

DATA ACQUISITION SYSTEM FOR PILOT MILL

by

ISAIH KGABE MOLEPO

submitted in accordance with the requirements
for the degree of

MAGISTER TECHNOLOGIAE

in the subject

ELECTRICAL ENGINEERING

at the

University of South Africa

SUPERVISOR: Prof. FRANCOIS MULENGA

April 2016

36932388

Declaration

I Isaih Kgabe Molepo declare that the electronic copy submitted to UNISA represent the final examined and approved version of the dissertation.


..... (Signature)

01 June 2016
..... (Date)

Abstract

This dissertation describes the development, design, implementation and evaluation of a data acquisition system, with the main aim of using it for data collection on a laboratory pilot ball mill. An open-source prototype hardware platform was utilised in the implementation of the data acquisition function, however, with limitations. An analogue signal conditioning card has been successfully developed to interface the analogue signals to the dual domain ADC module. Model-based software development was used to design and develop the algorithms to control the DAS acquisition process, but with limited capabilities. A GUI application has been developed and used for the collection and storage of the raw data on the host system. The DAS prototype was calibrated and collected data successfully through all the channels; however, the input signal bandwidth was limited to 2Hz.

Acknowledgements

- I first want to thank GOD for giving me the strength and wisdom to do this research.
- I am also grateful to my wife for her support.
- I extend my appreciation and thanks to my supervisor Dr. Francois Mulenga for his guidance and support.
- I appreciate the assistance of Khomotso Bopape of Let's Edit (Pty) Ltd for editing my dissertation.
- Finally, I want to thank UNISA for the opportunity to carry out this research and funding for the project.

Table of Contents

| | |
|--|------|
| Declaration..... | ii |
| Abstract..... | iii |
| Acknowledgements | iv |
| List of Abbreviations and Acronyms | xi |
| List of Tables | xiii |
| List of Figures..... | xv |
| Chapter 1 Introduction..... | 1 |
| 1.1 Background | 1 |
| 1.2 Problem Statement | 2 |
| 1.3 Research Aim and Objectives | 3 |
| 1.4 Relevance of this Research..... | 4 |
| 1.5 Research Approach | 4 |
| 1.6 Dissertation Outline..... | 5 |
| Chapter 2 Literature Review..... | 6 |
| 2.1 Introduction | 6 |
| 2.2 Background of Mill Measurements..... | 6 |
| 2.2.1 Inductive Proximity Sensor..... | 7 |
| 2.2.2 Conductivity Sensor..... | 9 |
| 2.3 Sensors and Transducers Technology | 16 |
| 2.3.1 DAS and Sensor Characteristics | 19 |
| 2.3.1.1 Sensor Transfer Function..... | 20 |
| 2.3.1.2 Accuracy | 26 |
| 2.3.1.3 Sensitivity | 27 |
| 2.3.1.4 Resolution | 28 |
| 2.3.1.5 Span | 30 |
| 2.3.1.6 Linearity and Nonlinearity..... | 30 |
| 2.3.1.7 Hysteresis..... | 31 |

| | | |
|-----------|---|----|
| 2.3.1.8 | Noise | 32 |
| 2.3.1.9 | Repeatability | 33 |
| 2.3.1.10 | Dead Band | 34 |
| 2.3.2 | DAS and Sensor Errors | 35 |
| 2.3.2.1 | Systematic Errors | 36 |
| 2.3.2.2 | Random Errors | 37 |
| 2.3.3 | 4-20mA Transmission Protocol | 38 |
| 2.4 | DAS and Signal Processing | 39 |
| 2.4.1 | Data Acquisition System (DAS) | 40 |
| 2.4.2 | Signal Conditioning | 41 |
| 2.4.2.1 | Amplification | 42 |
| 2.4.2.2 | Filtering | 42 |
| 2.4.2.3 | Impedance Matching and Isolation | 43 |
| 2.4.3 | Signal Acquisition and Processing | 44 |
| 2.4.3.1 | Analogue-to-Digital Converter (ADC) | 45 |
| 2.4.3.2 | Signal Processing | 56 |
| Chapter 3 | Design, Testing and Calibration Methodology | 59 |
| 3.1 | Introduction | 59 |
| 3.2 | Design Methodology | 59 |
| 3.2.1 | DAS System Specification | 59 |
| 3.2.2 | Signal Conditioning Card Design | 60 |
| 3.2.3 | PCB Design | 61 |
| 3.2.4 | DAS ADC | 62 |
| 3.2.5 | DAS Controller Embedded Platform | 63 |
| 3.2.6 | DAS Controller Host Platform | 67 |
| 3.3 | DAS Testing | 68 |
| 3.3.1 | Instruments | 68 |
| 3.4 | Signal Conditioning Card Tests | 69 |
| 3.4.1 | DC Tests | 70 |
| 3.4.2 | Signal Chain Tests | 72 |
| 3.5 | Embedded Controller Tests | 75 |

| | | |
|-----------|--|-----|
| 3.6 | Host Controller Tests | 85 |
| 3.7 | ADC Tests | 87 |
| 3.7.1 | ADC DC Test..... | 88 |
| 3.7.2 | ADC AC Test..... | 88 |
| 3.8 | System Calibration | 88 |
| 3.8.1 | Procedures..... | 89 |
| 3.9 | Difficulties Encountered | 90 |
| Chapter 4 | System Description and Model..... | 91 |
| 4.1 | Introduction | 91 |
| 4.2 | DAS System | 91 |
| 4.3 | Signal Conditioning..... | 92 |
| 4.3.1 | Parameters to Be Measured | 92 |
| 4.3.2 | Sensor 4-20mA Transmitter Model | 94 |
| 4.3.3 | Simulink 4-20mA Transmitter Model | 96 |
| 4.3.4 | DAS 4-20mA Receiver Model..... | 101 |
| 4.3.5 | Simulink 4-20mA Receiver Model..... | 102 |
| 4.3.6 | Analogue Filter | 106 |
| 4.4 | Analogue-to-Digital Converter (ADC) | 112 |
| 4.4.1 | Sampler and Hold Modelling..... | 112 |
| 4.4.2 | Simulink Idealiser ADC Quantiser | 114 |
| 4.4.3 | ADC System-level Model..... | 117 |
| 4.5 | Data Acquisition Controller | 119 |
| 4.5.1 | Data Collector | 119 |
| 4.5.2 | Data Processing Controller | 122 |
| 4.5.3 | Data Displaying and Storage Controller | 123 |
| Chapter 5 | DAS Hardware, Firmware and Software | 126 |
| 5.1 | Introduction | 126 |
| 5.2 | Signal Conditioning Card Design | 126 |
| 5.2.1 | Input Stage | 127 |
| 5.2.2 | Buffer Amplifier | 127 |
| 5.2.3 | Differential Amplifier | 129 |

