ω we can calculate

$G1 = \frac{1}{20*10^3} = 5 * 10^{-5}S$

$G2 = \frac{1}{5.1*10^3} = 1.96 * 10^{-4}S$

$G3 = \frac{1}{40.15} = 0.0249S$

And

$$A = 1 + \frac{R1}{R2} + 2\frac{R1}{R3} = 1 + \frac{20 * 10^3}{5.1 * 10^3} + 2 * \frac{20 * 10^3}{40.15} = 1 + 3.92 + 0.996 * 1000 \cong 1001$$

These calculations are based on the guidelines provided by Texas Instruments for the TLV9002 operational amplifier [16]. Since resistances with a value of 40.15Ω are not available on the market, we used the closest available value, which is 40.2Ω, resulting in a gain of approximately 999.92. The design ensures accurate signal amplification and minimal error due to the high precision and stability of the resistor values used.

3.1.1 First Attempt: Single-Channel Breadboard Prototype

The first attempt to build the strain gauge amplifier circuit was conducted on a breadboard. This approach allowed for rapid prototyping and easy modification of the circuit components. Using a breadboard, we were able to test and refine the circuit design iteratively. This stage was crucial for identifying potential issues and making necessary adjustments before committing to a printed circuit board (PCB) design.

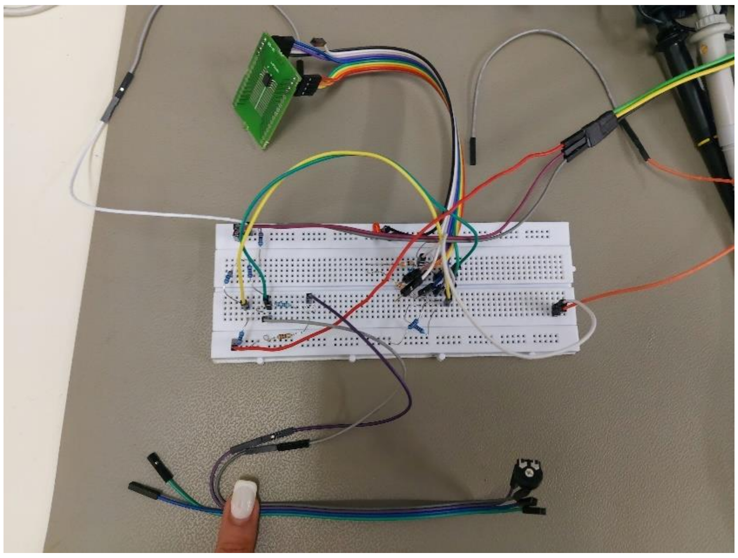


Figure 5: Breadboard Prototype

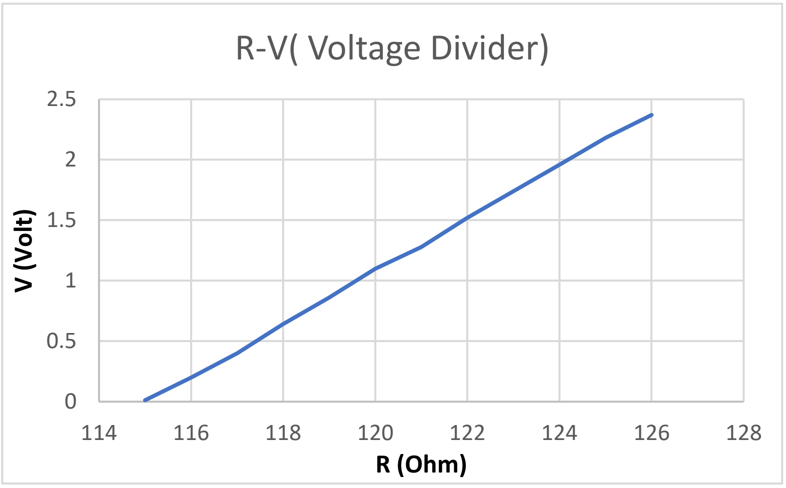
This method provided flexibility for modifications and troubleshooting. The breadboard setup included the operational amplifier which was placed at the center of the breadboard, with connections made to the relevant signal paths. Precision resistors and capacitors were incorporated to maintain signal integrity and jumper wires were used extensively to create the necessary connections between components, developing the single-channel amplifier circuit on a temporary platform.

The breadboard prototype allowed for rapid adjustments and testing, facilitating the identification and rectification of issues that were not apparent in the initial PCB version. This approach ensured that the core functionality of the amplifier circuit could be thoroughly vetted before proceeding to the final multi-channel design.

To validate the design, a series of measurements were conducted by systematically adjusting a potentiometer. These measurements were essential to calibrate the system and understand the correlation between resistance and voltage output, which is vital for accurate strain monitoring. A voltage divider circuit was employed, enabling controlled variation of the voltage across the potentiometer and providing a consistent method for recording voltage changes corresponding to different resistance values. The voltage range used was from 0V to 2.5/5V (with and without the voltage divider), and the resistance range varied between 115Ω to 117Ω. The data collected during these measurements are presented below:

- **Measurement Set 1: Step Size 1Ω, 2.5V/5V**

R (Ohm)	V (V)	V*2 (V)
115	0.012	0.024
116	0.2	0.4
117	0.4	0.8
118	0.64	1.28
119	0.86	1.72
120	1.1	2.2
121	1.28	2.56
122	1.52	3.04
123	1.74	3.48
124	1.96	3.92
125	2.18	4.36
126	2.37	4.74



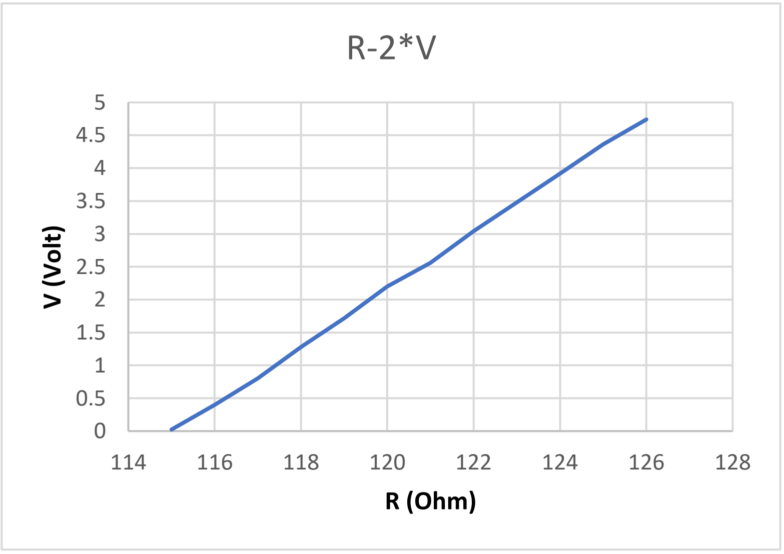
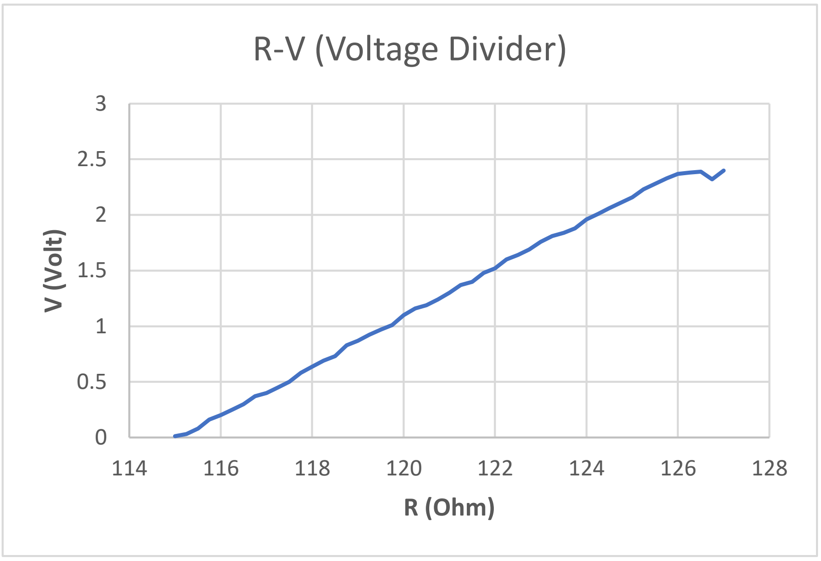


Figure 6: R-V measurements Set 1

- **Measurement Set 2: Step Size 0.25Ω, 2.5V/5V**

R (Ohm)	V (V)	V*2 (V)
115	0.012	0.024
115.25	0.03	0.06
115.5	0.08	0.16
115.75	0.16	0.32
116	0.2	0.4
116.25	0.25	0.5
116.5	0.3	0.6
116.75	0.37	0.74
117	0.4	0.8
117.25	0.45	0.9
117.5	0.5	1
117.75	0.58	1.16
118	0.635	1.27
118.25	0.69	1.38
118.5	0.73	1.46
118.75	0.83	1.66
119	0.87	1.74
119.25	0.925	1.85
119.5	0.97	1.94
119.75	1.01	2.02
120	1.1	2.2
120.25	1.16	2.32
120.5	1.19	2.38
120.75	1.24	2.48
121	1.3	2.6
121.25	1.37	2.74
121.5	1.4	2.8
121.75	1.48	2.96
122	1.52	3.04
122.25	1.6	3.2
122.5	1.64	3.28
122.75	1.69	3.38
123	1.76	3.52
123.25	1.81	3.62
123.5	1.84	3.68
123.75	1.88	3.76
124	1.96	3.92
124.25	2.01	4.02
124.5	2.06	4.12
124.75	2.11	4.22
125	2.16	4.32
125.25	2.23	4.46
125.5	2.28	4.56
125.75	2.33	4.66
126	2.37	4.74



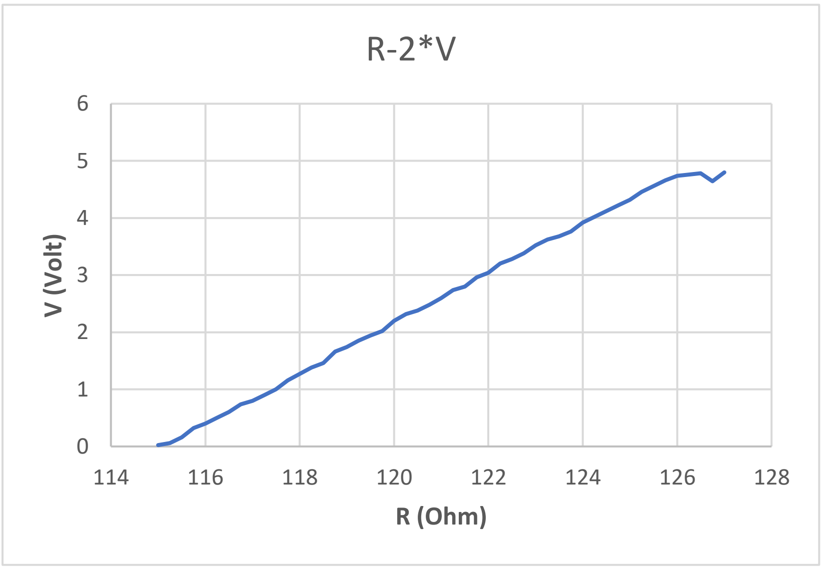


Figure 7: R-V measurements Set 2

3.1.2 Second Attempt: Single-Channel Prototype

The second prototyping attempt involved designing a single-channel strain gauge amplifier. This preliminary design was created to validate the basic amplification circuitry and to troubleshoot any potential issues before scaling up to an 8-channel configuration.

- **Schematic diagram**

This schematic (Figure 5) illustrates the strain gauge amplifier circuit using two operational amplifiers configured as an instrumentation amplifier. The circuit features a Wheatstone bridge (J3) to measure strain, with precision resistors (R1, R2, R3, R4, and R11) determining the gain and ensuring accurate signal amplification. Power is supplied through connectors J2 (+2.5V and +5V), with decoupling capacitors stabilizing the voltage supply, and the amplified output is sent to the FPGA via connector J1.

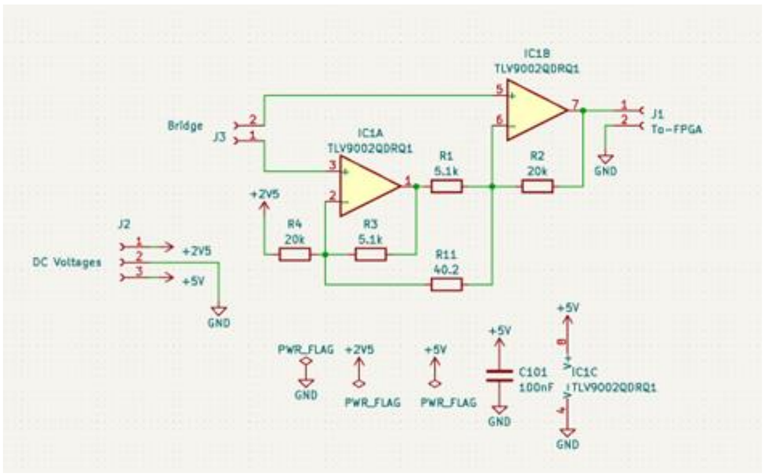


Figure 8: Schematic Diagram of the Single-Channel Prototype

- **PCB Design**

In the design, the focus was on a single channel to simplify the development process. The single-channel PCB includes the integrated circuit (IC) for the operational amplifier which is centrally placed, with its pins connected to the corresponding traces for signal input and output.

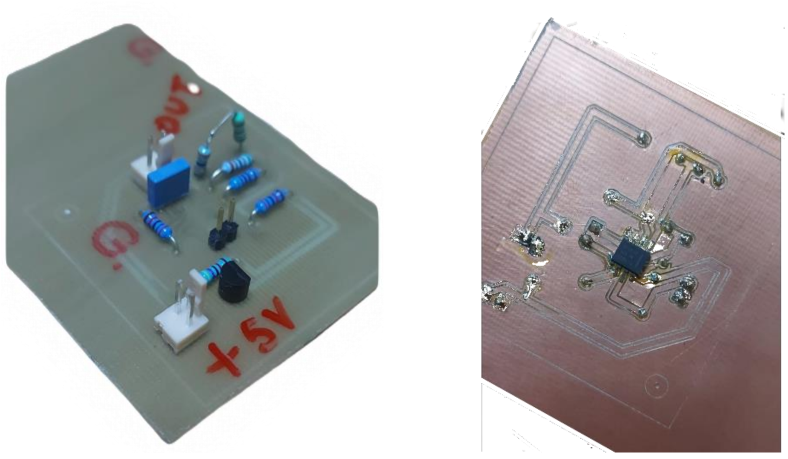


Figure 9: Single-Channel Prototype Top-Bottom View

Moreover, the board features precision resistors and capacitors essential for signal conditioning and noise filtering. The prototype includes basic connectors for power input (+5V and GND) and signal output (OUT).

The PCB was fabricated using a CNC cutter to accurately etch the copper traces. The Z-code for the CNC cutter was generated from the KiCad design files, ensuring precise and consistent milling of the PCB.

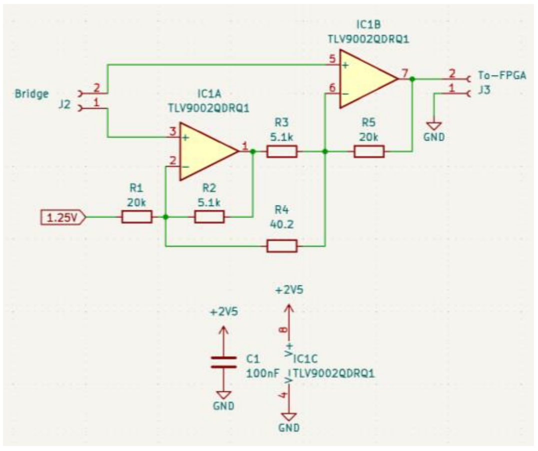
The assembled single-channel prototype was tested to verify its performance. Components were soldered onto the board, and initial tests were conducted to check for correct amplification of the strain gauge signals.

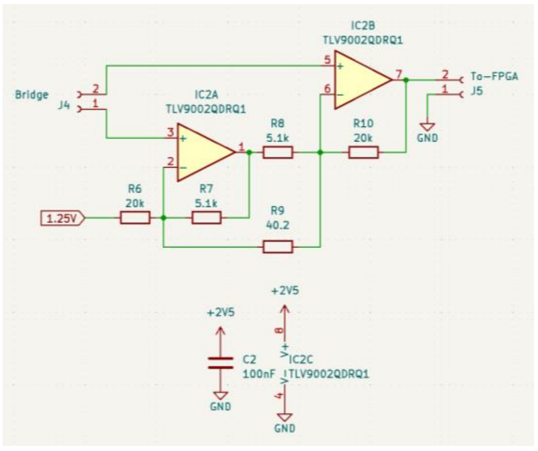
3.1.3 Final 8-Channel Strain Gauge Amplifier

The culmination of the iterative prototyping and testing process resulted in the development of the final 8-channel strain gauge amplifier. This advanced design integrates the insights gained from initial single-channel and breadboard prototypes, ensuring a robust and efficient solution for precise signal amplification. Developed using KiCad software, this PCB is tailored for applications requiring high accuracy and reliability in signal processing from multiple strain gauges.

- **Schematic diagrams**

The following schematic diagrams illustrate the detailed design for each of the eight channels in the final 8-channel strain gauge amplifier PCB. Each channel is built using similar components and configurations to ensure uniform performance across all channels.





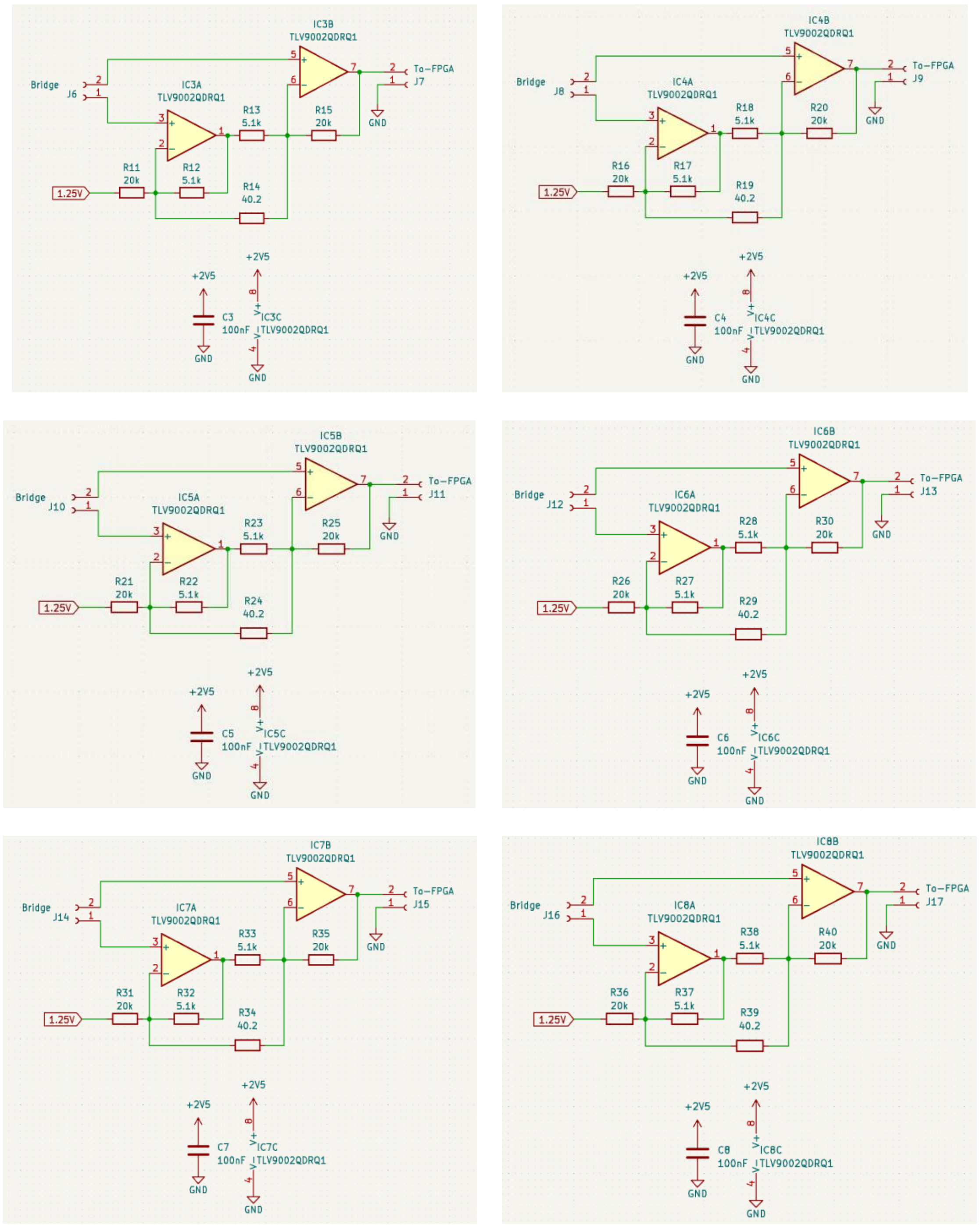


Figure 10: Schematic Diagrams of the 8-Channel Strain Gauge Amplifier

The root schematic serves as the foundational blueprint for each of the eight channels in the final strain gauge amplifier PCB. This schematic ensures uniformity and precision across all channels, providing a reliable and consistent framework for signal amplification.

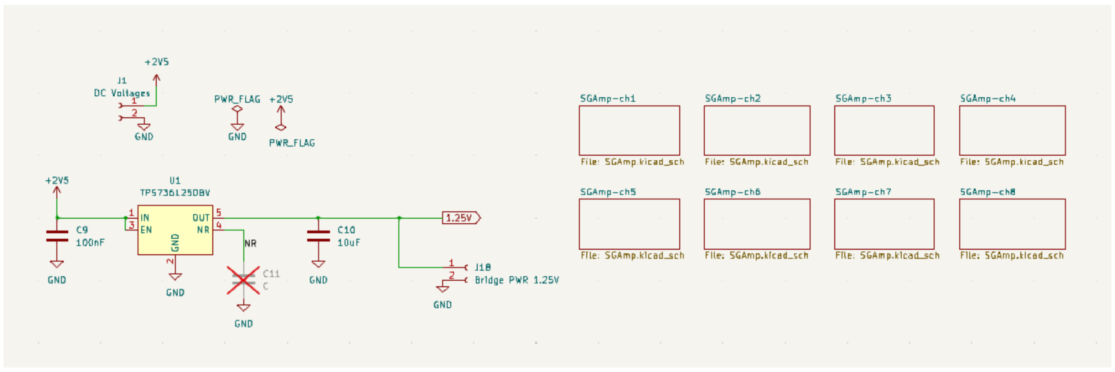


Figure 11: Root Schematic of the 8-Channel Strain Gauge Amplifier

The root schematic includes key components and connections. More analytically, each channel employs two TLV9002 operational amplifiers configured in a specific arrangement to amplify the strain gauge signals efficiently. The amplifiers are labeled IC1A/IC1B through IC8A/IC8B across the channels. Moreover, a series of resistors are used to set the gain and ensure proper operation of the amplifiers. Typical values include 20kΩ, 5.1kΩ, and 40.2Ω resistors, labeled R1 to R40 across all channels. These resistors are crucial for maintaining the accuracy and stability of the amplified signals. Each channel receives input from a strain gauge via connectors labeled J2 to J16. The amplified signal is transmitted to an FPGA through connectors labeled J3 to J17, enabling further processing or data acquisition. Each operational amplifier is powered by a +2.5V supply. Decoupling capacitors (100nF) labeled C1 to C8 are placed near the amplifiers to filter out noise and ensure stable operation. Last but not least, a 1.25V reference voltage is provided to each channel to maintain consistent signal amplification. This reference voltage is crucial for the proper functioning of the amplifiers, ensuring that the output signals are accurate and reliable.

The schematic is designed to be modular, with each channel following the same layout and component values. This modularity simplifies the design process and ensures that each channel performs identically, enhancing the overall reliability of the system.

- **PCB Design**

The top view of the PCB illustrates the layout and labeling of the components. The PCB is segmented into eight identical channels, labeled from Channel 1 to Channel 8, each dedicated to amplifying signals from a strain gauge input. Each channel incorporates several precision resistors essential for signal conditioning and amplification. For instance, Channel 1 includes resistors R1, R2, R3, R4, and R5.

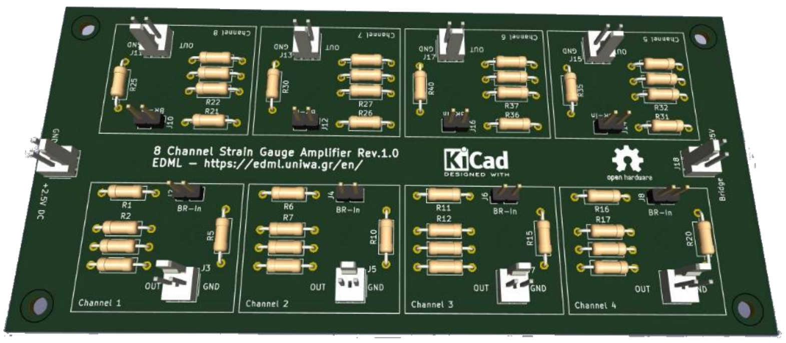

8 Channel Strain Gauge Amplifier Rev.1.0
EDML – https://edml.uniwa.gr/en/
KiCad
Channel 1
Channel 2
Channel 3
Channel 4
BR-In
OUT
GND
+2.5V DC
Bridge

Figure 12: 8-Channel Strain Gauge Amplifier Top View (KiCad 3D view)

Each channel features a connector (e.g., J2 in Channel 1) for strain gauge input and an OUT connector (e.g., J3 in Channel 1) for the amplified signal output. The PCB also includes power input terminals labeled +2.5V DC and GND, along with multiple ground points distributed across the board to ensure stable operation. Capacitors are strategically placed adjacent to the operational amplifiers to filter noise and stabilize the voltage supply. The silkscreen layer provides clear labels for each channel, component designations, and the PCB version information.

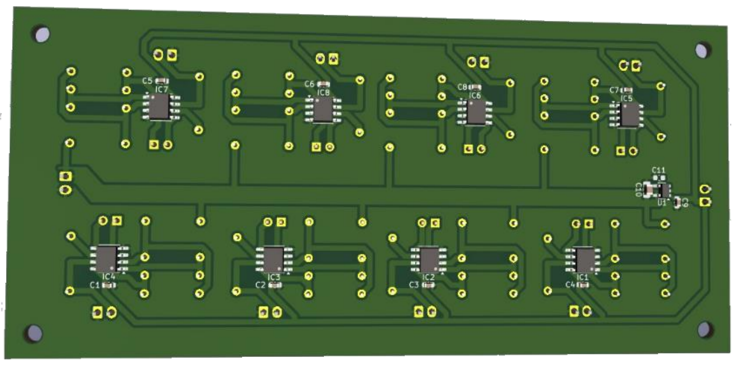


Figure 13: 8-Channel Strain Gauge Amplifier Bottom View (KiCad 3D view)

The bottom view of the PCB reveals the routing and placement of components on the underside. Each channel's operational amplifier integrated circuit (e.g., IC1-IC8) is visible, showing the detailed pin connections and layout. Decoupling capacitors (e.g., C1-C11) are strategically placed to ensure noise reduction and stable power supply. The copper traces are clearly visible, illustrating the interconnections between different components and channels. The routing is designed to minimize noise and interference. The bottom view also displays four mounting holes located at the corners, intended for securely attaching the PCB to an enclosure or framework.

This PCB design ensures efficient and accurate signal amplification from multiple strain gauges, making it suitable for various engineering applications, such as stress analysis and structural health monitoring. The meticulous arrangement of components and clear labeling facilitate easy assembly and troubleshooting.

- **Procedure for Implementing the 8-Channel Strain Gauge Amplifier Board**

The development of the 8-channel strain gauge amplifier PCB involved a meticulous process from design to final assembly.

Once the design was finalized, the PCB layout files were sent to a professional fabrication service. This step ensured high-quality printing and precise manufacturing of the PCB according to the design specifications. Along with the PCBs, stencils for solder paste application were also ordered. These stencils were crucial for accurately applying solder paste to the PCB pads.

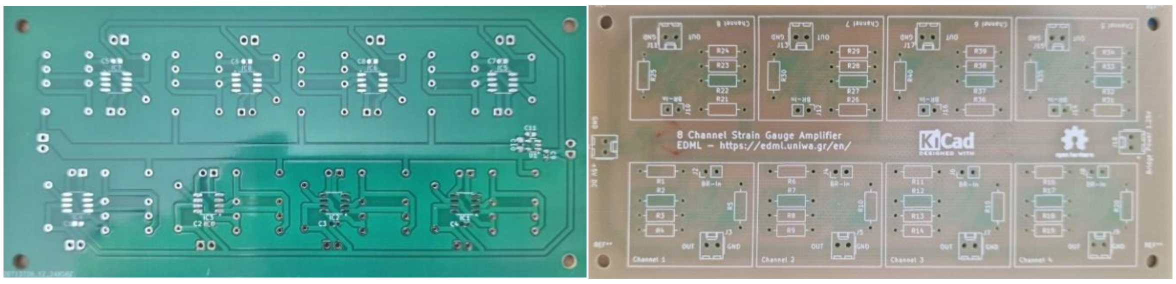

8 Channel Strain Gauge Amplifier
EDML – https://edml.uniwa.gr/en/
KiCad
Channel 1
Channel 2
Channel 3
Channel 4

Figure 14: Board of the 8-Channel Strain Gauge Amplifier

The components were initially placed on the PCB using a pick-and-place machine. This process ensured accurate placement of surface-mount devices (SMDs) on the board.

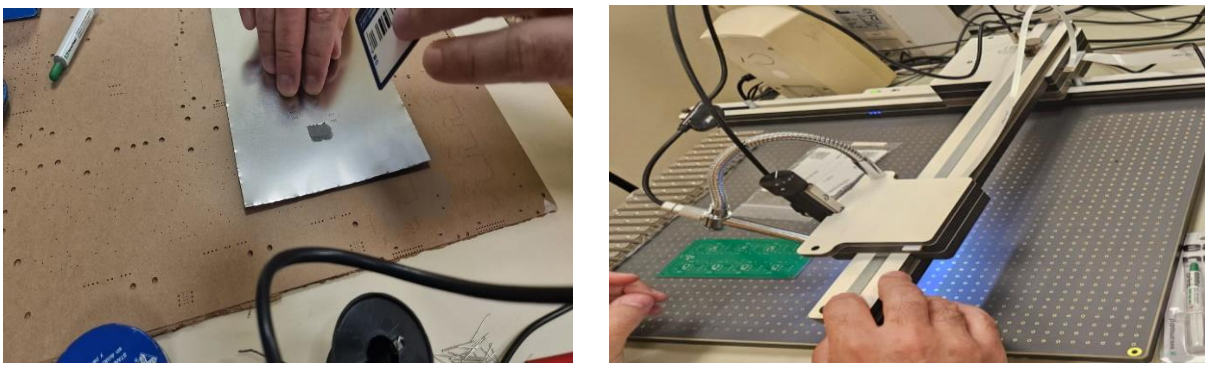


Figure 15: Solder paste application & Pick-and-place Process

After placing the components, the PCB was transferred to a reflow oven, the ProtoFlow E. The soldering process involves heating the board to a specific temperature to melt the solder paste, securing the components in place. This process ensures strong and reliable solder joints for the SMD components. In this procedure, the reflow oven heats the board through a precise temperature profile, involving preheat, soak, and reflow stages, followed by cooling.

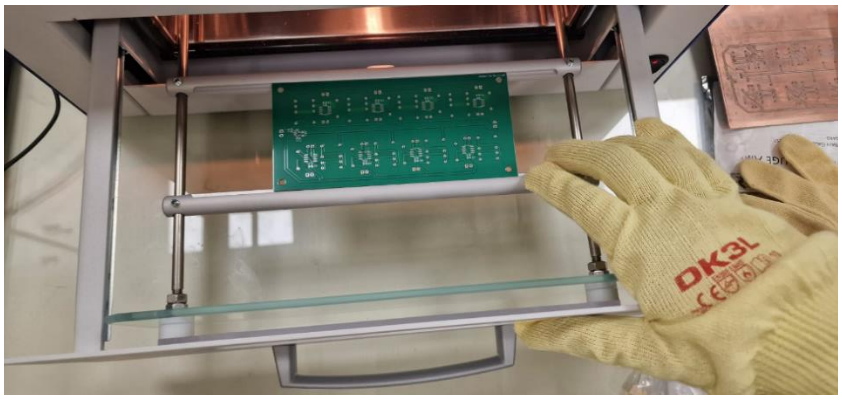


Figure 16: Reflow soldering process

After reflow soldering, any remaining components, particularly through-hole components, were soldered manually using a soldering iron. This step ensured that all components were securely attached and electrically connected.

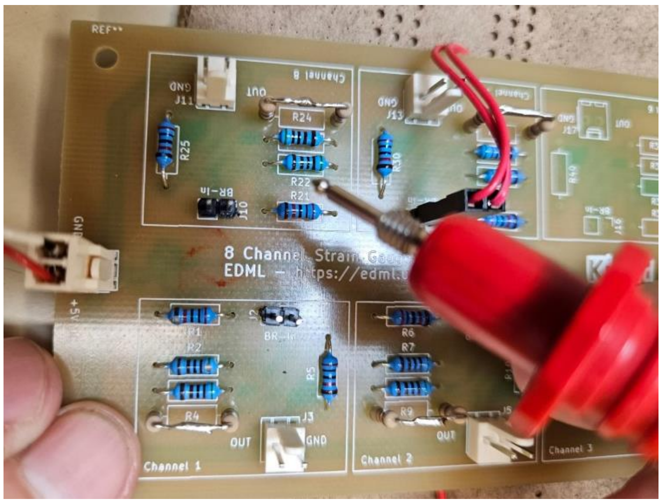


Figure 17: Manually Soldering process

After assembly, the PCB underwent thorough inspection and testing to verify the functionality and performance of each channel. This step ensured that the final product met the required specifications and was free of defects.

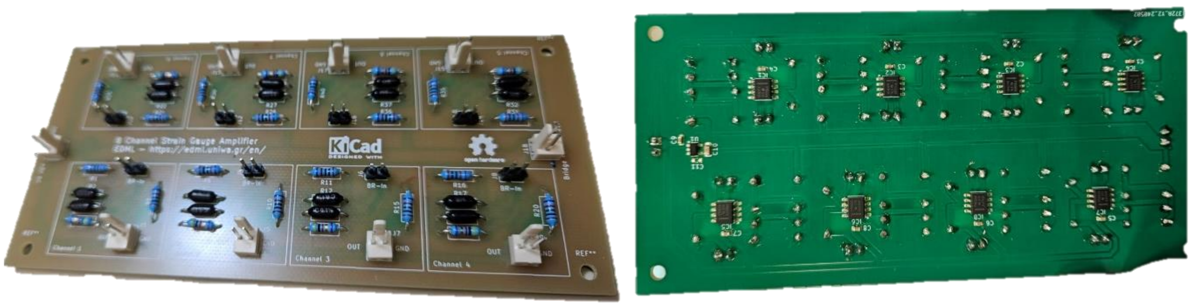


Figure 18: Final 8-Channel Strain Gauge Amplifier Board

The creation of the 8-channel strain gauge amplifier PCB involved a collaborative and detailed process, combining automated and manual assembly techniques. The high-quality fabrication, precise component placement using the pick-and-place machine, soldering in the reflow oven, and final manual soldering ensured the reliability and accuracy of the amplifier.

3.2 Initial Prototyping and Development of the Resistor Bridge

Following the detailed analysis of the 8-channel strain gauge amplifier, this section focuses on the complementary Resistor Bridge board, a critical component in the signal conditioning process. The Resistor Bridge board interfaces directly with the strain gauges, preparing the signals for accurate amplification. This ensures a seamless integration with the amplifier circuit, facilitating precise and reliable measurements in various engineering applications.