| | | |
|-----------|---|-----|
| 5.2.4 | Signal Scaling | 132 |
| 5.2.5 | Filter Design..... | 133 |
| 5.2.6 | Component Selection | 139 |
| 5.2.7 | Circuit Simulation..... | 144 |
| 5.2.7.1 | AC and DC Performance 5kHz Channel | 145 |
| 5.2.7.2 | AC and DC Performance 50kHz Channel | 154 |
| 5.3 | ADC | 164 |
| 5.3.1 | ADC Architecture | 164 |
| 5.3.2 | Sigma-Delta Converter | 164 |
| 5.3.3 | Successive Approximation Register (SAR)..... | 164 |
| 5.3.4 | ADC Design Specifications | 165 |
| 5.3.4.1 | Resolution | 165 |
| 5.3.4.2 | Acquisition and Conversion Time | 166 |
| 5.3.4.3 | DC Specifications | 168 |
| 5.3.4.4 | AC Specifications | 169 |
| 5.4 | Controller | 169 |
| 5.4.1 | Embedded System Platform..... | 170 |
| 5.4.1.1 | Peripheral Requirements..... | 170 |
| 5.4.1.2 | Memory Requirements | 172 |
| 5.4.1.3 | Processor (CPU) Requirements | 173 |
| 5.4.1.4 | Firmware Requirements..... | 174 |
| 5.4.2 | Host System Platform | 178 |
| 5.4.2.1 | Host Hardware Requirements | 178 |
| 5.4.2.2 | Host Software Requirements | 179 |
| Chapter 6 | DAS Hardware, Firmware and Software Implementation..... | 183 |
| 6.1 | Introduction | 183 |
| 6.2 | Signal Conditioning Card..... | 183 |
| 6.2.1 | 5kHz Channel Implementation | 183 |
| 6.2.2 | 50kHz Channel Implementation | 184 |
| 6.2.3 | PSU Implementation..... | 184 |
| 6.2.4 | Noise Decoupling..... | 185 |

| | | |
|------------|--|-----|
| 6.2.5 | PCB Implementation..... | 185 |
| 6.3 | Analogue-to-Digital Converter (ADC) and Embedded System..... | 187 |
| 6.4 | Controller Firmware..... | 187 |
| 6.4.1 | ADC Channels Subsystem..... | 188 |
| 6.4.2 | Relay Data Subsystem..... | 190 |
| 6.4.3 | Data Formatting Subsystem..... | 190 |
| 6.4.4 | Data Transmission Subsystem..... | 192 |
| 6.4.5 | Control Subsystem..... | 194 |
| 6.4.6 | System Alive Subsystem..... | 196 |
| 6.5 | Host System Software..... | 197 |
| 6.5.1 | DAS Host Graphical User Interface..... | 198 |
| Chapter 7 | Experimental Results and Analysis..... | 202 |
| 7.1 | Introduction..... | 202 |
| 7.2 | Signal Conditioning Card Tests..... | 202 |
| 7.2.1 | DC Test Results..... | 202 |
| 7.2.2 | Signal Chain Results..... | 205 |
| 7.3 | DAS Controller Test Results..... | 212 |
| 7.4 | DAS PC Software Test Results..... | 212 |
| 7.5 | ADC Subsystem Test Results..... | 214 |
| 7.5.1 | DC Test Results..... | 215 |
| 7.5.2 | AC Test Results..... | 220 |
| 7.6 | System Calibration..... | 230 |
| 7.6.1 | Up-scale Calibration..... | 230 |
| 7.7 | Limitations..... | 244 |
| Chapter 8 | Conclusion and Recommendations..... | 245 |
| 8.1 | Conclusion..... | 245 |
| 8.2 | Recommendations..... | 247 |
| References | | 249 |
| Appendix A | Signal Conditioning Card Interface..... | 266 |
| Appendix B | Signal Conditioning Card 3D Views..... | 267 |
| Appendix C | Bare Signal Conditioning PCB..... | 268 |

| | |
|--|-----|
| Appendix D Assembled Signal Conditioning Card..... | 269 |
| Appendix E Signal Conditioning Card Test Set-up | 270 |
| Appendix F Control Signals Subsystem Matlab Code | 271 |
| Appendix G DAS Host GUI Matlab Code | 273 |
| Appendix H Arduino Library | 300 |
| Appendix I Relay Data Subsystem Matlab Code | 301 |
| Appendix J Arduino C/C++ DAS Test Code | 303 |
| Appendix K Arduino Due adc test set-up..... | 311 |
| Appendix L DAS System Calibration Set-up..... | 312 |
| Appendix M Seven-order Lowpass Filter for 50kHz Channels | 314 |
| Appendix N 4-20mA Input, Differential and Scaling Stages Cascaded 5kHz Channels (MINUS Input Grounded) | 315 |
| Appendix O 4-20mA Input, Differential and Scaling Stages Cascaded 5kHz Channels (differential input)..... | 316 |
| Appendix P Input, Differential, Scaling and Filter Stages Cascaded 5kHz Channels | 317 |
| Appendix Q Input, Differential and Scaling Stages Cascaded 50kHz Channel with 4-20mA Input Signal (Minus input grounded)..... | 318 |
| Appendix R Input, Differential and Scaling Stages Cascaded 50kHz Channel with 0-10V Input Signal (minus input grounded)..... | 319 |
| Appendix S Seven-order Lowpass Filter 50kHz Channel DC Simulations | 320 |
| Appendix T Some of the popular microprocessors and microcontrollers | 321 |
| Appendix U Design Notes..... | 323 |
| Appendix V ADC Resolution..... | 324 |
| Appendix W 5kHz Channel Implementation | 325 |
| Appendix X 50kHz Channel Implementation | 326 |
| Appendix Y PSU Implementation..... | 327 |

List of Abbreviations and Acronyms

| | |
|------|---------------------------------------|
| AC | Alternating Current |
| ADC | Analogue-to-Digital Converter |
| ANSI | American National Standards Institute |
| AWG | Arbitrary Wave Generator |
| CCM | Continuous Charge Measurement |
| CCSU | Current Source Unit |
| CPS | Characteristic Power Spectrum |
| CSV | Comma Separated Variables |
| DAS | Data Acquisition System |
| DAQ | Data Acquisition |
| DC | Direct Current |
| DNL | Differential Nonlinearity |
| DSP | Digital Signal Processing |
| EDA | Electronic Design Automation |
| ENOB | Effective Number of Bits |
| FFT | Fast Fourier Transform |
| FIR | Finite Impulse Response |
| FPGA | Field Programmable Gate Array |
| FSR | Full-Scale Range |
| GBP | Gain-Bandwidth Product |
| GUI | Graphical User Interface |
| IC | Integrated Circuit |
| IDE | Integrated Development Environment |
| INL | Integral Nonlinearity |
| ISA | International Society of Automation |
| kSPS | Kilo Samples per Second |
| LDO | Low-Dropout |
| LED | Light-Emitting Diode |

| | |
|---------|---------------------------------------|
| LSB | Least Significant Bit |
| MIPS | Millions Instruction per Second |
| MSPS | Mega Samples per Second |
| OPAMP | Operational Amplifier |
| PCB | Printed Circuit Board |
| PC | Personal Computer |
| PCI | Peripheral Component Interconnect |
| PDS | Power Spectral Density |
| PSU | Power Supply Unit |
| RAM | Random Access Memory |
| ROM | Read-Only Memory |
| RMS | Root Mean Square |
| RTOS | Real-Time Operating System |
| S/N | Signal-to-Noise |
| SAR | Successive Approximation Register |
| SCU | Signal Condition Unit |
| SINAD | Signal-to-Noise Ratio plus Distortion |
| SNR | Signal-to-Noise Ratio |
| SoC | System-on-Chip |
| THD | Total Harmonic Distortion |
| TTL | Transistor-Transistor Logic |
| UI | User Interface |
| UNISA | University of South Africa |
| μ P | Microprocessor |
| VLSI | Very-Large-Scale Integration |