This following schematic shows the Wheatstone bridge configuration used to measure small changes in resistance. The bridge consists of four resistors R=120Ω and an additional resistor Ra=10*R = 1.2kΩa

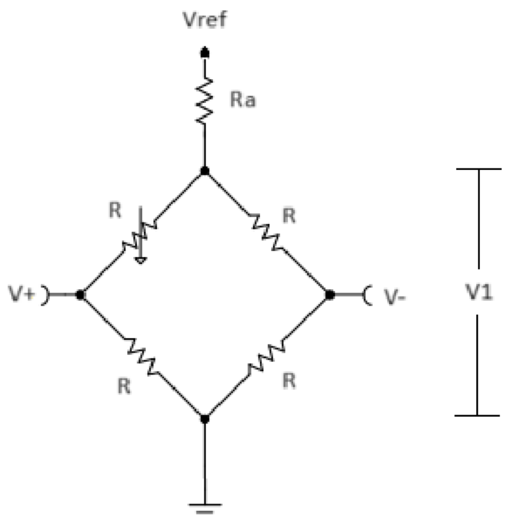


Figure 19: Schematic of the Weatstone (Resistor) Bridge

The calculations for the voltages in the bridge:

Voltage V1: $V1 = \frac{120*Vref}{1200+120} = 0.227V$

Voltage V^+: $V^- = Vref * \frac{1320}{1440} = 2.291V$

Voltages V^-: $V_1^+ = Vref * \frac{1305}{1425} = 2.289V$

$$V_2^+ = Vref * \frac{1335}{1455} = 2.294V$$

- **Schematic Diagram**

The schematic diagram of the Resistor Bridge board is integral for conditioning signals from strain gauges before amplification.

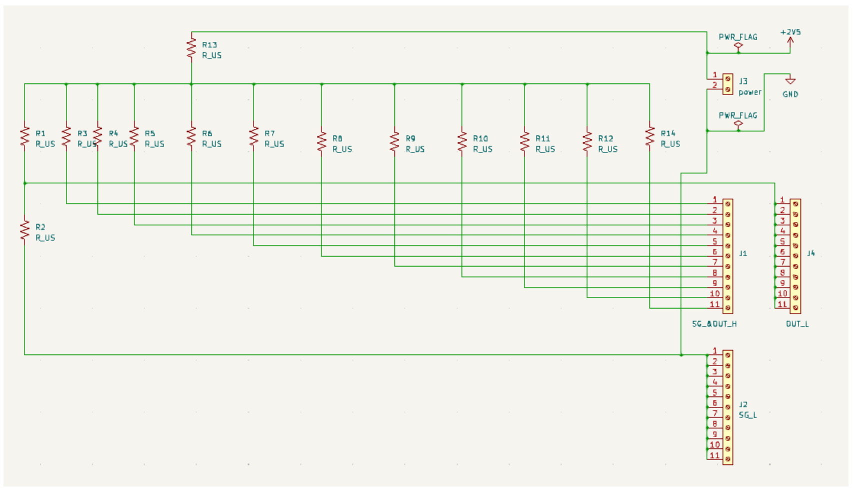


Figure 20: Resistor Bridge Schematic Diagram

Connectors J1 and J2 serve as the primary interfaces for the input signals from the strain gauges into the resistor bridge network. Each pin on these connectors corresponds to different input signals, which are routed to the respective resistor bridges for conditioning. J1 has a dual function, also serving as an output connector to transmit conditioned signals to the amplifier circuits. J3 is the power connector, supplying the necessary +3.3V power to the board and ensuring all components operate correctly. This connection distributes power throughout the circuit, supporting the operation of all the components. J4 is used to output the conditioned signals from the resistor bridges to the amplifier circuits. It consolidates the balanced signals and sends them to the next stage for amplification. The multiple pins on J4 correspond to conditioned output signals, labeled to indicate their respective signal lines sent to the amplifier.

The resistor bridges, consisting of precision resistors labeled R1 through R13, form the core of the signal conditioning network. These resistors balance the input signals from the strain gauges, ensuring each signal is properly conditioned before being output to the amplifier.

- **PCB Design**

The top view of the PCB features a series of precision resistors (R1 to R13) systematically aligned in a row. These resistors form the resistor bridges essential for balancing and conditioning the input signals from the strain gauges. The uniform placement ensures consistent performance across the board. Multiple connectors labeled J1 to J4 facilitate both input and output functions.

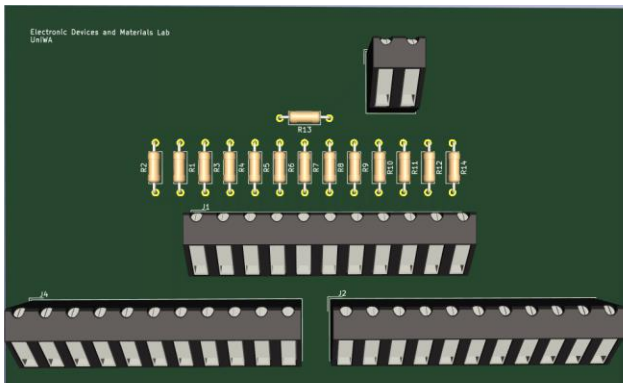


Figure 21: Resistor Bridge PCB Top View (KiCad 3D view)

The bottom view reveals the copper traces that establish the electrical connections between various components and pads. These traces are designed to minimize noise and interference. Additionally, the board includes strategically placed mounting holes to facilitate secure attachment to an enclosure or testing setup, ensuring stability during operation.

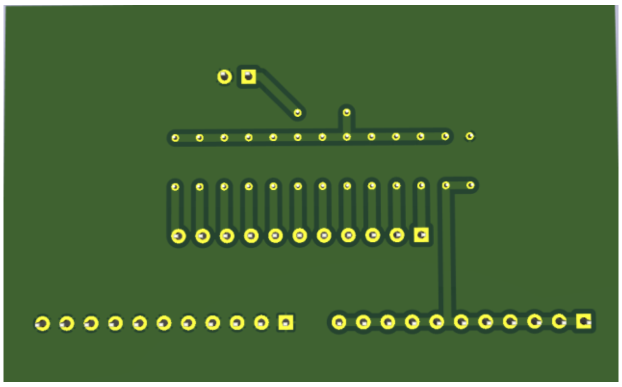


Figure 22: Resistor Bridge PCB Bottom View (KiCad 3D view)

- **Procedure for implementing the Resistor Bridge**

The development of the Resistor Bridge board followed a similar structured process with the amplification unit. Once the PCB layout files were finalized, they were sent to a professional fabrication service in China as well and the fabricated PCBs were then shipped to the Electronic Devices and Materials Laboratory (EDML) for further assembly.

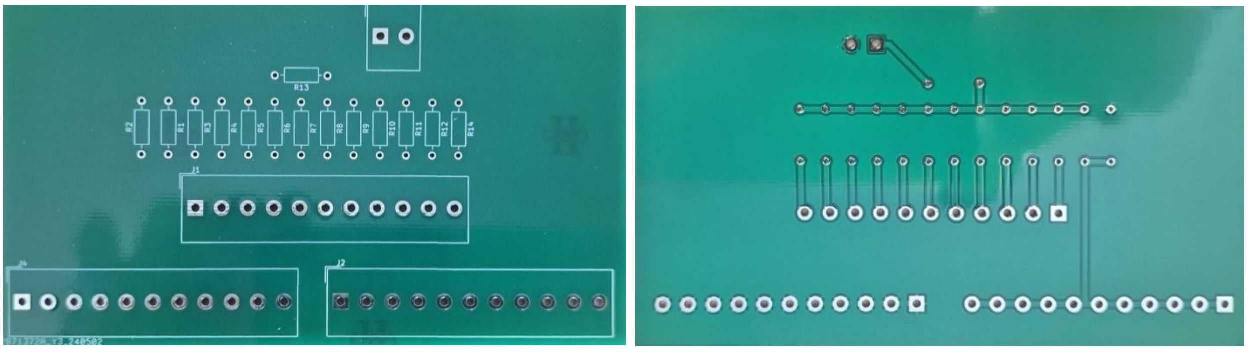


Figure 23: Resistor Bridge Board Top - Bottom

The assembly process was conducted entirely by hand. All the components, particularly the through-hole connectors and resistors, were soldered manually using a soldering iron.

Figure 24: Resistor Bridge Soldering Process

Once assembly was complete, the PCB underwent thorough inspection and testing to verify the functionality and performance of each resistor bridge. This step ensured the final product met all required specifications and was free from defects.

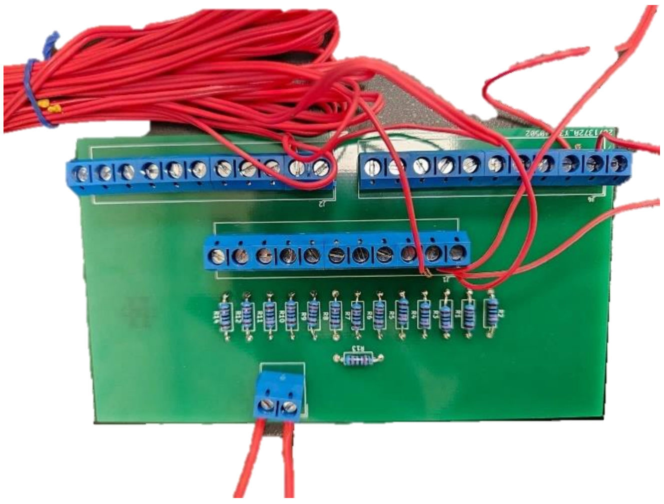


Figure 25: Resistor Bridge Final Board

In conclusion, the development and prototyping of the 8-channel strain gauge amplifier PCB and the complementary Resistor Bridge board have been thoroughly explored in this chapter. The transition from initial single-channel prototypes to the final multi-channel design involved multiple iterations and rigorous testing to ensure reliability and precision. The utilization of advanced design software and a blend of automated and manual assembly techniques within the Electronic Devices and Materials Laboratory (EDML) were pivotal in achieving the final design.

The Resistor Bridge board was highlighted for its essential role in signal conditioning, with a detailed account of its design, assembly, and integration with the amplifier circuit. These carefully developed components together form a robust signal amplification system, critical for accurate strain measurements in various engineering applications. The successful realization of this system underscores its potential for enhancing stress analysis and structural health monitoring, paving the way for future innovations in the field.

4 Chapter 4th: Software & Development

This chapter presents the implementation of the strain monitoring system, specifically focusing on the two distinct operational versions: the Serial Communication Version and the Autonomous Version. Both versions utilize the same hardware components, including the strain gauges, resistor bridge, signal conditioning unit, and the FPGA on the NEEK MAX10 Development Kit. The primary difference lies in their software configurations and methods of data handling and visualization.

The Serial Communication Version employs MATLAB for data visualization and interaction. This version facilitates data transmission from the FPGA to a computer via serial communication, where MATLAB processes and visualizes the data. This setup allows for flexible and detailed data analysis on a powerful computing platform, making it suitable for environments where direct interaction with the system is required.

In contrast, the Autonomous Version operates independently, leveraging the Nios II processor integrated within the FPGA and the Eclipse development environment for on-board data processing and display. This version processes and visualizes data directly on the NEEK MAX10's LCD screen, enabling real-time monitoring without the need for an external computer. This setup is ideal for scenarios requiring continuous monitoring in remote or inaccessible locations.

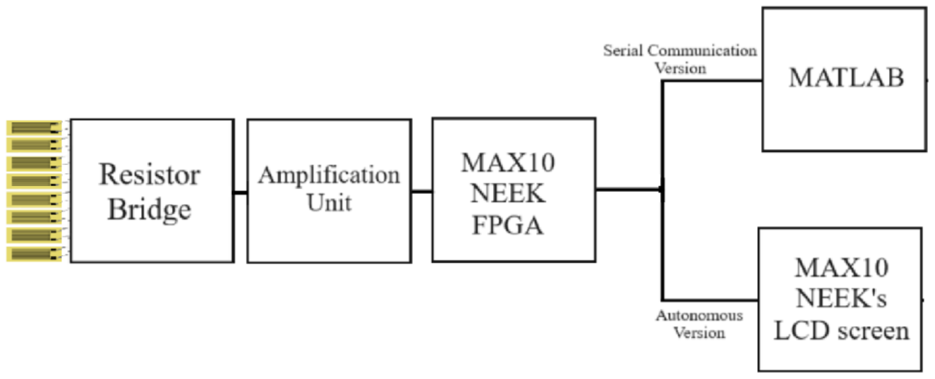


Figure 26: Block diagram of the system

Both versions are programmed using Quartus Prime Lite, which provides the necessary tools for FPGA configuration and integration with other software components. The following sections will delve into the detailed implementation of each version, starting with the Serial Communication Version, highlighting the specific configurations, software setups, and operational methodologies.

4.1 Serial Communication Version

This section details the implementation of the strain monitoring system using the Serial Communication Version. This version utilizes MATLAB for data acquisition, processing, and visualization, interfacing with the NEEK MAX10 FPGA Development Kit via serial communication. It is designed for environments that require detailed data analysis and flexible interaction with a powerful computing platform. The following sub-sections cover the setup and configuration of Quartus Prime Lite and the integration with MATLAB, providing a step-by-step guide to developing and operating the system for real-time data monitoring and analysis.

4.1.1 Quartus Prime Lite Setup

The circuit diagram depicts an advanced FPGA-based system designed to interface with an ADC (Analog-to-Digital Converter), converting analog signals to digital format for further processing and transmission.

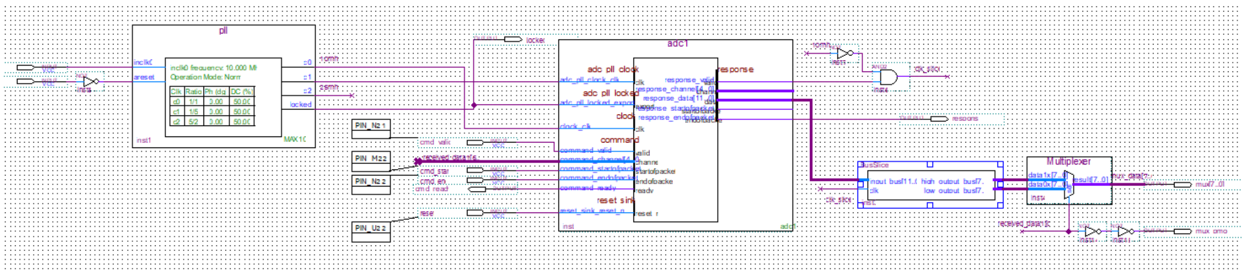


Figure 27: Block diagram of the serial communication version on Quartus (1)

The system starts with the analog input signal from the amplification unit being fed into the Modular ADC core Intel FPGA IP **[16]**. The ADC is configured to convert this analog signal into a 12-bit digital output, enabling precise representation of the analog data in digital form. The ADC operates synchronously, driven by a clock signal generated by a clock generator module, the ALTPLL **[17]** within the circuit. This clock generator ensures that the ADC and other components in the system receive consistent timing signals, which is crucial for synchronized operations and avoiding timing discrepancies.

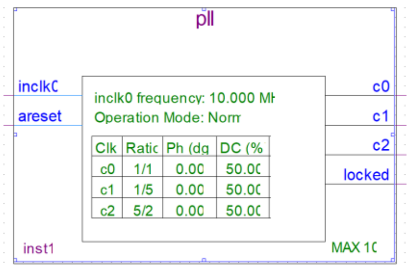


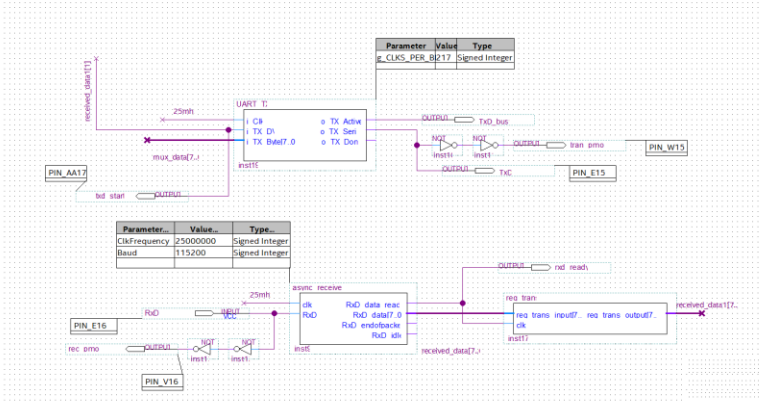
Figure 28: ALTPLL Module

Once the ADC converts the analog signal to a digital format, the 12-bit digital outputs are directed into the FPGA. Within the FPGA, there is sophisticated control logic designed to handle the synchronization of incoming data, ensuring that all bits are correctly aligned and processed. After processing, the digital signals are routed through a Bus Slicer, a crucial component for signal transmission in serial communication systems. Serial communication typically operates with 8-bit words, whereas the system's output signal is a 12-bit word. The Bus Slicer addresses this discrepancy by dividing the 12-bit signal into two separate 8-bit signals, designated as the high byte and the low byte. To ensure compatibility and proper alignment, four leading zeros are appended to the high byte, effectively padding the digital word to match the required format. This segmentation facilitates efficient and accurate data transmission through the serial communication interface **(Appendix 1)**.

The circuit incorporates a multiplexer, the LPM_MUX **[18]** to enhance signal routing efficiency. The multiplexer alternates between outputting the high byte and the low byte based on the command received from its select parameter. This mechanism allows the two signals generated by the ADC to be sequentially transmitted to the transmitter. Upon reception by the PC, these signals are recombined, effectively reconstructing the original data. This method ensures seamless serial transmission and data integrity.

For real-time monitoring and debugging, the circuit includes indicator LEDs. These visual indicators provide immediate feedback on the status of the signals and the operational state of the system. This feature is particularly useful for diagnosing issues and verifying that the system is functioning correctly.

The circuit also includes a comprehensive UART communication system, encompassing both the transmitter and receiver functionalities.



Parameter	Value	Type
g_CLKS_PER_BI	217	Signed Integer

Parameter...	Value...	Type...
ClkFrequency	25000000	Signed Integer
Baud	115200	Signed Integer

Figure 29: Block diagram of the serial communication version on Quartus (2)

The transmitter section includes the UART_TX module, which is responsible for converting parallel data into serial form for transmission. The module receives an 8-bit data input (TX_Byte), a data valid signal (TX_DV), and a clock signal (clk). It outputs a serial data stream (TX_Serial), an active transmission signal (TX_Active), and a transmission complete signal (TX_Done). A baud rate generator configured for 115200 bps with a clock frequency of 25 MHz ensures the correct timing for data transmission. The output from the UART_TX module is buffered and driven to an output pin (PIN_E15), maintaining signal integrity for reliable data transmission **(Appendix 2)**.

The receiver section comprises the async_receive module, which handles the reception of serial data and converts it back to parallel form. This module takes the serial input (RxD) and clock signal (clk) and outputs the received 8-bit data (RxD_data). Additional outputs include data ready (RxD_data_ready), indicating valid received data, end of packet (RxD_endofpacket), signifying the end of a data burst, and idle status (RxD_idle), indicating no data reception. The system ensures correct timing and synchronization for data sampling, processing the received data only when it is valid **(Appendix 3)**. Moreover, an 8-bit register that synchronizes data transfer with a clock signal is connected to the RxD_data output. The component, named reg_trans, has three ports: an 8-bit input (reg_trans_input), a clock input (clk), and an 8-bit output (reg_trans_output). On every rising edge of the clock signal, the value present at the reg_trans_input is transferred to reg_trans_output. This behavior is defined within the architecture sim, ensuring that data is only updated on the rising edge of the clock, making this component useful for timing and synchronization tasks in digital circuits **(Appendix 4)**.

The interfacing and control logic coordinate the data flow between the transmitter and receiver modules. Control signals such as TX_Active, TX_Done, RxD_data_ready, and RxD_idle provide status updates, facilitating seamless communication between different parts of the system. The design incorporates output buffers and drivers to maintain signal quality and prevent data loss or corruption during transmission.

Overall, the circuit effectively implements a robust and reliable UART communication system on an FPGA, demonstrating the capability to handle serial data transmission and reception with precise timing and control, making it suitable for various applications requiring UART-based communication. This FPGA-based design converts analog signals to digital, and outputs the conditioned digital data for further use. The inclusion of clock synchronization, signal buffering, multiplexing, and real-time monitoring makes it a robust solution for applications requiring precise and efficient handling of both analog and digital signals.

4.1.2 MATLAB Integration

This section details the integration of MATLAB for real-time data acquisition, processing, visualization, and logging using serial communication. MATLAB, known for its powerful data analysis and visualization capabilities, is used to interface with the strain monitoring system, enabling efficient handling of serial data streams from the FPGA. The MATLAB script is designed to manage multiple data channels, process the incoming data, and provide both real-time visual feedback and persistent logging for subsequent analysis. This approach provides a comprehensive solution for monitoring and analyzing strain data in real time, leveraging MATLAB’ s robust functionalities to enhance the system's performance and usability.

- **Code Explanation (Appendix 5)**

The MATLAB code is a comprehensive implementation for real-time data acquisition, processing, visualization, and logging using a serial communication link. The code starts by configuring the serial port on COM8 with a baud rate of 115200, specifying no parity, one stop bit, and eight data bits. It sets the serial port timeout to 0.1 seconds to ensure timely data reads. The number of data points to average is defined by the variable “np”, set to 5 in this case.

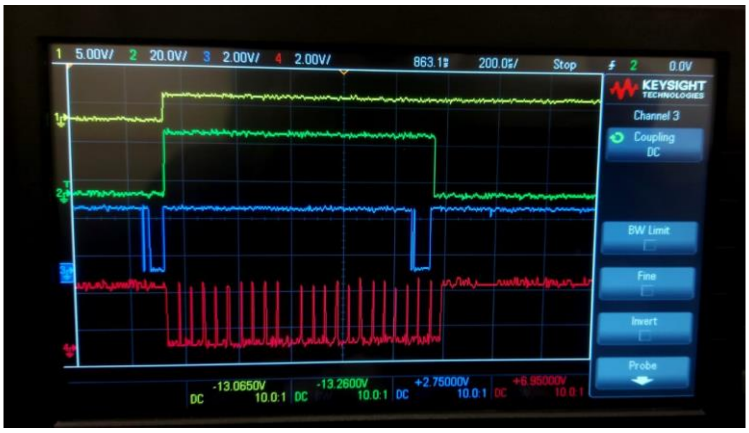


Figure 30: Serial communication signals zoomed

Buffers and counters for each of the six data channels are initialized to zero, preparing the system to store and manage incoming data efficiently. These buffers are essential for holding the incoming data points before averaging them. The figure for plotting the real-time data is set up with animated lines for each of the six channels, each assigned a distinct color for easy differentiation. The plot is configured with labels for the x-axis (Time), the y-axis (Average Data Value), and a title (Real-time Average Data Plot), and grid lines are enabled for better visualization.

The main data acquisition process occurs in an infinite while loop. Within this loop, the code sends a series of commands to the device via the serial port to request data from each channel sequentially. It uses `write` commands with specific binary values representing these requests.

After each command, the code pauses briefly to ensure the device has time to process and respond. The serial port is flushed using `flush` to clear any residual data that might cause errors during the read operation.

For each channel, the code reads the high and low bytes of the data using the `read` function, then combines these bytes into a single 16-bit value using bitwise operations. The combined data is validated to ensure it is not empty and is a v value. If valid, the data is added to the corresponding buffer, and the buffer counter is incremented. When a buffer is filled with the specified number of points (`np`), the code calculates the average of the buffered data.

To account for potential offset errors, the code initially sets a baseline (calc) for each channel based on the first average calculation. Subsequent averages are adjusted by subtracting this baseline. The adjusted averages are then plotted in real-time using the "addpoints" function, which updates the animated lines in the figure. This continuous plotting provides a dynamic visual representation of the data as it is being received and processed.

Simultaneously, the code prints the raw data (high and low bytes) and the combined 16-bit data for each channel to the console. It also converts the 16-bit data values to voltages, assuming a full-scale range of 0 to 3.3V, using the formula "(data * 3.3) / 4095". These voltage values provide a more intuitive understanding of the data in terms of real-world measurements.

In addition to real-time visualization, the code logs the processed data to a text file. Initially, it creates a new file named `VoltageData.txt` and writes a header line indicating the channels. During each iteration of the loop, it opens this file in append mode and writes the averaged voltage values for all six channels, ensuring that new data is continuously logged without overwriting previous entries. This persistent logging allows for later analysis and record-keeping.

- **Oscilloscope signals explanation:**

Channel 1 (yellow): The selector of the multiplexer

Channel 2 (green): Matlab's "Read" command

Channel 3 (blue): Matlab's "Write" command

Channel 4 (red): High/Low byte

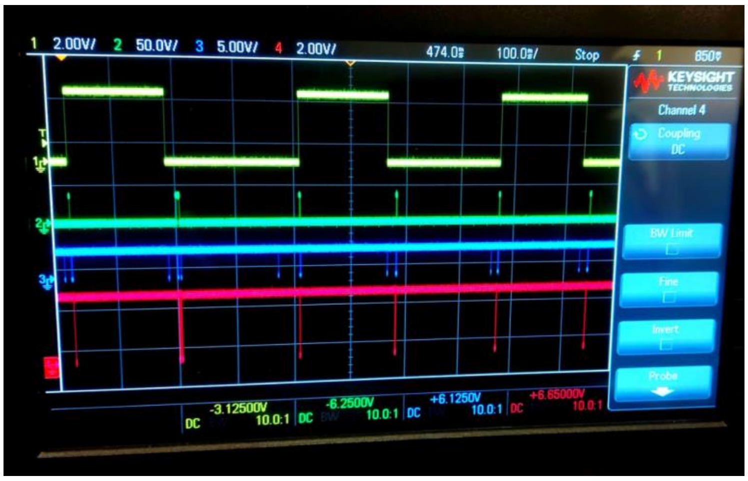


Figure 31: Serial communication signals

Overall, this MATLAB code provides a solution for real-time data acquisition, processing, and visualization from a serial device, with additional functionality for logging data to a file. It effectively handles the complexities of serial communication, data buffering, averaging, and plotting, making it a valuable tool for applications requiring continuous monitoring and analysis of multiple data channels.

4.2 Autonomous Version

This section explores the autonomous version of the strain monitoring system, designed to operate independently using the Nios II processor within the NEEK MAX10 FPGA Development Kit. The system performs on-board data processing and real-time display on the integrated LCD screen, eliminating the need for an external computer. This capability is crucial for continuous monitoring in remote or inaccessible locations. The following sub-sections detail the setup and configuration using Quartus Prime Lite and the development of the software with Eclipse, providing a comprehensive guide to implementing the autonomous version of the system.

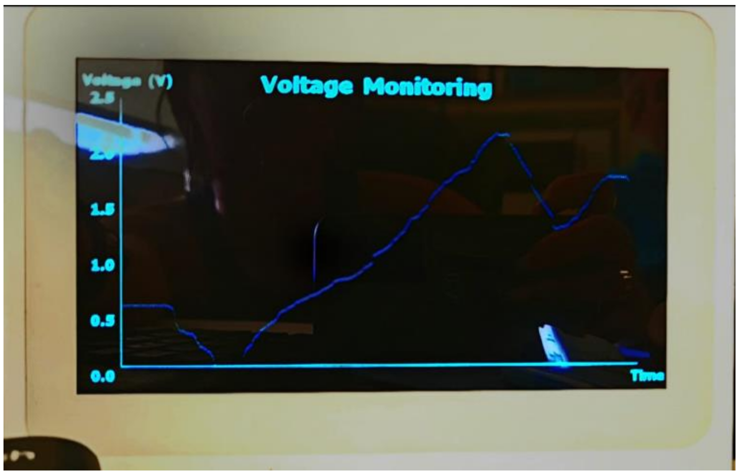
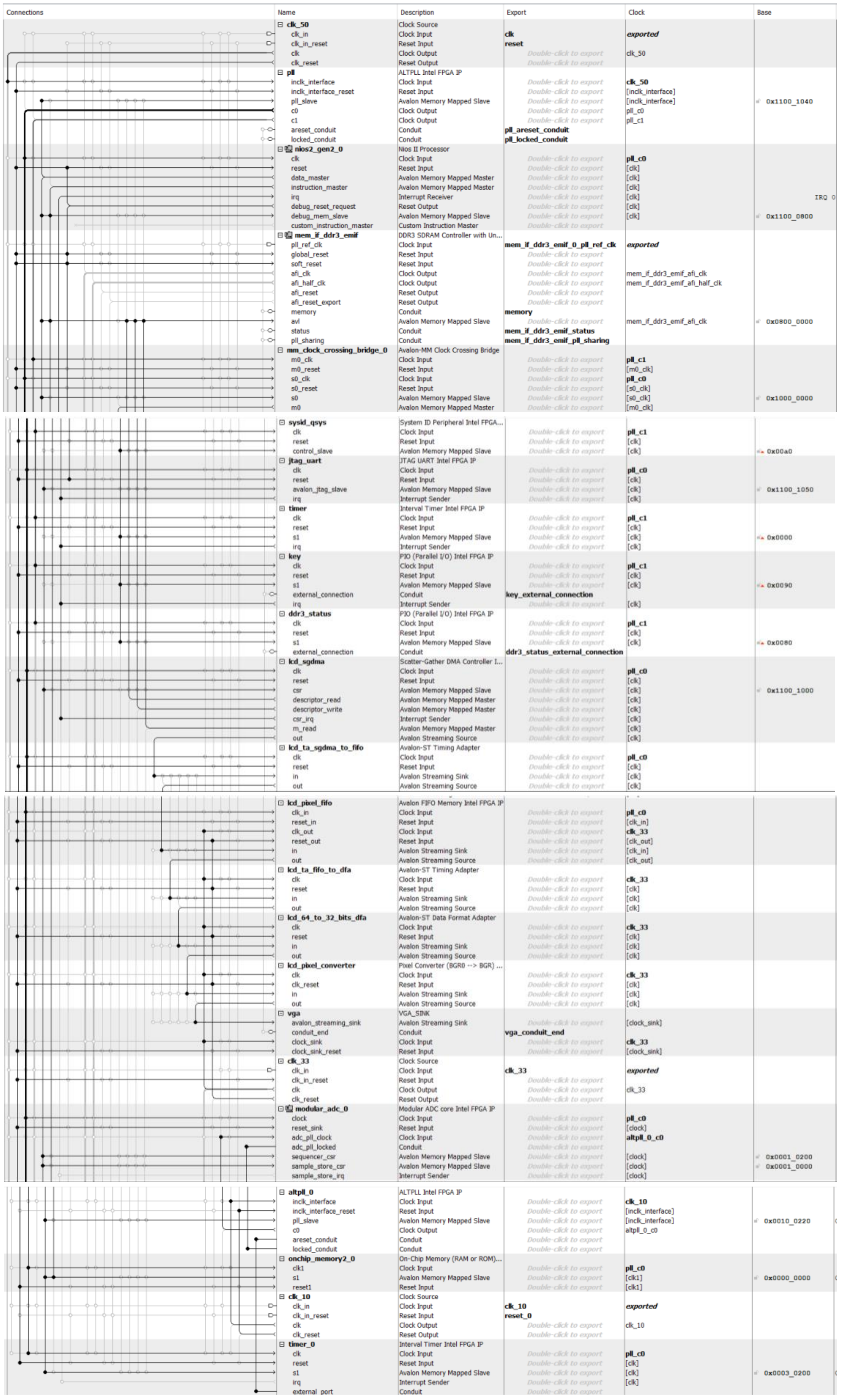


Figure 32: Autonomous Version – Visualization

4.2.1 Quartus Prime Lite and Nios II Setup

The embedded platform designed for the MAX 10 FPGA in Quartus Platform Designer integrates various components to support complex processing tasks, efficient memory management, comprehensive peripheral interfacing, and robust system communication. This system is engineered to handle applications that demand high performance and versatile interfacing options.



Connections	Name	Description	Export	Clock	Base
	⊟ **clk_50**	Clock Source			
	clk_in	Clock Input	**clk**	***exported***	
	clk_in_reset	Reset Input	**reset**		
	clk	Clock Output	*Double-click to export*	clk_50	
	clk_reset	Reset Output	*Double-click to export*		
	⊟ **pll**	ALTPLL Intel FPGA IP			
	inclk_interface	Clock Input	*Double-click to export*	**clk_50**	
	inclk_interface_reset	Reset Input	*Double-click to export*	[inclk_interface]	
	pll_slave	Avalon Memory Mapped Slave	*Double-click to export*	[inclk_interface]	0x1100_1040
	c0	Clock Output	*Double-click to export*	pll_c0	
	c1	Clock Output	*Double-click to export*	pll_c1	
	areset_conduit	Conduit	**pll_areset_conduit**		
	locked_conduit	Conduit	**pll_locked_conduit**		
	⊟ **nios2_gen2_0**	Nios II Processor			
	clk	Clock Input	*Double-click to export*	**pll_c0**	
	reset	Reset Input	*Double-click to export*	[clk]	
	data_master	Avalon Memory Mapped Master	*Double-click to export*	[clk]	
	instruction_master	Avalon Memory Mapped Master	*Double-click to export*	[clk]	
	irq	Interrupt Receiver	*Double-click to export*	[clk]	IRQ 0
	debug_reset_request	Reset Output	*Double-click to export*	[clk]	
	debug_mem_slave	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	0x1100_0800
	custom_instruction_master	Custom Instruction Master	*Double-click to export*		
	⊟ **mem_if_ddr3_emif**	DDR3 SDRAM Controller with Un...			
	pll_ref_clk	Clock Input	**mem_if_ddr3_emif_0_pll_ref_clk**	***exported***	
	global_reset	Reset Input	*Double-click to export*		
	soft_reset	Reset Input	*Double-click to export*		
	afi_clk	Clock Output	*Double-click to export*	mem_if_ddr3_emif_afi_clk	
	afi_half_clk	Clock Output	*Double-click to export*	mem_if_ddr3_emif_afi_half_clk	
	afi_reset	Reset Output	*Double-click to export*		
	afi_reset_export	Reset Output	*Double-click to export*		
	memory	Conduit	**memory**		
	avl	Avalon Memory Mapped Slave	*Double-click to export*	mem_if_ddr3_emif_afi_clk	0x0800_0000
	status	Conduit	**mem_if_ddr3_emif_status**		
	pll_sharing	Conduit	**mem_if_ddr3_emif_pll_sharing**		
	⊟ **mm_clock_crossing_bridge_0**	Avalon-MM Clock Crossing Bridge			
	m0_clk	Clock Input	*Double-click to export*	**pll_c1**	
	m0_reset	Reset Input	*Double-click to export*	[m0_clk]	
	s0_clk	Clock Input	*Double-click to export*	**pll_c0**	
	s0_reset	Reset Input	*Double-click to export*	[s0_clk]	
	s0	Avalon Memory Mapped Slave	*Double-click to export*	[s0_clk]	0x1000_0000
	m0	Avalon Memory Mapped Master	*Double-click to export*	[m0_clk]	
	⊟ **sysid_qsys**	System ID Peripheral Intel FPGA...			
	clk	Clock Input	*Double-click to export*	**pll_c1**	
	reset	Reset Input	*Double-click to export*	[clk]	
	control_slave	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	0x00a0
	⊟ **jtag_uart**	JTAG UART Intel FPGA IP			
	clk	Clock Input	*Double-click to export*	**pll_c0**	
	reset	Reset Input	*Double-click to export*	[clk]	
	avalon_jtag_slave	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	0x1100_1050
	irq	Interrupt Sender	*Double-click to export*	[clk]	
	⊟ **timer**	Interval Timer Intel FPGA IP			
	clk	Clock Input	*Double-click to export*	**pll_c1**	
	reset	Reset Input	*Double-click to export*	[clk]	
	s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	0x0000
	irq	Interrupt Sender	*Double-click to export*	[clk]	
	⊟ **key**	PIO (Parallel I/O) Intel FPGA IP			
	clk	Clock Input	*Double-click to export*	**pll_c1**	
	reset	Reset Input	*Double-click to export*	[clk]	
	s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	0x0090
	external_connection	Conduit	**key_external_connection**		
	irq	Interrupt Sender	*Double-click to export*	[clk]	
	⊟ **ddr3_status**	PIO (Parallel I/O) Intel FPGA IP			
	clk	Clock Input	*Double-click to export*	**pll_c1**	
	reset	Reset Input	*Double-click to export*	[clk]	
	s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	0x0080
	external_connection	Conduit	**ddr3_status_external_connection**		
	⊟ **lcd_sgdma**	Scatter-Gather DMA Controller I...			
	clk	Clock Input	*Double-click to export*	**pll_c0**	
	reset	Reset Input	*Double-click to export*	[clk]	
	csr	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	0x1100_1000
	descriptor_read	Avalon Memory Mapped Master	*Double-click to export*	[clk]	
	descriptor_write	Avalon Memory Mapped Master	*Double-click to export*	[clk]	
	csr_irq	Interrupt Sender	*Double-click to export*	[clk]	
	m_read	Avalon Memory Mapped Master	*Double-click to export*	[clk]	
	out	Avalon Streaming Source	*Double-click to export*	[clk]	
	⊟ **lcd_ta_sgdma_to_fifo**	Avalon-ST Timing Adapter			
	clk	Clock Input	*Double-click to export*	**pll_c0**	
	reset	Reset Input	*Double-click to export*	[clk]	
	in	Avalon Streaming Sink	*Double-click to export*	[clk]	
	out	Avalon Streaming Source	*Double-click to export*	[clk]	
	⊟ **lcd_pixel_fifo**	Avalon FIFO Memory Intel FPGA IP			
	clk_in	Clock Input	*Double-click to export*	**pll_c0**	
	reset_in	Reset Input	*Double-click to export*	[clk_in]	
	clk_out	Clock Input	*Double-click to export*	**clk_33**	
	reset_out	Reset Input	*Double-click to export*	[clk_out]	
	in	Avalon Streaming Sink	*Double-click to export*	[clk_in]	
	out	Avalon Streaming Source	*Double-click to export*	[clk_out]	
	⊟ **lcd_ta_fifo_to_dfa**	Avalon-ST Timing Adapter			
	clk	Clock Input	*Double-click to export*	**clk_33**	
	reset	Reset Input	*Double-click to export*	[clk]	
	in	Avalon Streaming Sink	*Double-click to export*	[clk]	
	out	Avalon Streaming Source	*Double-click to export*	[clk]	
	⊟ **lcd_64_to_32_bits_dfa**	Avalon-ST Data Format Adapter			
	clk	Clock Input	*Double-click to export*	**clk_33**	
	reset	Reset Input	*Double-click to export*	[clk]	
	in	Avalon Streaming Sink	*Double-click to export*	[clk]	
	out	Avalon Streaming Source	*Double-click to export*	[clk]	
	⊟ **lcd_pixel_converter**	Pixel Converter (BGR0 --> BGR) ...			
	clk	Clock Input	*Double-click to export*	**clk_33**	
	clk_reset	Reset Input	*Double-click to export*	[clk]	
	in	Avalon Streaming Sink	*Double-click to export*	[clk]	
	out	Avalon Streaming Source	*Double-click to export*	[clk]	
	⊟ **vga**	VGA_SINK			
	avalon_streaming_sink	Avalon Streaming Sink	*Double-click to export*	[clock_sink]	
	conduit_end	Conduit	**vga_conduit_end**		
	clock_sink	Clock Input	*Double-click to export*	**clk_33**	
	clock_sink_reset	Reset Input	*Double-click to export*	[clock_sink]	
	⊟ **clk_33**	Clock Source			
	clk_in	Clock Input	**clk_33**	***exported***	
	clk_in_reset	Reset Input	*Double-click to export*		
	clk	Clock Output	*Double-click to export*	clk_33	
	clk_reset	Reset Output	*Double-click to export*		
	⊟ **modular_adc_0**	Modular ADC core Intel FPGA IP			
	clock	Clock Input	*Double-click to export*	**pll_c0**	
	reset_sink	Reset Input	*Double-click to export*	[clock]	
	adc_pll_clock	Clock Input	*Double-click to export*	**altpll_0_c0**	
	adc_pll_locked	Conduit	*Double-click to export*		
	sequencer_csr	Avalon Memory Mapped Slave	*Double-click to export*	[clock]	0x0001_0200
	sample_store_csr	Avalon Memory Mapped Slave	*Double-click to export*	[clock]	0x0001_0000
	sample_store_irq	Interrupt Sender	*Double-click to export*	[clock]	
	⊟ **altpll_0**	ALTPLL Intel FPGA IP			
	inclk_interface	Clock Input	*Double-click to export*	**clk_10**	
	inclk_interface_reset	Reset Input	*Double-click to export*	[inclk_interface]	
	pll_slave	Avalon Memory Mapped Slave	*Double-click to export*	[inclk_interface]	0x0010_0220
	c0	Clock Output	*Double-click to export*	altpll_0_c0	
	areset_conduit	Conduit	*Double-click to export*		
	locked_conduit	Conduit	*Double-click to export*		
	⊟ **onchip_memory2_0**	On-Chip Memory (RAM or ROM)...			
	clk1	Clock Input	*Double-click to export*	**pll_c0**	
	s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk1]	0x0000_0000
	reset1	Reset Input	*Double-click to export*	[clk1]	
	⊟ **clk_10**	Clock Source			
	clk_in	Clock Input	**clk_10**	***exported***	
	clk_in_reset	Reset Input	**reset_0**		
	clk	Clock Output	*Double-click to export*	clk_10	
	clk_reset	Reset Output	*Double-click to export*		
	⊟ **timer_0**	Interval Timer Intel FPGA IP			
	clk	Clock Input	*Double-click to export*	**pll_c0**	
	reset	Reset Input	*Double-click to export*	[clk]	
	s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	0x0003_0200
	irq	Interrupt Sender	*Double-click to export*	[clk]	
	external_port	Conduit	*Double-click to export*		