List of Tables

| | |
|--|-----|
| Table 3.1 DAS minimum requirements..... | 60 |
| Table 3.2 4-20mA receiver input stage requirements..... | 61 |
| Table 3.3 Embedded system platform comparison | 65 |
| Table 3.4 Instruments list | 69 |
| Table 3.5 Power supply and ground isolation specifications | 70 |
| Table 3.6 PSU voltage test specifications with no input signal | 71 |
| Table 3.7 PSU voltage specification with +1.5V DC input signal..... | 71 |
| Table 3.8 PSU voltage specifications with 1V AC input signal..... | 72 |
| Table 3.9 Channel DC offset voltage with input at 0V | 73 |
| Table 3.10 Channel DC offset voltage with input open (floating) | 73 |
| Table 3.11 Signal chain 1.5V DC input test..... | 74 |
| Table 3.12 AC input test frequency..... | 75 |
| Table 3.13 Current calibration points..... | 90 |
| Table 3.14 Voltage calibration points..... | 90 |
| Table 4.1 Typical mill sensor bandwidth | 94 |
| Table 5.1 Lowpass filter (5kHz channel) requirements | 135 |
| Table 5.2 Lowpass filter (50kHz channel) requirements | 137 |
| Table 5.3 Resistor and capacitor values for seven-order lowpass filter | 138 |
| Table 5.4 Passive components tolerances..... | 139 |
| Table 5.5 Op-amp DC parameters | 140 |
| Table 5.6 Op-amp AC parameters | 141 |
| Table 5.7 TL084BC design parameters..... | 142 |
| Table 5.8 THS4032 design parameters | 143 |
| Table 5.9 SAR ADC minimum requirements | 168 |
| Table 5.10 SAR ADC DC errors requirements | 169 |
| Table 5.11 Digital interface requirements | 171 |
| Table 6.1 Character “Cx” interpretation..... | 195 |
| Table 7.1 Power supply and ground isolation results..... | 203 |

| | |
|---|-----|
| Table 7.2 PSU voltage measurement results with no input signal | 203 |
| Table 7.3 PSU voltage measurement results with +1.5V DC input signal..... | 204 |
| Table 7.4 PSU voltage measurement results with +1.5V AC input signal..... | 204 |
| Table 7.5 DC offset voltages with input at 0V | 205 |
| Table 7.6 Channel DC offset voltage with input open (floating) | 206 |
| Table 7.7 Signal chain DC input results | 208 |
| Table 7.8 Current loop calibration set points step-up measurements | 231 |
| Table 7.9 Voltage loop calibration set points step-up measurements | 238 |

List of Figures

| | |
|--|----|
| Figure 2.1 Inductive proximity probe's static response curves for a 30mm steel ball at various distances away from the probe's centre..... | 8 |
| Figure 2.2 Proximity probe signals with respect to mill rotation | 9 |
| Figure 2.3 Conductivity probe bolts | 10 |
| Figure 2.4 Conductivity probe signal indicating centrifuging..... | 12 |
| Figure 2.5 Conductivity probe signal | 12 |
| Figure 2.6 Pilot Mill with strain gauge sensor installed on the lifter bar | 14 |
| Figure 2.7 Sensor represented as a Black Box | 25 |
| Figure 2.8 Sensor with 4-20ma current loop interface | 39 |
| Figure 2.9 Simplified DAS..... | 41 |
| Figure 3.1 Typical μ P or μ C SoC block diagram..... | 62 |
| Figure 3.2 Model-based development | 64 |
| Figure 3.3 Typical host platform | 68 |
| Figure 3.4 DAS controller model test set-up..... | 76 |
| Figure 3.5 DAS controller model | 77 |
| Figure 3.6 Data message structure..... | 79 |
| Figure 3.7 Serial port interface..... | 80 |
| Figure 3.8 Data acquisition subsystem | 81 |
| Figure 3.9 ADC channels subsystem..... | 82 |
| Figure 3.10 Arduino® Due hardware platform ADC channels..... | 83 |
| Figure 3.11 System Alive interface | 83 |
| Figure 3.12 Control subsystem model..... | 84 |
| Figure 3.13 DAS user interface components layout..... | 85 |
| Figure 3.14 Start of data acquisition process command message structure | 86 |
| Figure 3.15 Data message structure..... | 87 |
| Figure 3.16 DAS system calibration set-up..... | 89 |
| Figure 4.1 System level DAS diagram | 92 |
| Figure 4.2 Sensor with 4-20mA transmitter model | 96 |

| | |
|---|-----|
| Figure 4.3 Simulink model for sensor 4-20mA transmitter | 97 |
| Figure 4.4 Simulation set-up for 4-20mA transmitter model | 98 |
| Figure 4.5 4-20mA transmitter output..... | 99 |
| Figure 4.6 Output current (Y) vs. Input signal (X)..... | 100 |
| Figure 4.7 Erroneous output current..... | 100 |
| Figure 4.8 Simplified 4-20mA receiver diagram | 101 |
| Figure 4.9 Simulink model for 4-20mA receiver | 103 |
| Figure 4.10 Simulation set-up for 4-20mA receiver model..... | 104 |
| Figure 4.11 4-20mA receiver output signal..... | 105 |
| Figure 4.12 Input current (X Axis) vs. Output voltage (Y Axis) | 105 |
| Figure 4.13 Single-pole lowpass filter..... | 106 |
| Figure 4.14 Lowpass filter transfer function | 107 |
| Figure 4.15 Single-pole lowpass filter simulation..... | 108 |
| Figure 4.16 Single-pole filter output at 50kHz (cut-off frequency) | 108 |
| Figure 4.17 Single-pole filter output at 400kHz..... | 109 |
| Figure 4.18 Fourth-order lowpass filter simulation..... | 110 |
| Figure 4.19 Fourth-pole filter output at 50kHz (cut-off frequency)..... | 111 |
| Figure 4.20 Fourth-pole filter output at 100kHz | 111 |
| Figure 4.21 Ideal sampler and hold model | 113 |
| Figure 4.22 Simulink zero-order hold model | 113 |
| Figure 4.23 Zero-order hold simulation set-up..... | 114 |
| Figure 4.24 Zero-order hold output signal at 10Hz and 100Hz sampling frequency | 114 |
| Figure 4.25 Idealised ADC quantiser simulation set-up | 115 |
| Figure 4.26 Simulation results for quantisation interval 0.1 and 0.01..... | 116 |
| Figure 4.27 Simulation results for quantisation interval 0.001 | 116 |
| Figure 4.28 Zero-order hold and idealised ADC quantiser in series simulation set-up | 117 |
| Figure 4.29 System-level ADC model simulation set-up..... | 118 |
| Figure 4.30 Data acquisition controller model | 121 |
| Figure 4.31 Data processing controller | 123 |
| Figure 4.32 Data displaying and storage controller..... | 124 |
| Figure 5.1 4-20mA signal receiver | 128 |