Figure 33: NIOS II Platform Design

The system's primary clock source is a 50 MHz signal (`clk_50`). This clock signal is processed by the `pll` component, generating several necessary clock signals (`pll_c0`, `pll_c1`) that drive various system components. Additionally, an ALTPLL component (`altpll_0`) processes a `clk_10` signal for specific clocking requirements. Another essential clock signal, `clk_33`, is integrated for additional system timing needs. These clock signals ensure that different parts of the system operate at their optimal frequencies, enhancing overall performance. Coordinated and reliable reset operations across the system are managed through `clk_in_reset` and `clk_reset` signals, maintaining system stability and preventing undesired behaviors during startup and operation.

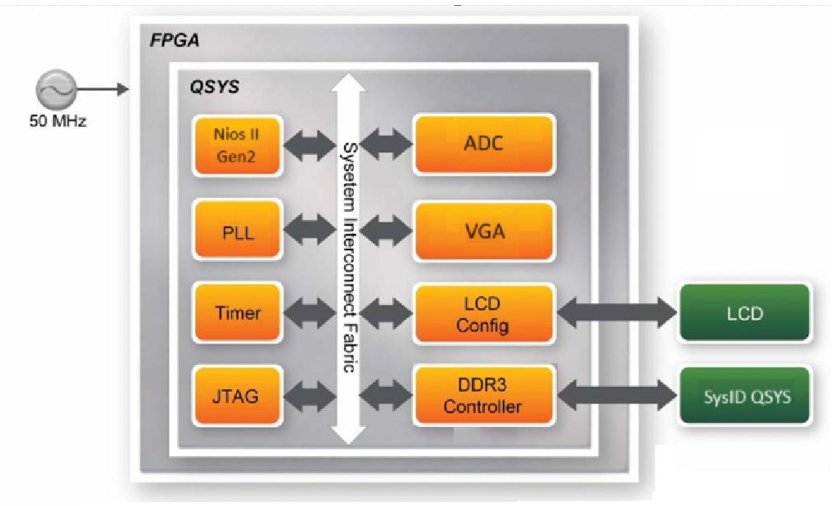


Figure 34: Block Diagram of qsys

At the heart of the system is the Nios II Gen 2 processor (`nios2_gen2_0`), renowned for its flexibility and efficiency in embedded applications. This processor communicates with other system components via Avalon Memory Mapped Master interfaces, facilitating seamless data exchange and control. The processor is responsible for executing the primary computational tasks and managing peripheral interactions, making it a central component in a wide range of embedded applications, from industrial control to multimedia processing. The memory subsystem includes both on-chip and off-chip memory components to cater to different storage needs. The on-chip memory (`onchip_memory2_0`) provides rapid, local storage that is directly accessible by the processor, ideal for storing frequently accessed data and instructions. The DDR3 memory interface (`mem_if_ddr3_emif`) allows access to external DDR3 memory, offering substantial high-speed storage capacity essential for data-intensive applications. This interface ensures the system can efficiently handle large volumes of data, which is crucial for applications requiring extensive data logging or real-time data processing.

The `mm_clock_crossing_bridge_0` component is crucial for managing data transfer between different clock domains within the system. By ensuring safe and reliable data transfer between components operating at varying clock speeds, this bridge prevents timing issues that could lead to data corruption or system instability, thus maintaining the integrity and reliability of data transactions across the system. The system is equipped with a variety of peripherals to enhance its functionality. The system ID (`sysid_qsys`) provides a unique identifier for the system, aiding in version control and system identification, which is vital during development and deployment. The JTAG UART (`jtag_uart`) enables communication over the JTAG interface, which is essential for debugging and interacting with development tools. Two timers (`timer` and `timer_0`) implement interval timing for generating periodic interrupts, which are crucial for real-time applications where precise timing is necessary for task scheduling and execution. The key interface (`key`) manages input from external keys, allowing for user interaction and control. The DDR3 status (`ddr3_status`) monitors the operational status of the DDR3 memory, providing essential diagnostic information to ensure reliable memory operation and facilitate troubleshooting.

The platform includes several components specifically designed to manage the LCD screen of the board. The LCD Scatter-Gather DMA (`lcd_sgdma`) facilitates efficient data transfer for the LCD display, reducing the load on the processor by handling data movement autonomously. The LCD SGDMA to FIFO (`lcd_sgdma_to_fifo`) converts data streams into a FIFO format, facilitating buffered data handling for the LCD display. The LCD FIFO to DFA (`lcd_fifo_to_dfa`) adapts data formats for compatibility with the LCD display requirements. The LCD 64 to 32 Bits (`lcd_64_to_32_bits`) converts data widths to ensure compatibility between components with different data bus sizes, crucial for interfacing with the LCD display. The LCD Pixel Converter (`lcd_pixel_converter`) converts pixel data formats, essential for correct image rendering on the LCD display. The LCD Pixel FIFO (`lcd_pixel_fifo`) buffers pixel data to ensure smooth and continuous data flow to the LCD display.

The system supports video output and analog signal processing through a VGA controller (`vga`) and the modular ADC (`modular_adc_0`). The VGA controller drives a VGA display, enabling graphical output for applications requiring visual feedback, such as user interfaces or data visualization. The modular ADC converts analog signals to digital format, allowing the system to interface with analog sensors and inputs, which is critical for applications involving real-world data acquisition. To facilitate interaction with external hardware and enhance debugging capabilities, the system includes several external connections and conduits. The key external connection (`key_external_connection`) manages inputs from external keys, expanding the user interface options and allowing for more interactive control mechanisms. The DDR3 status external connection (`ddr3_status_external_connection`) provides external access to DDR3 memory status information, aiding in diagnostics and system monitoring, which is essential for maintaining system health and performance.

In summary, the embedded platform for the MAX 10 FPGA, designed with Quartus Platform Designer, integrates critical components for high-performance processing, efficient memory management, and robust peripheral interfacing. The Nios II Gen 2 processor handles computational tasks and peripheral interactions, supported by on-chip and DDR3 memory for rapid data access and substantial storage. Clock management ensures optimal performance across system components. Peripheral components, including the JTAG UART, timers, and key interface, enhance system functionality and user control. The system effectively manages graphical output through components like the LCD Scatter-Gather DMA and LCD Pixel FIFO, and supports analog signal processing with the VGA controller and modular ADC.

4.2.2 Eclipse Setup and Operation

In this section, we delve into the setup and operation of Eclipse for developing the software necessary to run the autonomous version of the strain monitoring system on the NEEK MAX10 FPGA. Eclipse, a versatile and powerful Integrated Development Environment (IDE), is employed alongside the Nios II to facilitate efficient software development for embedded systems.

The Nios II processor, integrated into the FPGA, requires a well-structured software environment to execute tasks such as data acquisition from the analog-to-digital converter (ADC), data processing, and graphical representation on the LCD display. Eclipse, with its extensive plugin ecosystem and robust debugging capabilities, provides the ideal platform for developing and managing these complex tasks.

The autonomous system is designed to operate independently, processing data on-board and displaying the results in real-time. This chapter outlines the steps to set up Eclipse for this purpose, configure the Nios II processor, and develop the C program that drives the system's functionality. By leveraging Eclipse and the Nios II EDS, the development process is streamlined, ensuring efficient code management and deployment.

- **Code Explanation (Appendix 6)**

The main.c code of this project is programmed to interface with an analog-to-digital converter (ADC) and an LCD display, with the primary function of reading data from eight ADC channels and graphically representing this data on the screen. The program begins by setting up necessary system and hardware-specific libraries, ensuring that it can access input/output operations, system calls, video display capabilities, simple graphics, fonts, and ADC functionality. It defines constants for color codes, ADC base addresses, screen dimensions, and the maximum ADC value, which facilitates clear and manageable code.

The core of the program involves initializing the LCD display, sampling ADC values, and rendering these values as distinct colored graphs. The `init_lcd` function sets up the LCD display, configuring parameters like device name, screen resolution, color depth, and buffer location. It also clears the screen to a white background to prepare it for graphical output. The `read_adc` function is pivotal as it reads values from all eight ADC channels, prints these values for debugging purposes, and stores them in an array.

Graphical functions such as `plot_point` and `draw_graph` are used to plot points and draw lines on the display. The `plot_point` function plots individual points, while `draw_graph` constructs a graph by connecting these points with lines. Each channel is assigned a unique color, enhancing visual differentiation. The `draw_axes` function adds context by drawing and labeling the x and y axes on the display.

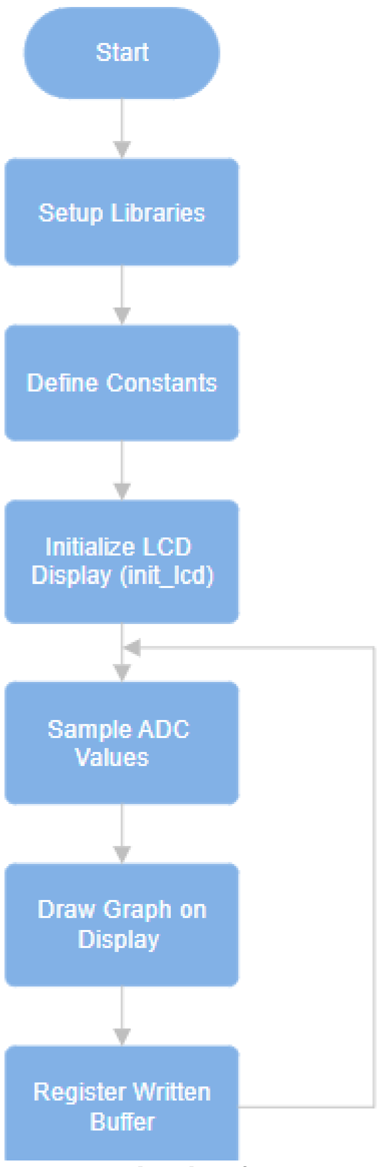


Figure 35: Flowchart for main.c

In the main function, the program orchestrates the overall execution by initializing the LCD display, sampling ADC values with a delay to simulate a realistic sampling rate, and storing these values in a structured array. The display is then cleared, and the axes and graph are drawn based on the sampled data. Finally, the program enters a continuous loop to allow for potential real-time updates, maintaining flexibility for further development or integration.

In summary, this program effectively transforms raw ADC data into a visual representation on an LCD screen, providing a clear and real-time graphical display of multiple ADC channels. It is a valuable tool for monitoring and analyzing analog signals, making complex data easily interpretable through distinct visual graphs.

5 Chapter 5th : Implementation of the Monitoring System and Results

5.1 Implementation of the System

The implementation of the strain monitoring system integrates essential hardware and software components to achieve reliable, real-time data acquisition and analysis. In the hardware setup, KYOWA KFG-5-120-C1-11L3M2R strain gauges are strategically positioned on the test specimens to capture strain data from various regions. The configuration comprises four sensors along the central axis and two outside the central region, allowing for comprehensive strain analysis across the specimen. These strain gauges are connected in the designed resistor bridge. This setup, as pictured in the following block diagram is crucial for converting the small changes in resistance of the strain gauges into measurable voltage changes, which can then be amplified and processed.

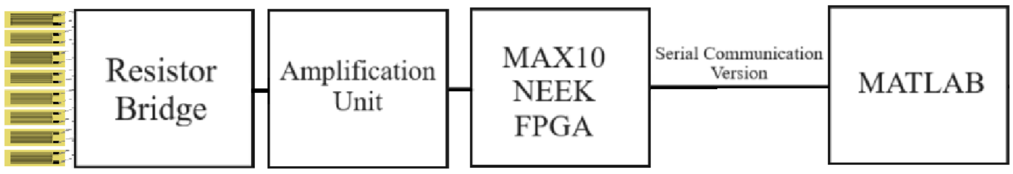


Figure 36: Block Diagram of the Serial Communication Version

The specimen was placed centrally between the compression plates and secured to prevent lateral movement and three experiments were conducted. Firstly, a gradually increasing compressive load was applied to the specimen. Secondly, an initial preload was to the specimen, which was gradually decreased from the preload value, and the specimen's behavior was monitored. Finally, the specimen was repositioned for a 3-point bending test setup. It was placed on two supports, and a load was applied around the midpoint and the load was gradually increased.

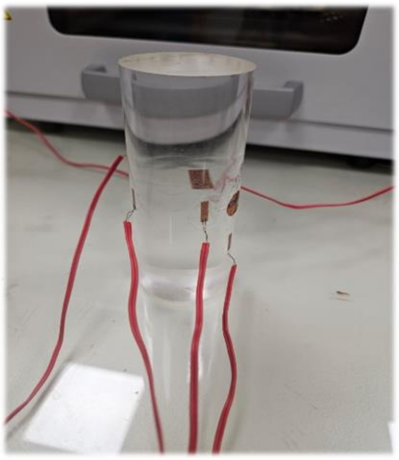


Figure 37: Testing Speciment

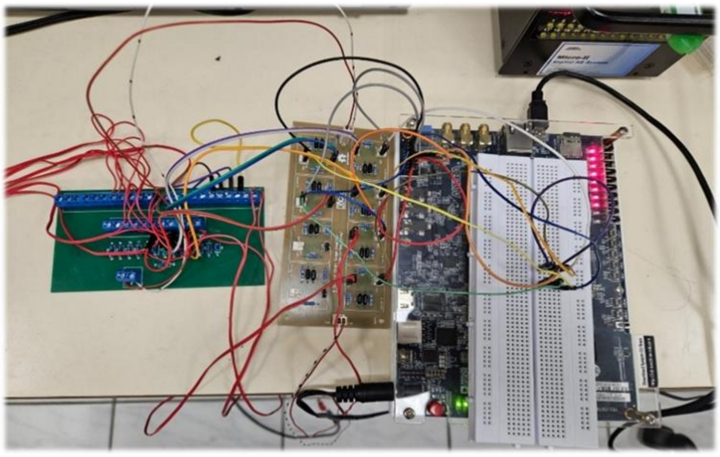


Figure 38: System's Setup

Throughout the tests, data acquisition systems continuously monitored and recorded the load and deformation. The recorded data were saved for further analysis, to determine the mechanical properties of the plexiglass specimen, such as compressive strength, flexural strength, and modulus of elasticity. Multiple tests were conducted to ensure the reliability and repeatability of the results.

5.2 Experiment 1: Strain Measurement under Compressive Load

Experimental Setup

The first experiment involved measuring the strain on a plexiglass test sample subjected to compression up to 60kN in successive loading cycles. This setup aimed to observe the elastic deformation behavior of the material within this strain range. The plexiglass specimen was equipped with multiple strain sensors, strategically placed to capture the deformation across different regions of the sample.

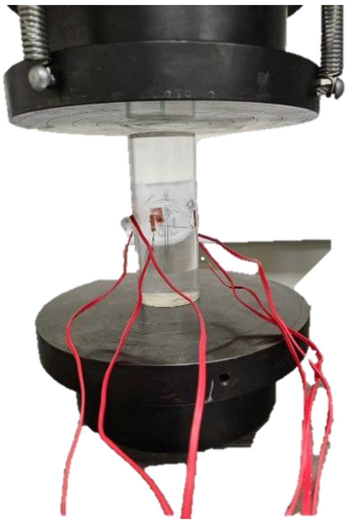


Figure 39: Experiment 1 Setup

The sensors were arranged with four positioned along the central axis and two outside the central region. This configuration allowed for a comparative analysis of strain distribution, particularly focusing on the effects of barrel deformation caused by friction at the contact surfaces. Notably, no lubrication was used to remove friction effects in order to enhance the visibility of this phenomenon.

Procedure

The experiment began with the initial setup, where the plexiglass sample was placed in a compression loading frame, and the strain sensors were calibrated and connected to the data acquisition system. Following this, the sample underwent incremental loading cycles, gradually increasing the force up to 60kN. During each loading cycle, strain data from all sensors were recorded in real-time. After reaching the maximum load, the sample was gradually unloaded to observe the strain recovery. This loading and unloading process was repeated multiple times to ensure the consistency and reliability of the collected data.

Results and Analysis

The results showed that the plexiglass material exhibited elastic deformation within the applied strain range. The sensors located along the central axis of the specimen recorded larger strains compared to those positioned outside the central region. This discrepancy is attributed to the barrel deformation effect, where friction at the contact surfaces (due to the lack of lubrication) restricted the deformation in the outer regions.

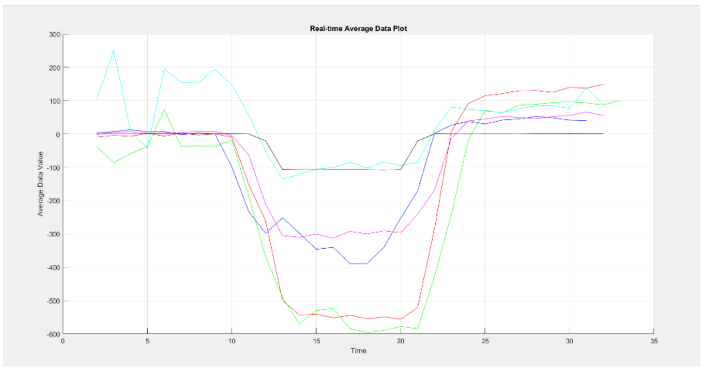


Figure 40: Strain Measurement under Compressive Load Diagram

The graph presented illustrates the real-time average data values over the duration of the experiment. The data highlights the following key observations:

- Central Axis Sensors: These sensors showed significant strain values, indicating higher deformation in the central region.
- Outer Region Sensors: The sensors outside the central axis recorded lower strain values, confirming the impact of friction-induced barrel deformation.

This insight is crucial for understanding the material behavior under compression and can inform improvements in experimental setups to mitigate such effects in future tests.

Overall, this experiment successfully demonstrated the strain monitoring capabilities of the developed system and provided valuable data on the elastic deformation characteristics of plexiglass under compressive load. The findings underscore the importance of sensor placement and friction management in accurate strain measurement.

5.3 Experiment 2: Strain Measurement with Preload

Experimental Setup

In this experiment, the objective was to study the strain behavior of the plexiglass test sample under compression with a preload of 30kN. The aim was to observe the response of the strain gauges during both the reduction and the increase in the length of the sample. The test setup included four strain sensors along the central axis of the specimen and two sensors outside the central region, providing a comprehensive analysis of strain distribution.

Procedure

The procedure for this experiment involved placing the plexiglass sample in the compression testing machine and calibrating the strain sensors, which were connected to the data acquisition system. A preload of 30kN was applied to establish a baseline for the measurements. Following this, the sample's length was incrementally reduced and increased to observe the strain behavior. During these length adjustments, real-time strain data from all sensors were recorded. This process of adjusting the length and recording data was repeated multiple times to ensure the consistency and reliability of the collected data.

Results and Analysis

The results indicated that the strain measurements were consistent with the expected behavior under the applied conditions. During compression, the strain gauges recorded negative values, indicating a reduction in length. During relaxation, the gauges showed positive values, indicating an increase in length. This pattern was observed consistently across all sensors, confirming the material's elastic deformation response.

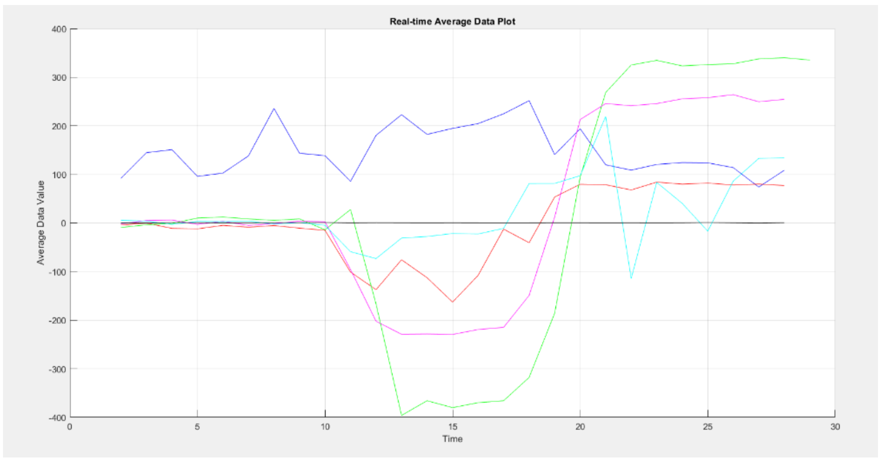


Figure 41: Strain Measurement with Preload

However, it is important to note that the sensor visualized with the dark blue color experienced damage and was smashed during the procedure. As a result, the data from this specific sensor is not accurate and should be interpreted with caution.

The strain values recorded by the sensors along the central axis were more significant compared to those at the edges, reflecting greater deformation in the central region. The edge sensors exhibited relatively smaller strain values, consistent with reduced deformation due to the preload and subsequent length adjustments.

The graph presented illustrates the real-time average data values over the duration of the experiment. The data highlights the following key observations:

- Central Axis Sensors: These sensors showed significant strain values, indicating higher deformation in the central region.
- Outer Region Sensors: The sensors outside the central axis recorded lower strain values, confirming the impact of friction-induced barrel deformation.
- Dark Blue Sensor: The anomalous data from the dark blue sensor due to damage should be disregarded in the final analysis.

This experiment validated the performance of the strain monitoring system under conditions of preload and controlled length variation, showcasing its ability to provide reliable and detailed strain measurements critical for understanding material behavior under such loading scenarios.

5.4 Experiment 3: Strain Measurement under 3-Point Bending

Experimental Setup

The third experiment involved measuring the strain behavior of the plexiglass specimen under 3-point bending using the Instron restraint system. This test aimed to observe the material's response in both the tensile and compressive zones, as well as the neutral region. The setup included strain sensors placed on the lower side (tensile zone), upper side (compressive zone), and the neutral region at one-third the height of the specimen.

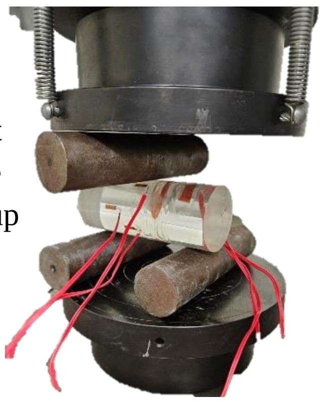


Figure 42: Experiment 3 Setup

Procedure

The procedure for this experiment involved placing the plexiglass sample on the 3-point bending fixture of the Instron machine. The strain sensors were calibrated and connected to the data acquisition system. The sample was subjected to bending by applying a load at the central point while the ends were supported. Real-time strain data from all sensors were recorded during the loading process using MATLAB for continuous monitoring and data logging. The behavior of the strain gauges was observed in the tensile zone, compressive zone, and neutral region. This bending process was repeated to ensure the consistency and reliability of the collected data.

Results and Analysis

The results from the 3-point bending experiment provided clear insights into the strain distribution across the specimen. In the tensile zone (lower side of the specimen), the strain gauges recorded positive values, indicating an increase in length due to tensile loading. Conversely, in the compressive zone (upper side of the specimen), the strain gauges showed negative values, indicating a decrease in length due to compressive loading. Notably, the strain gauges located in the neutral region at one-third the height of the specimen recorded near-zero deflections, as expected, indicating minimal strain in this area.

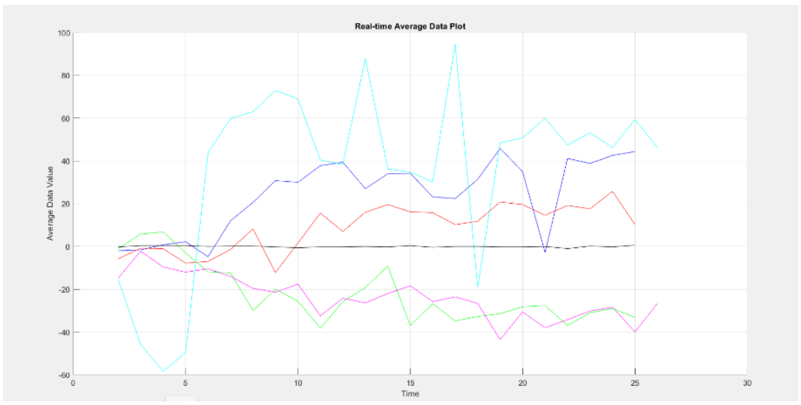


Figure 43: Strain Measurement under 3-Point Bending

The graph presented illustrates the real-time average data values over the duration of the experiment. The data highlights the following key observations:

- Tensile Zone Sensors: These sensors recorded positive strain values, reflecting the elongation of the specimen in the tensile zone.
- Compressive Zone Sensors: These sensors recorded negative strain values, indicating compression in the upper side of the specimen.
- Neutral Region Sensors: The sensors in the neutral region showed near-zero strain values, confirming minimal deformation in this area.

This experiment successfully demonstrated the different strain behaviors in the tensile and compressive zones and the neutral region under 3-point bending. The results validate the system's capability to accurately capture and analyze strain data, providing a detailed understanding of the material's response to bending loads.

Overall, the findings from the 3-point bending experiment underscore the effectiveness of the strain monitoring system in diverse loading scenarios, highlighting its applicability for detailed material behavior analysis in both tensile and compressive zones.

5.5 Comparison of Results

The comparison of results from the three experiments provided significant insights into the material's behavior under various loading conditions. The compressive load and preload experiments revealed consistent strain distribution and magnitude patterns, demonstrating the material's elastic deformation behavior. In the compressive load experiment, the strain sensors along the central axis recorded larger strain values compared to those outside the central region, indicating significant deformation in the central part due to the applied load. This consistency was mirrored in the preload experiment, where the central axis sensors also showed greater strain values during both length reduction and increase, confirming the material's uniform response to different loading conditions.

In terms of load response, the compressive load experiment showed that strain values remained consistent across successive loading cycles, highlighting the material's ability to elastically recover after each cycle. Similarly, in the preload experiment, the strain gauges recorded negative values during compression and positive values during relaxation, consistent with the material's elastic properties and its capacity to return to its original shape after unloading. Additionally, both experiments indicated that the strain values at the edge sensors were relatively smaller, underscoring the impact of friction at the contact surfaces and demonstrating the strain monitoring system's reliability across different loading scenarios.

The 3-point bending experiment further validated the system's precision in monitoring strain across different regions of the specimen. The strain gauges in the tensile zone recorded positive strain values, indicating elongation, while those in the compressive zone showed negative values, indicating compression. This clear differentiation between the tensile and compressive zones was essential for understanding the material's behavior under bending loads. Moreover, the sensors in the neutral region recorded near-zero strain values, confirming minimal deformation in this area and aligning with theoretical expectations.

6 Chapter 6th : Conclusion and Future Improvements

6.1 Conclusion

This thesis has detailed the comprehensive development and implementation of an advanced strain monitoring system utilizing FPGA technology. The system's capability to provide real-time data acquisition, processing, and visualization has been demonstrated through various experiments, highlighting its accuracy and reliability in measuring strain under different loading conditions.

The Serial Communication Version of the system, utilizing MATLAB for data visualization and interaction, has proven effective in environments where detailed data analysis and direct interaction with a powerful computing platform are required. Conversely, the Autonomous Version, leveraging the Nios II processor within the FPGA, has shown its strength in providing continuous monitoring without the need for an external computer, making it suitable for remote or inaccessible locations.

Through the experiments conducted, the strain monitoring system has successfully captured and analyzed strain data, providing valuable insights into the material behavior under compressive load, preload, and 3-point bending scenarios. The system's robustness and reliability in real-time monitoring applications underscore its potential for broader applications in structural health monitoring and material analysis.

Finally, the development of this strain monitoring system marks a significant advancement in the field of structural health monitoring. Its ability to provide accurate, real-time data and its potential for further improvements ensure its relevance and applicability in a wide range of engineering and industrial applications. The insights gained from this work pave the way for future innovations, contributing to the ongoing advancement of monitoring technologies and the enhancement of structural safety and reliability.

6.2 Future Improvements

The development and implementation of the strain monitoring system as described in this thesis have shown significant promise in providing accurate and real-time data for various applications. However, there are several areas where future improvements could enhance the system's performance, usability, and applicability.

Incorporating more advanced and diverse types of sensors could provide a more comprehensive understanding of material behavior under different conditions. Future versions could integrate temperature sensors to account for thermal effects on strain measurements. Implementing wireless communication technologies could enhance the system's flexibility and ease of deployment, particularly in remote or difficult-to-access locations. This improvement would eliminate the need for physical connections and enable real-time data transmission over long distances. Developing more sophisticated data analysis algorithms, potentially incorporating machine learning techniques, could automate the identification of patterns and anomalies in the strain data. This would improve the system's ability to predict failures and assess structural health more accurately. Reducing the size of the hardware components and developing portable versions of the system would make it more practical for field applications. Portable systems could be used in dynamic environments where mobility is essential.

Improving the power efficiency of the system, possibly through the use of low-power components or energy harvesting technologies, could extend the operational life of the system, particularly in remote monitoring scenarios. Developing a more user-friendly interface, possibly with touchscreen capabilities or mobile app integration, could improve the accessibility and usability of the system for various users, including those without extensive technical expertise. Incorporating real-time feedback mechanisms such as alarms or notifications when certain strain thresholds are exceeded could provide immediate alerts to users, enhancing the system's utility in critical monitoring applications. Expanding the system's data storage capabilities and integrating with cloud platforms could facilitate long-term data analysis and accessibility from multiple devices. This would support more comprehensive monitoring and historical data review. Additionally, in the autonomous version, the data are not yet saved on the SD card of the board, so future improvements should include implementing this feature to ensure data preservation and further analysis.

Overall, these enhancements would significantly increase the functionality, reliability, and user experience of the strain monitoring system, making it a more powerful tool for structural health monitoring and material analysis.

Bibliography - References - Internet Sources

[1] Wirtz, S.F., Cunha, A.P.A., Labusch, M., Marzun, G., Barcikowski, S. and Söffker, D., 2018. Development of a low-cost FPGA-based measurement system for real-time processing of acoustic emission data: proof of concept using control of pulsed laser ablation in liquids. Sensors, 18(6), p.1775
https://doi.org/10.3390/s18061775

[2] Yuan, M.; Fang, Z.; Xiao, P.; Tong, R.; Zhang, M.; Huang, Y. An FPGA-Based Laser Virtual Scale Method for Structural Crack Measurement. Buildings 2023, 13, 261. https://doi.org/10.3390/buildings13010261

[3] Altera, 2015. MAX 10 NEEK (NEEK-MAX10) Development Kit User Guide
https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/ug/ug-max10-neek.pdf

[4] Ju, M.; Dou, Z.; Li, J.-W.; Qiu, X.; Shen, B.; Zhang, D.; Yao, F.-Z.; Gong, W.; Wang, K. Piezoelectric Materials and Sensors for Structural Health Monitoring: Fundamental Aspects, Current Status, and Future Perspectives. Sensors 2023, 23, 543. https://doi.org/10.3390/s23010543

[5] Xiang, W.; Wei, J.; Zhang, F. Structural Health Monitoring Design and Performance Evaluation of a Middle-Span Bridge. Sensors 2023, 23, 8702.
https://doi.org/10.3390/s23218702

[6] Zymelka D.; Togashi K.; Kobayashi T.; Concentric Array of Printed Strain Sensors for Structural Health Monitoring. Sensors 2020, 20(7), 1997;
https://doi.org/10.3390/s20071997

[7] Cheilakou, E., Tsopelas, N., Anastasopoulos, A., Kourousis, D., Rychkov, D., Gerhard, R., Frankenstein, B., Amditis, A., Damigos, Y., Bouklas, C., 2018. Strain monitoring system for steel and concrete structures. Proceedings of Structural Integrity, 1025-1032. https://doi.org/10.1016/j.prostr.2018.09.005

[8] Bae, J., 2020. Introduction to strain gauges. Procedia Structural Integrity, 20(4), https://doi.org/10.1016/j.jcsr.2011.12.006

[9] https://www.omega.com/en-us/resources/data-acquisition

[10] https://img51.gkzhan.com/5/20130726/635104310915586549686.pdf

[11]https://www.grc.nasa.gov/www/k-12/airplane/tunwheat.html

[12] https://www.omega.com/en-us/resources/signal-conditioners

[13] https://www.omega.com/en-us/resources/types-of-signal-conditioners

[14] Xilinx, 2023. *What is an FPGA?*.
https://www.xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.html

[15] Intel® FPGA Basics and Getting Started
https://www.intel.com/content/www/us/en/support/programmable/support-resources/fpga-training/getting-started.html

[16]https://documentation.altera.com/#/link/sam1393576011848/sam1393996851885

[17] https://www.intel.com/content/www/us/en/docs/programmable/683732/17-0/altpll-phase-locked-loop-ip-core-user-guide.html

[18]https://www.intel.com/content/www/us/en/programmable/quartushelp/17.0/hdl/mega/mega_file_lpm_mux_etc.htm

Appendix 1 – FPGA Based Data Acquisition System for Strain Gauge Array Monitoring

Evangelia-Agni ALEXOPOULOU, Efstathios D. KYRIAKIS-BITZAROS, Ilias DARADIMOS, Ermioni D. PASIOU, Ioannis CHRISTAKIS, Ilias STAVRAKAS, FPGA BASED DATA ACQUISITION SYSTEM FOR STRAIN GAUGE ARRAY MONITORING, 39th Danubia-Adria Symposium on Advances in Experimental Mechanics, 26-29 Sept, 2023, Siofok, Hungary.