| | |
|--|-----|
| Figure 5.2 Input buffer stage | 129 |
| Figure 5.3 Differential amplifier stage | 131 |
| Figure 5.4 Signal-scaling stage..... | 133 |
| Figure 5.5 Active lowpass filter for 5kHz channels | 134 |
| Figure 5.6 Active two-order lowpass filter 50kHz channels | 138 |
| Figure 5.7 4-20mA receiver and buffer stage with poor DC performance | 146 |
| Figure 5.8 4-20mA receiver and buffer stage with good DC performance | 146 |
| Figure 5.9 Input and output DC transfer characteristics for 4-20mA signal | 147 |
| Figure 5.10 0-10V Receiver and buffer stage DC performance (5kHz channel) | 147 |
| Figure 5.11 Input and output DC transfer characteristics for 0-10V signal (5kHz channel) . | 148 |
| Figure 5.12 Signal receiver and buffer stages AC response (5kHz channel)..... | 149 |
| Figure 5.13 Input and output DC transfer characteristics for 4-20mA signal scaled by 0.5 (5kHz channel) | 150 |
| Figure 5.14 Input and output DC transfer characteristics for 0-10V signal scaled by 0.25 (5kHz channel) | 150 |
| Figure 5.15 DC transfer characteristics for 4-20mA signal scaled by 0.5 (5kHz channel) with lowpass filter stage | 151 |
| Figure 5.16 DC transfer characteristics for 0-10V signal scaled by 0.25 (5kHz channel) with lowpass filter stage | 152 |
| Figure 5.17 AC transfer characteristics lowpass single-order filter (5kHz channel) | 153 |
| Figure 5.18 AC transfer characteristics lowpass single-order filter -3dB point (5kHz channel) | 153 |
| Figure 5.19 Overall AC transfer characteristics (5kHz channel) | 154 |
| Figure 5.20 4-20mA receiver and buffer 50kHz channel (with 4mA input signal) | 155 |
| Figure 5.21 0-10V receiver and buffer 50kHz channel (with 1V input signal)..... | 156 |
| Figure 5.22 0-10V Receiver and buffer stage 50kHz channel (with 10mV input signal)..... | 156 |
| Figure 5.23 Input and output DC transfer characteristics for 4-20mA signal (50kHz channel) receiver and buffer stage..... | 157 |
| Figure 5.24 Input and output DC transfer characteristics for 0-10V signal (50kHz channel) receiver and buffer stage..... | 157 |
| Figure 5.25 DC transfer characteristics for 4-20mA signal scaled by 0.5 (50kHz channel).. | 159 |

| | |
|---|-----|
| Figure 5.26 DC transfer characteristics for 0-10V signal scaled by 0.25 (50kHz channel) ... | 159 |
| Figure 5.27 Signal receiver, buffer and scaling stages AC performance (50kHz channel) ... | 160 |
| Figure 5.28 DC transfer characteristics for 4-20mA signal scaled by 0.5 (50kHz channel) with filter stage | 161 |
| Figure 5.29 DC transfer characteristics for 0-10V signal scaled by 0.25 (50kHz channel) with filter stage | 161 |
| Figure 5.30 AC transfer characteristics lowpass seven-order filter (50kHz channel)..... | 162 |
| Figure 5.31 AC transfer characteristics lowpass seven-order filter -3dB point (50kHz channel) | 162 |
| Figure 5.32 Overall AC transfer characteristics (50kHz channel) 0-10V signal..... | 163 |
| Figure 5.33 Overall AC transfer characteristics (50kHz channel) 4-20mA signal | 163 |
| Figure 5.34 SAR input model..... | 167 |
| Figure 5.35 Data acquisition system firmware data flow diagram..... | 176 |
| Figure 5.36 Host platform software data flow diagram..... | 182 |
| Figure 6.1 Bypassing capacitors..... | 185 |
| Figure 6.2 PCB layer stack..... | 186 |
| Figure 6.3 Grounding scheme | 186 |
| Figure 6.4 Analogue input subsystem | 189 |
| Figure 6.5 Data formatting subsystem..... | 191 |
| Figure 6.6 Data message structure..... | 192 |
| Figure 6.7 Data transmission subsystem | 193 |
| Figure 6.8 Serial port configuration | 193 |
| Figure 6.9 Start of data acquisition process command message structure | 195 |
| Figure 6.10 Stop of data acquisition process command message structure..... | 196 |
| Figure 6.11 System Alive subsystem model | 197 |
| Figure 6.12 DAS user interface components layout..... | 200 |
| Figure 6.13 DAS user GUI running | 201 |
| Figure 7.1 0-10V channels input stage | 207 |
| Figure 7.2 4-20mA outputs at 5Hz input signal (5kHz channel)..... | 210 |
| Figure 7.3 4-20mA channel output at 50kHz input signal (5kHz channel)..... | 211 |
| Figure 7.4 0-10V channel output at 100kHz input signal (50kHz channel)..... | 211 |

| | |
|--|-----|
| Figure 7.5 4-20mA channel output with common mode signal input | 212 |
| Figure 7.6 DAS GUI real-time plots channel 1-6..... | 213 |
| Figure 7.7 DAS GUI real-time plots channel 7-12..... | 213 |
| Figure 7.8 Saved data file | 214 |
| Figure 7.9 +1.5V input ADC measurements | 216 |
| Figure 7.10 PC scope 1.5V measurement..... | 217 |
| Figure 7.11 +1.5V ADC measurements comparison between 3.3V and 3.287V reference voltage | 218 |
| Figure 7.12 +3.3V input ADC measurements | 220 |
| Figure 7.13 250mV 1Hz signal test results | 221 |
| Figure 7.14 Test signal 250mVpeak 1Hz sine wave | 222 |
| Figure 7.15 250mVpeak 5Hz signal test results (running Arduino Simulink® model)..... | 222 |
| Figure 7.16 Channel 1 and 2 250mV 5Hz signal test results (running Arduino Simulink® model)..... | 223 |
| Figure 7.17 Channel 11 and 12 250mV 5Hz signal test results (running Arduino Simulink® model)..... | 224 |
| Figure 7.18 Channel 1 250mV 2Hz signal test results (running Arduino Simulink® model)..... | 225 |
| Figure 7.19 System Alive output frequency 1.1kHz no data streaming (running Arduino Simulink® model)..... | 226 |
| Figure 7.20 System Alive output frequency 8.2Hz data streaming (running Arduino Simulink® model)..... | 227 |
| Figure 7.21 System Alive output frequency 96kHz no data streaming (running Arduino C++ code) | 228 |
| Figure 7.22 System Alive output frequency 432Hz data streaming (running Arduino C++ code) | 229 |
| Figure 7.23 Channel 1 and 2 250mV 2Hz signal test result (running Arduino C++ code).... | 229 |
| Figure 7.24 Channel 1 250mV 5Hz signal test result (running Arduino C++ code)..... | 230 |
| Figure 7.25 Channel 1 transfer function..... | 232 |
| Figure 7.26 Channel 2 transfer function..... | 233 |
| Figure 7.27 Channel 11 transfer function..... | 233 |
| Figure 7.28 Channel 1 absolute, relative error and accuracy | 235 |

| | |
|--|-----|
| Figure 7.29 Channel 2 absolute, relative error and accuracy | 236 |
| Figure 7.30 Channel 11 absolute, relative error and accuracy | 237 |
| Figure 7.31 Channel 8 transfer function..... | 239 |
| Figure 7.32 Channel 9 transfer function..... | 239 |
| Figure 7.33 Channel 12 transfer function..... | 240 |
| Figure 7.34 Channel 8 absolute, relative error and accuracy | 241 |
| Figure 7.35 Channel 9 absolute, relative error and accuracy | 242 |
| Figure 7.36 Channel 12 absolute, relative error and accuracy | 243 |

Chapter 1 Introduction

1.1 Background

The harsh environments in milling devices make it difficult to directly measure inside ball mills (Alatalo, 2011). Therefore, electronic and signal processing techniques are required to achieve accurate measurements. Mineral processing researchers have relied on advanced sensor systems to collect in-mill data. From this data, they were able to derive models to monitor and characterise changes in operating conditions (Tano, 2005). Different approaches are used for the study of milling, such as indirect measurements (Alatalo, 2011).