39th Danubia-Adria Symposium on Advances in Experimental Mechanics
Sep 26-29, 2023
Siófok, Hungary

. The proposed system can be extended using existing LEON3 peripherals to include features such as sensor's raw data storage on SD memory card, networking capabilities through the Ethernet interface and data visualization.

www.das2023.hu

FPGA BASED DATA ACQUISITION SYSTEM FOR STRAIN GAUGE ARRAY MONITORING

Evangelia-Agni ALEXOPOULOU[1], Efstathios D. KYRIAKIS-BITZAROS[1], Ilias DARADIMOS[1], Ermioni D. PASIOU[2], Ilias STAVRAKAS[1]

1. Department of Electrical and Electronics Engineering, Faculty of Engineering, University of West Attica, Thivon Avenue 250, GR-12241 Athens, Greece,
Email: evangelia_alexopoulou@outlook.com; mpitz@uniwa.gr; drid@uniwa.gr; ilias@uniwa.gr;

2. National Technical University of Athens, Department of Mechanics, Laboratory for Testing and Materials, 5 Heroes of Polytechnion Avenue, Theocaris Building, 157 73, Athens, Greece, E-mail: epasiou@teemail.gr;

Introduction

Structural monitoring is crucial for ensuring the safety and efficiency of various structures, since the measurement of the deformations and strains developed could be an indicator of potential structural problems [1]. Recent communication and electronics developments enabled the scientific community to design and implement various systems for the monitoring of strain changes in laboratory and field scales [2-4]. In this work a complete strain gauge data acquisition system is designed and implemented using a Field Programmable Gate Array (FPGA), supported by a signal conditioning circuit, capable of performing data analysis and visualization locally.
The concept was to:

a. obtain high strain sampling rate
b. local processing capabilities
c. real-time data transfer capabilities and data visualization capabilities presented as strain distribution on the specimen's surface.

Materials & Methods

The 4 modules that comprise the system are, the strain gauges, the resistor bridge, the signal conditioning amplifier unit and the FPGA unit (see Fig.1). The complete system is designed and preliminary tested at the Electronic Devices and Materials Laboratory (EDML) of the University of West Attica and is open for public access [5]. The strain gauges are the KYOWA KFG-5-120- C1-11L3M2R, and the resistor bridge is the Microlink 770. The signal conditioning unit is build using an array of the Texas Instruments TLV9002D dual Operational Amplifier and the Terasic's DE-10 Lite development kit incorporating an Intel's MAX-10 FPGA device.

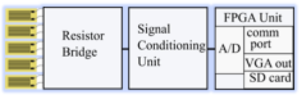


Fig. 1. Block diagram of the system.

Fig. 2. Schematic diagram of the signal conditioning unit

Designed concepts and preliminary results

The nominal resistance (R) of the strain gauges is 119.6Ω and the gauge factor (k) is 2.09. Eq. (1) is used to calculate the strain (ε): $k = \frac{\frac{\Delta R}{R}}{\frac{\Delta L}{L}} = \frac{\frac{\Delta R}{R}}{\varepsilon}$ (1) where ΔR is the resistance change due the change of the length of the strain gauge during loading. In this case the bridge excitation voltage is set to 2.5V. The strain gauge participates as member at the resistor bridge board that is capable to host up to 15 quarter bridge channels. The output of the bridge fed to a signal conditioning analog amplifier circuit. The schematic circuit shown in Fig.2 is the signal conditioning unit utilizing the TLV9002 QDRQ1 instrumentation amplifier with a gain of 60dB. It is designed to measure the changes in resistance of a strain gauge sensor having a nominal resistance of 120Ohms. The circuit produces an output voltage that can be measured and analyzed and is linearly related to the resistance changes. The power supply for the circuit is 5V, providing the necessary voltage for the amplifier to operate. Additionally, there is a reference voltage of 2.5V that biases the output voltage of the instrumentation amplifier to the mid-supply point. This allows for differential measurements in both positive and negative directions. The corresponding PCB was manufactured using the CNC CAT-Eco router. Five signal conditioning circuits are used in parallel in order to monitor up to five concurrently working strain gauges. The outputs of the signal conditioning circuits are sampled by the 12 bit analog to digital converter of the MAX-10 FPGA at the selected sampling frequency, using a round robin approach. The digitized signals are processed, using a properly configured LEON3 soft-core RISC processor, in order to achieve synchronization and extract the strain level of each channel as well as the strain distribution across the monitored structure. The LEON3 processor is also responsible for the control of the overall system and it may trigger alarms and alerts if the data meets specific criteria, such as exceeding a certain threshold, providing early warning of potential structural problems. An indicative 12bit recording of the digital word that corresponds to a single reading of a strain gauge (in this case channel 4) is shown in Fig.4.

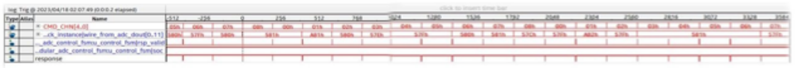


Fig.5. Digital word recording during a clock hit of the FPGA.

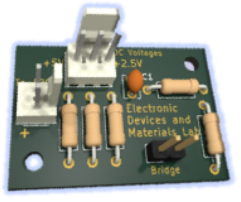


Fig. 3. 3D plot of the signal condition showing the upper side of the circuit.

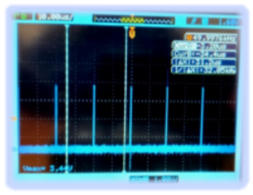


Fig. 4. Timing screenshot.

Conclusions

In this work, an FPGA based data acquisition system for strain gauges array monitoring is designed. The readiness level of the complete system is high having already implemented all the required hardware and data collection steps. In a while it is scheduled to have the SD card storage function ready and the communication protocol implemented in order to be able to store the data remotely and visualize the elaborated data. The proposed solution incorporates the flexibilities that raise from the use of high processing power and multipurpose electronics including the customization capabilities, and fast processing speed, making it an ideal platform for real-time data collection and analysis. The system's ability to be customized to meet specific requirements enables it to be used in a range of applications, from small-scale laboratory testing to large-scale structural monitoring.

References
[1] Bentley, J.P. Principles of measurement systems, Third edition; Prentic Hall, London, 1995.
[2] Cheilakou, E., Tsopelas, N., Anastasopoulos, A., Kourousis, D., Rychkov, D., Gerhard, R., Frankenstein, B., Amditis, A., Damigos, Y., Bouklas, C. Strain monitoring system for steel and concrete structures. Proc Struct Int, 2018, 10,25-32.
[3] Sause, M.G.R., Jasiūnienė, E. Structural health monitoring damage detection systems for aerospace. Springer Nature, 2021.
[4] Hongell, T., Ihalainen, J., Hakala, I. CiNetStrain - wireless strain gauge network - Calibration and reliability measurements, Eds. Zheng, J., Mitton, N., Li, J., Lorenz, P., Ad Hoc Networks, 275, ISBN 978- 3-642-36957-5.
[5] EDML gitlab site: https://gitlab.com/uniwaedml/strain-gauge-amplifier

Appendix 2 – VHDL Code for the Register of the Bus Slicer

```
-- Include the IEEE standard library for logical operations

library IEEE;

use IEEE.STD_LOGIC_1164.ALL;


-- Define the BusSlicer entity with input and output ports

entity BusSlicer is

    Port (

        input_bus : in STD_LOGIC_VECTOR(11 downto 0); -- 12-bit input bus

        clk : in STD_LOGIC;                          -- Clock input

        high_output_bus : out STD_LOGIC_VECTOR(7 downto 0); -- 8-bit high output bus

        low_output_bus : out STD_LOGIC_VECTOR(7 downto 0)   -- 8-bit low output bus

    );

end BusSlicer;


-- Define the architecture named 'Behavioral' for the BusSlicer entity

architecture Behavioral of BusSlicer is

begin

    -- Process triggered on the rising edge of the clock

    process(clk) is

    begin

        -- Check for rising edge of the clock

        if rising_edge(clk) then

            -- Assign the upper 4 bits of the input bus to the high output bus

            -- and ground the lower 4 bits of the high output bus

            high_output_bus <= "0000" & input_bus(11 downto 8);


            -- Assign the lower 8 bits of the input bus to the low output bus

            low_output_bus <= input_bus(7 downto 0);

        end if;

    end process;

end Behavioral;
```

Appendix 3 – VHDL CODE for the Transmitter

```
-- Include the IEEE standard library for logical operations

library ieee;

use ieee.std_logic_1164.all;

use ieee.numeric_std.all;

-- Define the UART_TX entity with generics and ports

entity UART_TX is

  generic (

    g_CLKS_PER_BIT : integer := 217  -- Generic value for the number of clock cycles
per bit

    );

  port (

    i_Clk       : in  std_logic;              -- Clock input

    i_TX_DV     : in  std_logic;              -- Data Valid signal input

    i_TX_Byte   : in  std_logic_vector(7 downto 0);  -- Byte to be transmitted

    o_TX_Active : out std_logic;              -- Indicates transmission is active

    o_TX_Serial : out std_logic;              -- Serial output bit

    o_TX_Done   : out std_logic               -- Indicates transmission is done

    );

end UART_TX;

-- Architecture definition of UART_TX

architecture RTL of UART_TX is

  -- State machine states

  type t_SM_Main is (s_Idle, s_TX_Start_Bit, s_TX_Data_Bits, s_TX_Stop_Bit,
s_Cleanup);

  signal r_SM_Main : t_SM_Main := s_Idle;  -- Current state of the state machine

  -- Internal signals

  signal r_Clk_Count : integer range 0 to  g_CLKS_PER_BIT-1 := 0;  -- Clock cycle
counter

  signal r_Bit_Index : integer range 0 to 7 := 0;  -- Index for the data bits (8 bits
total)

  signal r_TX_Data   : std_logic_vector(7 downto 0) := (others => '0');  -- Data to
be transmitted

  signal r_TX_Done   : std_logic := '0';  -- Transmission done signal

begin

  -- Process for UART transmission

  p_UART_TX : process (i_Clk)

  begin
```

```
    if rising_edge(i_Clk) then  -- Trigger on the rising edge of the clock
      case r_SM_Main is
        -- Idle state: wait for Data Valid signal
        when s_Idle =>
          o_TX_Active <= '0';  -- Transmission inactive
          o_TX_Serial <= '1';  -- Drive line high for idle state
          r_TX_Done   <= '0';  -- Clear transmission done flag
          r_Clk_Count <= 0;    -- Reset clock count
          r_Bit_Index <= 0;    -- Reset bit index
          if i_TX_DV = '1' then  -- If Data Valid signal is high
            r_TX_Data <= i_TX_Byte;  -- Load data byte to be transmitted
            r_SM_Main <= s_TX_Start_Bit;  -- Move to Start Bit state
          else
            r_SM_Main <= s_Idle;  -- Stay in Idle state
          end if;
        -- Start Bit state: send start bit (0)
        when s_TX_Start_Bit =>
          o_TX_Active <= '1';  -- Transmission active
          o_TX_Serial <= '0';  -- Start bit (0)
          if r_Clk_Count < g_CLKS_PER_BIT-1 then  -- Wait for the duration of a start
bit
            r_Clk_Count <= r_Clk_Count + 1;  -- Increment clock count
            r_SM_Main   <= s_TX_Start_Bit;  -- Stay in Start Bit state
          else
            r_Clk_Count <= 0;  -- Reset clock count
            r_SM_Main   <= s_TX_Data_Bits;  -- Move to Data Bits state
          end if;
        -- Data Bits state: send data bits
        when s_TX_Data_Bits =>
          o_TX_Serial <= r_TX_Data(r_Bit_Index);  -- Output the current data bit
          if r_Clk_Count < g_CLKS_PER_BIT-1 then  -- Wait for the duration of a data
bit
            r_Clk_Count <= r_Clk_Count + 1;  -- Increment clock count
            r_SM_Main   <= s_TX_Data_Bits;  -- Stay in Data Bits state
          else
            r_Clk_Count <= 0;  -- Reset clock count
            if r_Bit_Index < 7 then  -- If not all bits are sent
              r_Bit_Index <= r_Bit_Index + 1;  -- Move to the next bit
              r_SM_Main   <= s_TX_Data_Bits;  -- Stay in Data Bits state
            else
              r_Bit_Index <= 0;  -- Reset bit index
              r_SM_Main   <= s_TX_Stop_Bit;  -- Move to Stop Bit state
            end if;
          end if;
        -- Stop Bit state: send stop bit (1)
        when s_TX_Stop_Bit =>
          o_TX_Serial <= '1';  -- Stop bit (1)
          if r_Clk_Count < g_CLKS_PER_BIT-1 then  -- Wait for the duration of a stop
bit
            r_Clk_Count <= r_Clk_Count + 1;  -- Increment clock count
            r_SM_Main   <= s_TX_Stop_Bit;  -- Stay in Stop Bit state
          else
            r_TX_Done   <= '1';  -- Set transmission done flag
            r_Clk_Count <= 0;  -- Reset clock count
            r_SM_Main   <= s_Cleanup;  -- Move to Cleanup state
          end if;
        -- Cleanup state: finalize transmission
        when s_Cleanup =>
          o_TX_Active <= '0';  -- Transmission inactive
          r_TX_Done   <= '1';  -- Keep transmission done flag high
          r_SM_Main   <= s_Idle;  -- Move to Idle state
        -- Default case (should not occur)
        when others =>
          r_SM_Main <= s_Idle;  -- Move to Idle state
      end case;
    end if;
  end process p_UART_TX;
  -- Output assignment
  o_TX_Done <= r_TX_Done;  -- Connect internal done signal to output

end RTL;
```

Appendix 4 – Verilog Code for the Receiver

```
module async_receiver(

    input clk,           // Clock input

    input RxD,           // Serial data input

    output RxD_data_ready, // One clock pulse when RxD_data is valid

    output [7:0] RxD_data, // Received data byte

    output RxD_endofpacket, // One clock pulse when no more data is received

    output RxD_idle        // Indicates no data is being received

);


parameter ClkFrequency = 25000000; // Clock frequency (25MHz)

parameter Baud = 115200;           // Baud rate for UART communication


// Baud generator for 8x oversampling

parameter Baud8 = Baud * 8;        // 8 times the Baud rate for oversampling

parameter Baud8GeneratorAccWidth = 16;

wire [Baud8GeneratorAccWidth:0] Baud8GeneratorInc = ((Baud8 <<
(Baud8GeneratorAccWidth - 7)) + (ClkFrequency >> 8)) / (ClkFrequency >> 7);

reg [Baud8GeneratorAccWidth:0] Baud8GeneratorAcc;

always @(posedge clk) Baud8GeneratorAcc <= Baud8GeneratorAcc[Baud8GeneratorAccWidth-
1:0] + Baud8GeneratorInc;

wire Baud8Tick = Baud8GeneratorAcc[Baud8GeneratorAccWidth]; // Baud rate tick signal


// Synchronize and invert the RxD signal

reg [1:0] RxD_sync_inv;

always @(posedge clk) if (Baud8Tick) RxD_sync_inv <= {RxD_sync_inv[0], ~RxD};

// Inverting RxD so idle becomes "0", avoiding phantom characters at startup


// Debounce and synchronize the RxD signal

reg [1:0] RxD_cnt_inv;

reg RxD_bit_inv;


always @(posedge clk)

if (Baud8Tick) begin

    if (RxD_sync_inv[1] && RxD_cnt_inv != 2'b11) RxD_cnt_inv <= RxD_cnt_inv + 2'h1;

    else if (~RxD_sync_inv[1] && RxD_cnt_inv != 2'b00) RxD_cnt_inv <= RxD_cnt_inv -
2'h1;
```

```
    if (RxD_cnt_inv == 2'b00) RxD_bit_inv <= 1'b0;
    else if (RxD_cnt_inv == 2'b11) RxD_bit_inv <= 1'b1;
end

// State machine for receiving data
reg [3:0] state;
reg [3:0] bit_spacing;

// "next_bit" controls when the data sampling occurs
// Values from 8 to 11 typically work well depending on connection quality
wire next_bit = (bit_spacing == 4'd10);

always @(posedge clk)
if (state == 0)
    bit_spacing <= 4'b0000;
else if (Baud8Tick)
    bit_spacing <= {bit_spacing[2:0] + 4'b0001} | {bit_spacing[3], 3'b000};

// State machine transitions for receiving each bit
always @(posedge clk)
if (Baud8Tick) begin
    case (state)
        4'b0000: if (RxD_bit_inv) state <= 4'b1000;  // Start bit detected?
        4'b1000: if (next_bit) state <= 4'b1001;  // Bit 0
        4'b1001: if (next_bit) state <= 4'b1010;  // Bit 1
        4'b1010: if (next_bit) state <= 4'b1011;  // Bit 2
        4'b1011: if (next_bit) state <= 4'b1100;  // Bit 3
        4'b1100: if (next_bit) state <= 4'b1101;  // Bit 4
        4'b1101: if (next_bit) state <= 4'b1110;  // Bit 5
        4'b1110: if (next_bit) state <= 4'b1111;  // Bit 6
        4'b1111: if (next_bit) state <= 4'b0001;  // Bit 7
        4'b0001: if (next_bit) state <= 4'b0000;  // Stop bit
        default: state <= 4'b0000;  // Default to idle state
    endcase
end
```

```
// Shift register to store received data bits

reg [7:0] RxD_data;

always @(posedge clk)

if (Baud8Tick && next_bit && state[3])

    RxD_data <= {~RxD_bit_inv, RxD_data[7:1]};


// Data ready and error signals

reg RxD_data_ready, RxD_data_error;

always @(posedge clk) begin

    RxD_data_ready <= (Baud8Tick && next_bit && state == 4'b0001 && ~RxD_bit_inv);
// Data ready on valid stop bit

    RxD_data_error <= (Baud8Tick && next_bit && state == 4'b0001 &&  RxD_bit_inv);
// Error on invalid stop bit

end


// End of packet detection

reg [4:0] gap_count;

always @(posedge clk)

if (state != 0)

    gap_count <= 5'h00;

else if (Baud8Tick && ~gap_count[4])

    gap_count <= gap_count + 5'h01;


assign RxD_idle = gap_count[4]; // Idle signal if no data received for a while


// End of packet signal

reg RxD_endofpacket;

always @(posedge clk)

    RxD_endofpacket <= Baud8Tick && (gap_count == 5'h0F);


Endmodule
```

Appendix 5 – VHDL Code for the Register of the Transmitter

```
-- Include the IEEE standard library for logical operations

library IEEE;

use IEEE.std_logic_1164.all;

-- Define the reg_trans entity with input and output ports

entity reg_trans is

    port (

        signal reg_trans_input : in  std_logic_vector(7 downto 0); -- 8-bit input
signal

        signal clk             : in  std_logic;                    -- Clock input

        signal reg_trans_output: out std_logic_vector(7 downto 0)  -- 8-bit output
signal

    );

end reg_trans;


-- Define the architecture named 'sim' for the reg_trans entity

architecture sim of reg_trans is

begin

    -- Process triggered on the rising edge of the clock

    process(clk) is

    begin

        -- Check for rising edge of the clock

        if rising_edge(clk) then

            -- On rising edge, transfer input data to output

            reg_trans_output <= reg_trans_input;

        end if;

    end process;

end sim;
```

Appendix 6 – Matlab Code for the Serial Communication

```
% Set up the serial port with the specified parameters

s = serialport("COM8", 115200, Parity="none", StopBits= 1, DataBits= 8);

s.Timeout = 0.1;  % Set the timeout for serial communication

np = 5; % Number of points for averaging

% Initialize buffers for each channel

buffer_ch6 = zeros(1, np);

buffer_ch5 = zeros(1, np);

buffer_ch4 = zeros(1, np);

buffer_ch3 = zeros(1, np);

buffer_ch2 = zeros(1, np);

buffer_ch1 = zeros(1, np);

% Initialize counters for each channel

count_ch6 = 0;

count_ch5 = 0;

count_ch4 = 0;

count_ch3 = 0;

count_ch2 = 0;

count_ch1 = 0;

% Set up the figure for plotting

figure;

% Initialize animated lines for each channel with different colors

hAve6 = animatedline('Color', 'b'); % For data_ch6, in blue

hAve5 = animatedline('Color', 'r'); % For data_ch5, in red

hAve4 = animatedline('Color', 'g'); % For data_ch4, in green

hAve3 = animatedline('Color', 'm'); % For data_ch3, in magenta

hAve2 = animatedline('Color', 'c'); % For data_ch2, in cyan

hAve1 = animatedline('Color', 'k'); % For data_ch1, in black

% Configure plot axes and title

xlabel('Time');

ylabel('Average Data Value');

title('Real-time Average Data Plot');

grid on;

% Display start message

disp('Starting data reception...');
```

```
% Initialize batch indices for each channel

batchIndex6 = 1;

batchIndex5 = 1;

batchIndex4 = 1;

batchIndex3 = 1;

batchIndex2 = 1;

batchIndex1 = 1;

% Initialize average calculations for each channel

calc1 = -1;

calc2 = -1;

calc3 = -1;

calc4 = -1;

calc5 = -1;

calc6 = -1;

% Open a file for writing and write header line

fileID = fopen('VoltageData.txt', 'w');

fprintf(fileID, 'Channel 1, Channel 2, Channel 3, Channel 4, Channel 5, Channel
6\n');

fclose(fileID);

% Infinite loop to continuously read and process data

while true

    % Send example commands to the serial port

    write(s, bin2dec('00011100'), "uint8");

    pause(0.001);

    write(s, bin2dec('00011111'), "uint8");

    pause(0.001);

    write(s, bin2dec('00011101'), "uint8");

    flush(s);

    % Read high byte for channel 6

    data_high_ch6 = read(s, 1, "uint8");

    pause(0.001);

    write(s, bin2dec('00011110'), "uint8");

    pause(0.001);

    write(s, bin2dec('00011100'), "uint8");

    pause(0.001);

    flush(s);
```

```
% Read low byte for channel 6
data_low_ch6 = read(s, 1, "uint8");
% Combine high and low bytes for channel 6
data_ch6 = bitshift(uint16(data_high_ch6), 8) + uint16(data_low_ch6);
% Process data for channel 6
if ~isempty(data_ch6) && isscalar(data_ch6)
    count_ch6 = count_ch6 + 1;
    buffer_ch6(count_ch6) = data_ch6;
    if count_ch6 == np
        average_ch6 = mean(buffer_ch6);
        if calc6 == -1
            calc6 = average_ch6;
        else
            average_ch6 = average_ch6 - calc6;
            addpoints(hAve6, batchIndex6, average_ch6);
        end
        batchIndex6 = batchIndex6 + 1;
        count_ch6 = 0;
        buffer_ch6 = zeros(1, np); % Reset buffer
    end
end
% Repeat similar steps for channel 5
write(s, bin2dec('00011000'), "uint8");
pause(0.001);
write(s, bin2dec('00011011'), "uint8");
pause(0.001);
write(s, bin2dec('00011001'), "uint8");
flush(s);
data_high_ch5 = read(s, 1, "uint8");
pause(0.001);
write(s, bin2dec('00011010'), "uint8");
pause(0.001);
write(s, bin2dec('00011000'), "uint8");
pause(0.001);
flush(s);
```

```
data_low_ch5 = read(s, 1, "uint8");
data_ch5 = bitshift(uint16(data_high_ch5), 8) + uint16(data_low_ch5);
if ~isempty(data_ch5) && isscalar(data_ch5)
    count_ch5 = count_ch5 + 1;
    buffer_ch5(count_ch5) = data_ch5;
    if count_ch5 == np
        average_ch5 = mean(buffer_ch5);
        if calc5 == -1
            calc5 = average_ch5;
        else
            average_ch5 = average_ch5 - calc5;
            addpoints(hAve5, batchIndex5, average_ch5);
        end
        batchIndex5 = batchIndex5 + 1;
        count_ch5 = 0;
        buffer_ch5 = zeros(1, np); % Reset buffer
    end
end
% Repeat similar steps for channel 4
write(s, bin2dec('00010100'), "uint8");
pause(0.001);
write(s, bin2dec('00010111'), "uint8");
pause(0.001);
write(s, bin2dec('00010101'), "uint8");
flush(s);
data_high_ch4 = read(s, 1, "uint8");
pause(0.001);
write(s, bin2dec('00010110'), "uint8");
pause(0.001);
write(s, bin2dec('00010100'), "uint8");
pause(0.001);
flush(s);
data_low_ch4 = read(s, 1, "uint8");
data_ch4 = bitshift(uint16(data_high_ch4), 8) + uint16(data_low_ch4);
if ~isempty(data_ch4) && isscalar(data_ch4)
    count_ch4 = count_ch4 + 1;
```

```
        buffer_ch4(count_ch4) = data_ch4;
        if count_ch4 == np
            average_ch4 = mean(buffer_ch4);
            if calc4 == -1
                calc4 = average_ch4;
            else
                average_ch4 = average_ch4 - calc4;
                addpoints(hAve4, batchIndex4, average_ch4);
            end
            batchIndex4 = batchIndex4 + 1;
            count_ch4 = 0;
            buffer_ch4 = zeros(1, np); % Reset buffer
        end
    end
    % Repeat similar steps for channel 3
    write(s, bin2dec('00010000'), "uint8");
    pause(0.001);
    write(s, bin2dec('00010011'), "uint8");
    pause(0.001);
    write(s, bin2dec('00010001'), "uint8");
    flush(s);
    data_high_ch3 = read(s, 1, "uint8");
    pause(0.001);
    write(s, bin2dec('00010010'), "uint8");
    pause(0.001);
    write(s, bin2dec('00010000'), "uint8");
    pause(0.001);
    flush(s);
    data_low_ch3 = read(s, 1, "uint8");
    data_ch3 = bitshift(uint16(data_high_ch3), 8) + uint16(data_low_ch3);
    if ~isempty(data_ch3) && isscalar(data_ch3)
        count_ch3 = count_ch3 + 1;
        buffer_ch3(count_ch3) = data_ch3;
        if count_ch3 == np
            average_ch3 = mean(buffer_ch3);
            if calc3 == -1
```

```
                calc3 = average_ch3;
            else
                average_ch3 = average_ch3 - calc3;
                addpoints(hAve3, batchIndex3, average_ch3);
            end
            batchIndex3 = batchIndex3 + 1;
            count_ch3 = 0;
            buffer_ch3 = zeros(1, np); % Reset buffer
        end
    end
    % Repeat similar steps for channel 2
    write(s, bin2dec('00001100'), "uint8");
    pause(0.001);
    write(s, bin2dec('00001111'), "uint8");
    pause(0.001);
    write(s, bin2dec('00001101'), "uint8");
    flush(s);
    data_high_ch2 = read(s, 1, "uint8");
    pause(0.001);
    write(s, bin2dec('00001110'), "uint8");
    pause(0.001);
    write(s, bin2dec('00001100'), "uint8");
    pause(0.001);
    flush(s);
    data_low_ch2 = read(s, 1, "uint8");
    data_ch2 = bitshift(uint16(data_high_ch2), 8) + uint16(data_low_ch2);
    if ~isempty(data_ch2) && isscalar(data_ch2)
        count_ch2 = count_ch2 + 1;
        buffer_ch2(count_ch2) = data_ch2;
        if count_ch2 == np
            average_ch2 = mean(buffer_ch2);
            if calc2 == -1
                calc2 = average_ch2;
            else
                average_ch2 = average_ch2 - calc2;
                addpoints(hAve2, batchIndex2, average_ch2);
            end
            batchIndex2 = batchIndex2 + 1;
            count_ch2 = 0;
            buffer_ch2 = zeros(1, np); % Reset buffer
        end
    end
    % Repeat similar steps for channel 1
    write(s, bin2dec('00000100'), "uint8");
    pause(0.001);
    write(s, bin2dec('00000111'), "uint8");
    pause(0.001);
    write(s, bin2dec('00000101'), "uint8");
    flush(s);
    data_high_ch1 = read(s, 1, "uint8");
    pause(0.001);
    write(s, bin2dec('00000110'), "uint8");
    pause(0.001);
    write(s, bin2dec('00000100'), "uint8");
    pause(0.001);
    flush(s);
    data_low_ch1 = read(s, 1, "uint8");
    data_ch1 = bitshift(uint16(data_high_ch1), 8) + uint16(data_low_ch1);
    if ~isempty(data_ch1) && isscalar(data_ch1)
        count_ch1 = count_ch1 + 1;
        buffer_ch1(count_ch1) = data_ch1;
        if count_ch1 == np
            average_ch1 = mean(buffer_ch1);
            if calc1 == -1
                calc1 = average_ch1;
            else
                average_ch1 = average_ch1 - calc1;
                addpoints(hAve1, batchIndex1, average_ch1);
            end
            batchIndex1 = batchIndex1 + 1;
            count_ch1 = 0;
            buffer_ch1 = zeros(1, np); % Reset buffer
        end

    end

        % Print data for debugging purposes

    fprintf("Channel 6 = %d %d %d %f \n", data_high_ch6, data_low_ch6, data_ch6,
(double(data_ch6) * 3.3) / 4095.);

    fprintf("Channel 5 = %d %d %d %f \n", data_high_ch5, data_low_ch5, data_ch5,
(double(data_ch5) * 3.3) / 4095.);

    fprintf("Channel 4 = %d %d %d %f \n", data_high_ch4, data_low_ch4, data_ch4,
(double(data_ch4) * 3.3) / 4095.);

    fprintf("Channel 3 = %d %d %d %f \n", data_high_ch3, data_low_ch3, data_ch3,
(double(data_ch3) * 3.3) / 4095.);

    fprintf("Channel 2 = %d %d %d %f \n", data_high_ch2, data_low_ch2, data_ch2,
(double(data_ch2) * 3.3) / 4095.);

    fprintf("Channel 1 = %d %d %d %f \n", data_high_ch1, data_low_ch1, data_ch1,
(double(data_ch1) * 3.3) / 4095.);

        % Update the plot

    drawnow;

        % Convert averages to voltage and write to file

    c6 = (double(average_ch6) * 3.3) / 4095.;

    c5 = (double(average_ch5) * 3.3) / 4095.;

    c4 = (double(average_ch4) * 3.3) / 4095.;

    c3 = (double(average_ch3) * 3.3) / 4095.;

    c2 = (double(average_ch2) * 3.3) / 4095.;

    c1 = (double(average_ch1) * 3.3) / 4095.;

        % Open the file for appending and write the data

    fileID = fopen('VoltageData.txt', 'a');

    fprintf(fileID, '%f,%f,%f,%f,%f,%f\n', c1, c2, c3, c4, c5, c6);

    fclose(fileID);

end
```

Appendix 7 – Embedded C Code of the main.c

```
#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/alt_alarm.h>

#include <system.h>

#include "io.h"

#include "app_selector.h"

#include "alt_video_display.h"

#include "simple_graphics.h"

#include "fonts.h"

#include "terasic_includes.h"

#include "math.h"

#include "alt_types.h"

#include "altera_modular_adc.h"

#include "altera_modular_adc_sequencer_regs.h"

#include "altera_modular_adc_sample_store_regs.h"


// Define color constants for easy reference

#define COLOR_WHITE 0xFFFF

#define COLOR_BLACK 0x0000

#define COLOR_RED 0xF800

#define COLOR_GREEN 0x07E0

#define COLOR_BLUE 0x001F


// Define ADC base address and data register address for easier access

#define ADC_BASE MODULAR_ADC_0_SAMPLE_STORE_CSR_BASE

#define ADC_DATA_REG ALTERA_MODULAR_ADC_SAMPLE_STORAGE_REG


// Define screen dimensions and maximum ADC value

#define SCREEN_WIDTH 800

#define SCREEN_HEIGHT 480

#define MAX_ADC_VALUE 4096  // Assuming a 12-bit ADC

// Function to read a value from the ADC

int read_adc() {
    alt_u32 adc_val;

    alt_adc_word_read(ADC_BASE, &adc_val, 1); // Read a single word from ADC

    printf("ADC Value Read: %u\n", adc_val);  // Print the read value for debugging

    return (int)adc_val;  // Return the read ADC value as an integer

}


// Function to initialize the LCD display

alt_video_display* init_lcd() {

    // Print message indicating LCD initialization

    printf("Initializing LCD display...\n");

    // Initialize the LCD display with specified parameters

    alt_video_display *display = alt_video_display_init(

        ALT_VIDEO_DISPLAY_SGDMA_NAME,          // Name of SGDMA device

        ALT_VIDEO_DISPLAY_COLS,                // Number of columns

        ALT_VIDEO_DISPLAY_ROWS,                // Number of rows

        ALT_VIDEO_DISPLAY_COLOR_DEPTH,         // Color depth

        ALT_VIDEO_DISPLAY_USE_HEAP,            // Buffer location

        ALT_VIDEO_DISPLAY_USE_HEAP,            // Descriptor location

        2                                      // Number of buffers

    );

    // Check if initialization was successful

    if (display == NULL) {

        printf("Error initializing video display\n");

        return NULL;  // Return NULL if initialization failed

    }

    // Clear the display

    printf("Clearing display...\n");

    alt_video_display_register_written_buffer(display); // Register the written
buffer

    alt_video_display_clear_screen(display, 0xFFFF); // Clear screen with white color

    return display;  // Return the initialized display

}


// Function to plot a point on the display

void plot_point(alt_video_display* display, int x, int y, int color) {

    printf("Plotting point at (%d, %d) with color %d\n", x, y, color);  // Print the
point coordinates and color
```

```
    vid_set_pixel(x, y, color, display);  // Plot the point on the display

}

// Function to draw a graph on the display

void draw_graph(alt_video_display* display, int *values, int length) {

    int i;

    int prev_x = 40, prev_y = SCREEN_HEIGHT - 40 - (values[0] * (SCREEN_HEIGHT - 80)
/ MAX_ADC_VALUE);

    printf("Starting to draw graph...\n");


    // Loop through the values and draw lines connecting the points

    for (i = 1; i < length; i++) {

        int x = 40 + (i * (SCREEN_WIDTH - 80)) / length;

        int y = SCREEN_HEIGHT - 40 - (values[i] * (SCREEN_HEIGHT - 80) /
MAX_ADC_VALUE);

        printf("Drawing line from (%d, %d) to (%d, %d)\n", prev_x, prev_y, x, y);

        vid_draw_line(prev_x, prev_y, x, y, 1, 0x0000, display);  // Draw line in
black color

        prev_x = x;  // Update previous x coordinate

        prev_y = y;  // Update previous y coordinate

    }

}


// Function to draw the axes on the display

void draw_axes(alt_video_display* display) {

    // Draw x-axis

    printf("Drawing x-axis...\n");

    vid_draw_line(40, SCREEN_HEIGHT - 40, SCREEN_WIDTH - 40, SCREEN_HEIGHT - 40, 1,
0x0000, display);  // Black color


    // Draw y-axis

    printf("Drawing y-axis...\n");

    vid_draw_line(40, 40, 40, SCREEN_HEIGHT - 40, 1, 0x0000, display);  // Black
color


    // Add x-axis label

    printf("Adding x-axis label...\n");

    vid_print_string(120, SCREEN_HEIGHT - 35, 0x0000, tahomabold_32, display,
"Time");  // Black color, Tahoma bold font
```

```
    // Add y-axis label
    printf("Adding y-axis label...\n");
    vid_print_string(5, 100, 0x0000, tahomabold_32, display, "ADC Value");  // Black 
color, Tahoma bold font
}


int main() {
    printf("Starting main program...\n");
    alt_video_display* display = init_lcd();  // Initialize the LCD display
    if (display == NULL) {
        printf("LCD initialization failed. Exiting...\n");
        return -1; // Exit if LCD initialization fails
    }

    int adc_values[80];  // Array to store ADC values
    int i;

    // Sample ADC values
    printf("Sampling ADC values...\n");
    for (i = 0; i < 80; i++) {
        adc_start(MODULAR_ADC_0_SEQUENCER_CSR_BASE);  // Start ADC conversion
        usleep(100000); // Delay to simulate sampling rate
        adc_values[i] = read_adc();  // Read ADC value and store in array
    }

    // Clear screen and draw graph
    printf("Drawing graph on the display...\n");
    draw_axes(display);  // Draw the axes
    draw_graph(display, adc_values, 80);  // Draw the graph with sampled values

    // Register written buffer and display it
    printf("Registering written buffer...\n");
    alt_video_display_register_written_buffer(display);

    printf("Entering continuous update loop...\n");
    while (1) {
        // Continuous update loop if needed
    }

    return 0;  // Return success
}
```