Other approach used is to measure the acoustic signal generated by the milling process. In this approach, mill characteristics are collected as vibration signals for further inferential analysis. The power drawn (current ripples generated) by the mill is also another approach used to characterize in-mill data (Alatalo, 2011). In all these approaches, mill characteristics are collected as electric signals for further inferential analysis.

All these signals require a data acquisition system (DAS) to accurately collect, process and store data for later analysis. Some of the parameters measured on mills include flow rate, density, viscosity, torque, mill speed, load mass, conductivity, force and mill vibration (Smit, 2000). These parameters are collected using sensors placed in strategic positions around the mill.

Generally, DAS systems are expensive and normally not adaptable, and they are tailor-made for specific input signals. If a change is required, then a new DAS system that meets the requirements must be purchased. In practice, it is usually very hard to find DAS systems that meet all user requirements, especially if there are special needs to be catered for. Moreover, this off-shelf DAS systems always provide post-processed data (i.e. digital signal processor (DSP)) (Felinger et al., 2015) to the user. This has the potential of filtering out data that may be useful to a mineral processing researcher.

There are numerous data acquisition systems in the electronic and scientific industry that have diverse functions and forms. They differ in terms of the type of measurement, characteristics of signals, level of accuracy required, and environment of application. They also differ in terms of complexity and implementation. DAS systems are classified into three categories: computer-based, embedded microcontroller-based, and field-programmable gate array (FPGA) based data systems. The PC-based system is generally the most expensive of the three categories, as it requires a data acquisition (DAQ) card (Ferrero Martín et al., 2014) and is less configurable and non-mobile.

Embedded microcontroller and FPGA platforms are now developed to a level whereby the entire DAS can be implemented in such platforms. These platforms generally cost less than the DAQ card. Moreover, they are easily reconfigurable if requirements change, with the FPGA platform being the more reconfigurable of the two platforms. A DAS system consists of the following subsystems:

- Sensors and Transducers
- Interface (wires, wireless, and optical)
- Signal Conditioning
- Data acquisition hardware (analogue-to-digital converter (ADC) and DAC)
- PC and Data Acquisition software

1.2 Problem Statement

For decades, mineral processing researchers have relied on advanced sensor systems to collect in-mill data. These mill characteristics are collected as electric signals for further inferential analysis. This is done with a DAS. The data acquisition system thus plays a pivotal role in any scientific research. Therefore, electronic and signal processing techniques are required to collect and achieve accurate measurements. The current DAS used in data collection has different interfaces due to different sensor interface requirements (Smit, 2000). This may affect signal quality, depending on the signal transmission method used in sending data to the DAS.

With the need to advance the study of mineral processing operations and specifically ball milling, the Department of Electrical and Mining Engineering at University of South Africa (UNISA) required a DAS. The system will be used to collect signals from sensors embedded in a pilot mill. All these signals require DAS to accurately collect and store data for later analyses. Therefore, this initiated the need to research and develop a data acquisition system. The system will collect signals generated by sensors transmitting data using 4-20mA or 0-10V protocol. This DAS will be based on an embedded microcontroller platform, with firmware developed using model-based design.

1.3 Research Aim and Objectives

The main objective of this research is to develop and design a data acquisition system that measures, processes, stores and displays raw signals collected from sensors and transducers on a pilot mill. The sensors are required to transmit these signals using a 4-20mA or 0-10V protocol. This will assist a mineral processing researcher to use different types of sensors and transducers and, therefore, experiment using different measuring techniques without the need to change a data acquisition system.

The secondary objective is to develop an inexpensive DAS system, using open-source hardware platform. This platform will then be used to implement the data acquisition hardware.

The individual research aims to successfully complete the research project are as follows:

- Use of MATLAB® and Simulink® as a software development environment:

Using Matlab® programming language to develop the GUI and software.

- Research latest methods in electronic system design focusing on a model-based design: MATLAB® and Simulink® used as a development tool to design, model and implement system firmware. This will include using hardware-in-loop model-based design techniques.

- Research latest and appropriate open-source (Dunn, 2014; Harnett, 2011; Russell et al., 2012) hardware platforms and implement a data acquisition solution using such platform.
- Design and implement as signal conditioning card:

The appropriate circuit design techniques and technologies are researched and implemented.

The goal is to design a highly accurate analogue signal conditioning card.

1.4 Relevance of this Research

The research aim is to develop a measuring instrument specifically designed for measuring and collecting data on a pilot mill. The mineral processing researcher will use this to measure signals and collect data that will make it possible to characterise the pilot mill behaviour. This research will, therefore, contribute to the development of a better understanding of ball mills in general. The study will further aid in the development of improved mineral processing techniques and better mill yields. The research will also aid in the contribution of knowledge production in the use and implementation of open-source platforms in high-end system applications, with firmware developed using model-based software development. Using model based design will assist in making DAS system reconfigurable and adaptable. As the core functions of the system will be based on the models, which can be changed and adapted for specific requirements.

1.5 Research Approach

An extensive study on relevant literature was conducted. The outcomes of the study were implemented when designing the system. The system was developed using relevant engineering and system engineering techniques to arrive at a solution. This included developing the interface card used to interface the sensors to the data acquisition hardware (i.e. ADC). This card performs all the analogue signal pre-processing tasks (i.e. conversions, filtering and scaling). The open-source hardware platform was chosen based on the high-end data acquisitions requirements. These included a high ADC sampling rate, high ADC resolution, adequate ADC channels, high processing power, adequate communication peripherals, adequate memory (random access memory (RAM) and read-only memory (ROM)) and low cost. The availability

of open-source development tools formed one of the criteria. These tools support Matlab® Simulink® model-based design. The DAS system firmware was developed using a model-based design in a Matlab® Simulink® environment. The system algorithm is built with Simulink® library blocks, and libraries provided for an open-source hardware platform. A Simulink® environment generates code automatically from the model. This code is deployed into the open-source hardware platform. The model is debugged and verified using the external mode feature of the Simulink® development environment. Signals are monitored and verified in real time using this mode. A graphical user interface (GUI) application is developed in a Matlab® programming environment and runs on the host system. The GUI is used to collect and store data streamed by the DAS.

1.6 Dissertation Outline

This dissertation is organised into eight chapters. Chapter 1 presents the research problem and relevance of the research. Chapter 2 presents the theoretical background of ball milling, load behaviour measuring techniques, DAS and sensors characteristics and limitations, and DAS subsystems. In Chapter 3, research methodology, design approach, testing, and calibration methods are presented. Chapter 4 describes the theory of DAS system components, and their models are presented. Chapter 5 details the DAS firmware, hardware and software requirements. Chapter 6 provides the DAS firmware, hardware and software implementation details, based on system requirements in Chapter 5. Chapter 7 provides experimental results and system limitations. In Chapter 8, the project conclusion and recommendations are presented.

Chapter 2 Literature Review

2.1 Introduction

Data acquisition systems have been used for decades in scientific research. Their development and design have been greatly researched. There are numerous systems that are designed for specific use and for generic use (i.e. LabView® DAQ cards). Application of DAS systems is widespread; they are used in almost all industries, such as in academic research, biomedical, telecommunications, manufacturing, home appliance, and power systems. Therefore, it is critical to understand their theory, design and limitations. This chapter starts by providing background literature on mill behaviour research and measurement techniques used to collect mill parameters, and then it discusses the theory of the DAS components, their design, limitation and technology.

2.2 Background of Mill Measurements

There are numerous studies carried out to characterise ball mill behaviour. Different measurement techniques have been used to this end. These techniques can be categorised into two types: on-shell (invasive) and off-shell (non-invasive) methods. The mill load and mill charge behaviour, under certain mill conditions, is measured using different kinds of sensors. There is a great deal of research that has been conducted using sensors such as conductivity, inductivity, strain, vibration and acoustic sensors to measure mill behaviour. The measurements are then correlated with mill parameters of interest to the researcher. It is essential to use a method that can provide precise, accurate and real-time information of the charge behaviour to the data acquisition system.

Because of the harsh conditions inside mills, it is a difficult task to directly measure mill parameters. The indirect measurement methods used by some mineral processing researchers include measurement of load behaviour of the charge inside mills using sensors such as acoustic sensors, load cells, power and bearing pressure measurement. The direct measurements or on-shell measurement methods used include measurement of mill

parameters using mostly inductive proximity probes, conductive probes, vibration and strain sensors. Tano (2005) claims that the sensors used for on-shell measurements are not well developed, and due to electro-mechanical issues originating from harsh demands of mill operations, these sensors do not last and suffer from drift. However, with new developments in sensor technology, this may not be valid anymore (i.e. military and industrial grade sensors).

2.2.1 Inductive Proximity Sensor

The first comprehensive study that used an inductive proximity sensor to measure load behaviour in a dry pilot mill was conducted by Kiangi and Moys (2006). The authors performed experiments that linked the behaviour of the mill with the signal picked up by the inductive proximity probe. The inductive proximity probe is an electronic device designed to detect the presence of a metallic object within its sensing surface area. The probe uses electrical inductance principles in sensing metallic objects. It consists of a coil of wire wound around a ferrite core, an oscillator, a detection and output circuit (Kiangi and Moys, 2006). The probe produces an electrical signal whenever a steel ball moves into its detection area.

Figure 2.1 shows the results of the test performed with the probe to determine its suitability. It shows the output generated by the probe with respect to the distance of the steel ball from the probe-measuring surface. Another test performed on the probe was the dynamic response of the sensor. Kiangi and Moys (2006) found that the time it takes for the signal to rise to 98% of its final value was 6.1ms, which is the rise time of the output signal. The inductive proximity output signal was showed to correlate with the load orientation (charge media) within the mill as a function of operating conditions. Their research does not mention any technical specification (i.e. resolution, span, or bandwidth) of the probe. The information is critical in choosing and designing the right data acquiring system.

Recently, Mulenga and Moys (2013) followed with research using an inductive proximity probe to measure the charge media position, in a tumbling pilot mill. They used the inductive proximity probe to measure the position of the load on a wet mill. Using the inductive

proximity sensor, they were able to measure the angular position of the media charge. The signal generated by the probe was transmitted to a computer via slip ring. The computer used the Waveview© application to collect real-time signals generated by the probe. Waveview© is a program from Eagle Technology (Pty) Ltd developed to view signal measured by DAQ cards and modules. The program also saves the signals in text format file and can be exported into any spreadsheet for analysis. Figure 2.2 depicts the output of the inductive proximity probe. According to Mulenga and Moys (2013), when the probe signal drops significantly to about 4V, it indicates the entry of the sensor into the media charge between 120° to 300°. From 300°, the signal returns to 10V, indicating that the sensor is leaving the media charge. The authors fail to mention in detail the transmission method used in transmitting the sensor's signal to the data acquisition system. The signal can be degraded if transmitted with an inappropriate method, and this will negatively affect the results.

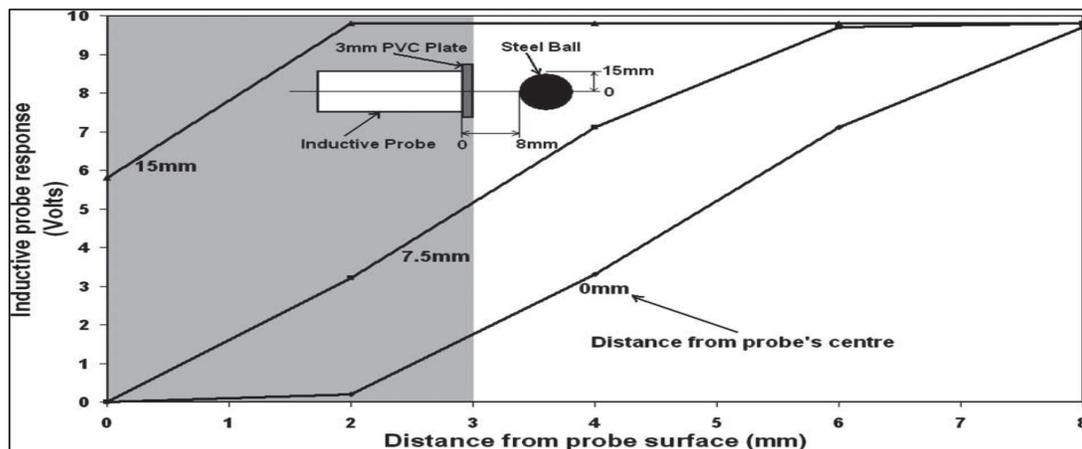


Figure 2.1 Inductive proximity probe's static response curves for a 30mm steel ball at various distances away from the probe's centre

Source: Kiangi and Moys (2006)

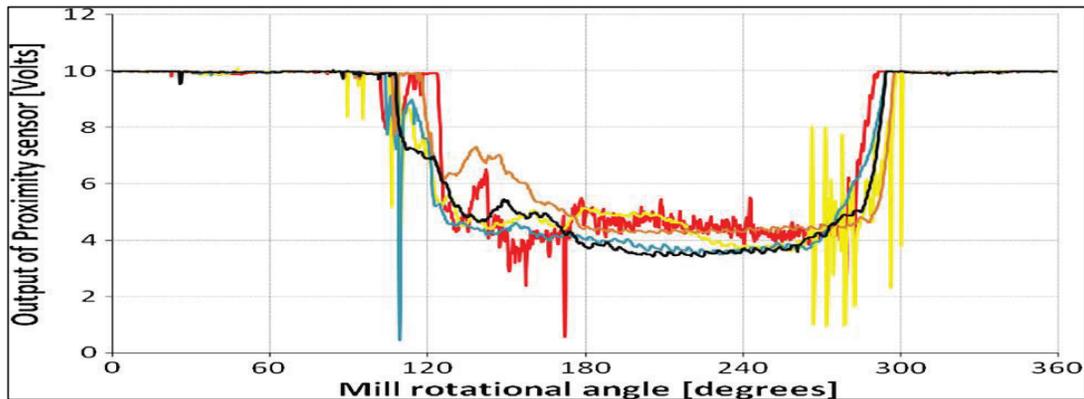


Figure 2.2 Proximity probe signals with respect to mill rotation

Source: Mulenga and Moys (2013)

2.2.2 Conductivity Sensor

Conductivity probe is one of the sensors used by several researchers in measuring and characterising the load behaviour of a milling process. This sensor was first used by Lanstiaik (1973) in measuring the volume of the load in a rotary mill. Moys (1984) used conductivity probe to measure parameters characterising the dynamic behaviour of the load in grinding mill. Then Vermeulen et al. (1984) used the conductive probe to measure the physical information inside a rotary mill. They used a bolt that was fitted with an electrically insulated electrode. The insulation provides electrical isolation between the mill liner and probe. The probe produces a signal when the conductive electrode is in contact with the mill load (charge media). This is possible because the charge media completes an electrical circuit between the probe and mill liner (Liddell and Moys, 1988). These bolts are used to clamp the liner blocks and lifter bars to mill shell.

Montini and Moys (1988) also used a mill liner bolt, as a conductive electrode probe, and it was mounted on the mill shell and electrically insulated from the mill shell by insulating washers. The probe was connected to an electronic circuit, which excited the probe with a 5 000Hz signal with a peak-to-peak amplitude of 8V. The current will flow from the probe to the mill shell through a slurry and the ball surrounding the probe. A signal will be generated

and is filtered to provide a DC signal, which is a function of sensed conductivity. The result showed that a reliable estimation of the percentage level of solids in the slurry inside a mill can be obtained from the probe signal. The conductive probe-generated signals have low voltage levels which can easily be contaminated by electrical noise and affect measurement results. The signal transmission method was not mentioned in this research, which can also influence the measurement.

Figure 2.3 presents the conductivity probe used.

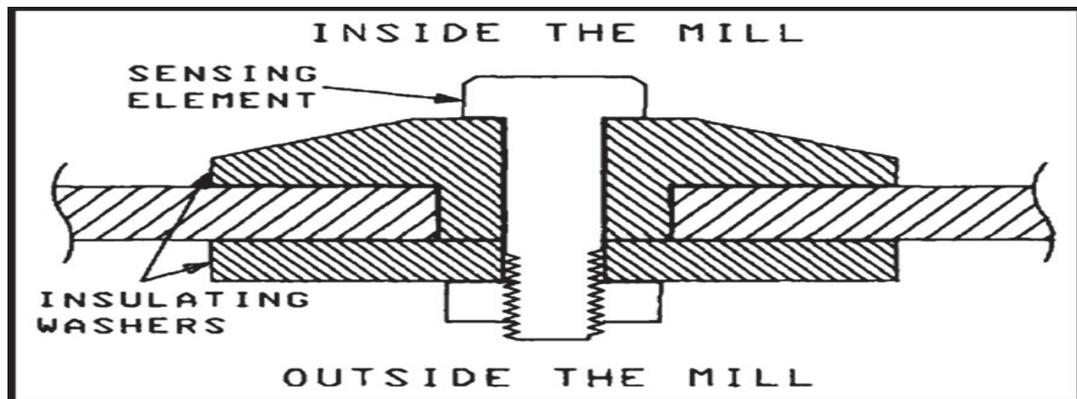


Figure 2.3 Conductivity probe bolts

Source: Montini and Moys (1988)

Moys et al. (1996b) used a similar liner bolt instrumented as a conductive probe to measure the orientation of the load and slurry pool passing through the mill. The signal generated by the probe was amplified and transmitted via a slipring to a personal computer (PC) which measured and recorded the signals. The PC data acquisition collected two types of data. The first type of data collected was on a continuous basis, while the second type was instantaneous readings of probes recorded as a function of time. The data recorded showed dramatic changes as conductivity probe enters and leaves the load during mill rotation. The load toe and shoulder position can be obtained from this collected data (Moys et al., 1996).

Again, in 1998 Van Nierop and Moys (1998) used conductive probes mounted on mill liner bolts to directly measure the load behaviour. They combined this measurement with movement sensors data. Their research focus was to measure the load behaviour of an industrial mill. They used four conductivity probes which were placed at 90° intervals around the circumference at the centre of the mill (Van Nierop and Moys, 1998). Signals generated by the probes were transmitted off the mill using telemetry, and a PC was used for data collection and storage. Their results showed that the collected data clearly indicates the toe position of the load, and when the probe goes under the load, the conductive signal increases significantly. Further, the angular shoulder position is indicated when the probe comes out of the load and the conductive signals decrease significantly. They concluded that the noise in the conductive signals collected just before the toe position signal may be due to cataracting¹ or crushing material impacting on the mill shell (Van Nierop and Moys, 1998). They also found that the conductivity probes data clearly indicates presence of centrifuging². This is indicated by an unchanging conductivity probe signal (Figure 2.4).

In 2013, Mulenga and Moys (2013) used conductivity probe embedded in a mill liner bolt to measure the mill behaviour of a tumbling ball mill. The conductivity probe was used to measure the position of the pulp. The probe detects the changes in the conductive liquids. Figure 2.3 presents the conductivity sensor used in this study. The bolt was used as the sensing element. The conducting liquid will produce a drop in resistance between the probe and the mill shell. This drop in resistance can be interpreted as the presence of slurry. They discovered that the probe can also reveal some relationship between media charge and slurry and also toe and shoulder angular positions of the slurry. The output of the conductivity probe in relation to the mill rotational angle or angular position is shown in Figure 2.5.

¹ Crashing balls thrown into the liners (Moys and Skorupa, 1992)

² A process by which solid particles are separated from the liquid by centrifugal forces (Armenante, n.d.)

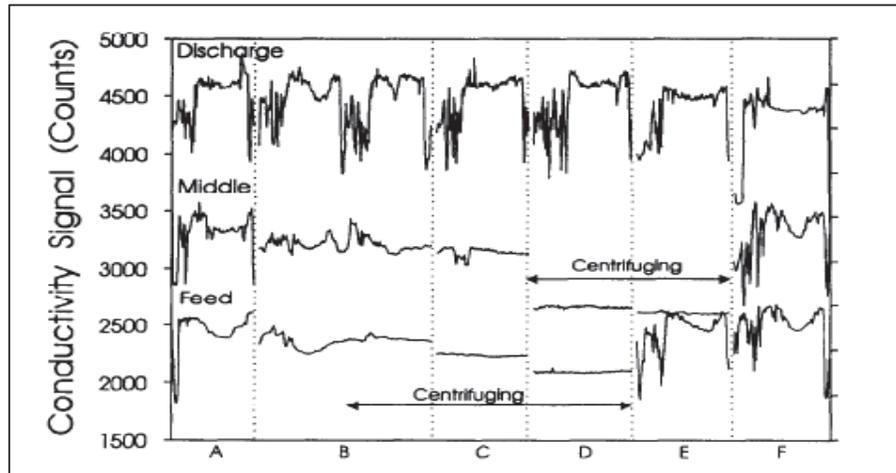


Figure 2.4 Conductivity probe signal indicating centrifuging

Source: Van Nierop and Moys (1998)

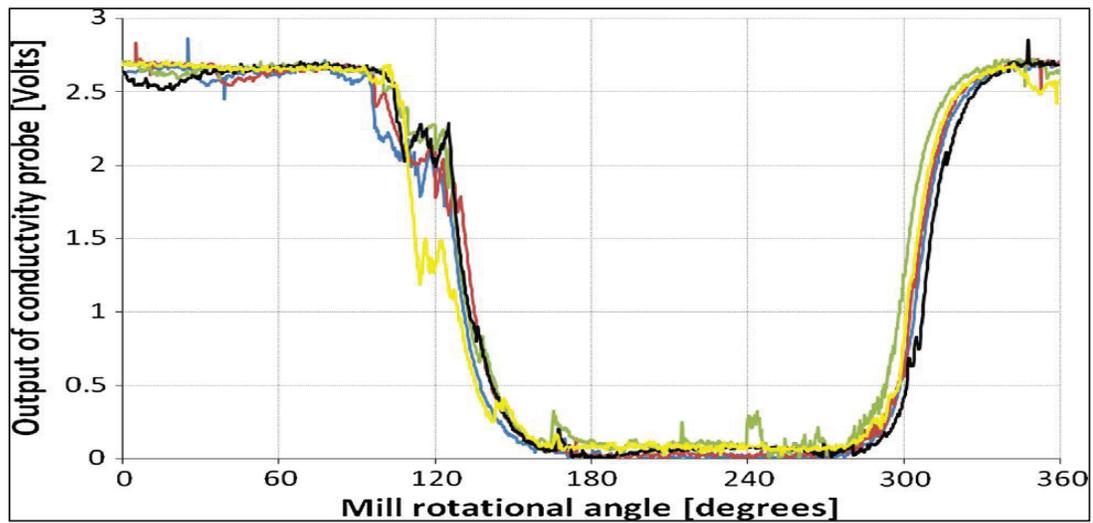


Figure 2.5 Conductivity probe signal

Source: Mulenga and Moys (2013)

Alternative Measuring Techniques

Tano (2005) reported on a method used in plants that measure the bearing back pressure on the mill feed and discharge end. This method gives an idea of the weight of the charge and can be correlated to the filling level. Another method developed by Branken Mineral using measurement of bearing pressure, power drawn and mill speed employed these parameters to calculate the weight of the charge load. This method showed that it was theoretically feasible to relate the mill load with bearing pressure, provided that wear on mill lining and drive system is taken into account. The author concluded that the greatest disadvantage of this method is the unstable pressure and temperature shift that causes sensor drift.

Another off-shell method uses acoustic noise generated by the mill to measure mill load behaviour. Different set-ups were used in measuring the sound emitted by the mill. This include using a single microphone to relate sound power with pulp viscosity (Watson, 1985) and arrays of microphones to detect changes in the position of the toe of the mill charge (Jaspan et al., 1986). This method's biggest disadvantage was that the background noise was also picked by the microphones. Additionally, to analyse the signals, a complex signal processing techniques will be required to extract useful information from the background noise.

Sen and Bhaumik (2013) used an acoustic signal as a control signal to regulate operation of the grinding mill. They found that sound changes can be used to detect changes on product sizes and undesirable conditions when they occur. They claim that no techniques or scheme exists or has been developed that can *“regulate the milling parameters when output deviates from desired particle size range (product) in real-time mode”*(Sen and Bhaumik, 2013:262). Their study focused mainly on acquiring, processing and analysing acoustic signals in real time.

An on-shell method using strain gauge was developed by Moys (1996). It uses strain gauge to measure the forces exerted by the load on the mill liner (Moys et al., 1996a). The authors used two strain gauges to form an active part of a Wheatstone bridge, which generated a signal

proportional to bridge deformation.

Kolacz (1997) used strain transducers to measure the filling level of a mill. The strain generated is due to the deformation of the mill shell as it is filled with balls and material. The rotation of the mill also exerts forces on the mill, which causes shell deformation. He asserts that the shell deformation is directly proportional to the mill load. The strain transducers must have a low detection threshold because of the low strain levels generated by the mill shell, overload protection, and good stability and reliability. Kolacz (1998) concluded that this method can be used to control the mill charge behaviour. Because of the limitation of the piezoelectric strain transducers, they can only be used for experimental measurements, with limited duration.

Tano (2005) used a Continuous Charge Measurement (CCM) system to study mill load behaviour. The system continuously measures the charge volume and the angle of repose. The system uses strain gauge sensors embedded in the lifter bars on the mill lining. The sensor generated signals due to deflecting of the lifter bars that are immersed in the charge as the mill rotates. The generated signal is amplified, processed and transmitted to the data acquisition computer via telemetry.

Figure 2.6 shows the cross-section of the lifter bar with strain gauge installed on the pilot mill.

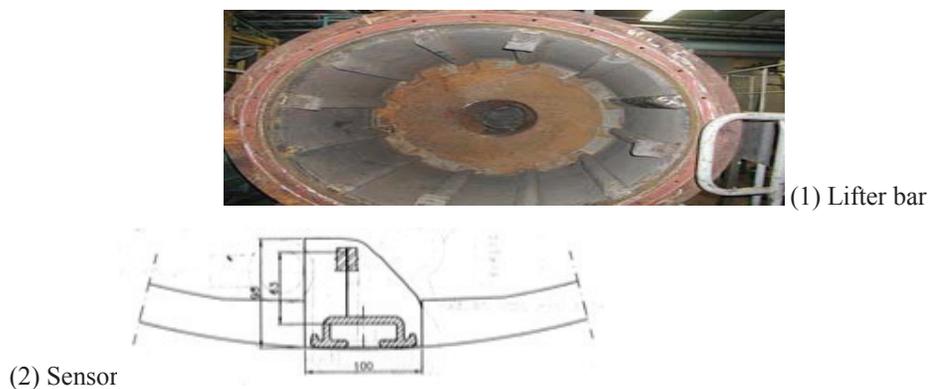


Figure 2.6 Pilot Mill with strain gauge sensor installed on the lifter bar

Source: Tano (2005)

Zeng and Forssberg (1992) used an accelerometer to measure vibration and noise emission generated by the grinding process. The grinding process generates intense mechanical vibrations and strong noise emissions. Digital signal processing techniques (i.e. FFT³) and analysis were used to obtain some form of relationship between the vibration signals and operating parameters. They focused mainly on analysing the vibration signals in the time domain and interpreted vibration data to relate the changes in vibration signatures to grinding parameters. The relative amplitudes of these sine waves with different frequencies contain information directly related to the operating state of grinding (Zeng and Forssberg, 1994).

The study of Behera et al. (2007) focused on examining and correlating the vibration signatures recorded from the laboratory scale ball mill to charge dynamics and milling characteristics. They recorded and analysed vibrating signatures of the mill in motion. They used Fast Fourier Transform (FFT) techniques to transform recorded time domain signals into frequency domain (Behera et al., 2007). The mill vibration was measured using an accelerometer connected to a data acquisition card. A LabView© application was used to control the rate of data acquisition and to display data continuously. The authors concluded that variations in vibration signatures of the mill can be related to specific mill speeds and ball load parameters. They also claim that vibration signatures can be used to identify mill overload conditions.

Si et al. (2009) followed up with a study investigating the load behaviour of an industrial scale tumbling mill using noise and vibration signature techniques, building from Behera et al. (2007) work. Their focus was to study the effects of mill load to mill noise and vibration signals. Data was collected from an industrial mill under normal practical working conditions. The data collected was analysed, and its characteristic power spectrum (CPS) was constructed by removing infective frequency components which have a small relation to mill load behaviour (Si et al., 2009). The authors also carried out an investigation on centroid frequency and frequency domain variance, which they compared with mill noise and vibration. They

³ Fast Fourier Transform an algorithm to calculate discrete Fourier transform

designed a data acquisition system (DAS) specifically used to collect mill noise and vibration data. The DAS contained the following subsystems: microphone, accelerometer, constant-current source unit (CCSU), signal condition unit (SCU), a data acquisition card, and user interface software. They concluded that noise and vibration signals can be used to monitor the behaviour of a tumbling mill.

Recently, Tang et al. (2010) investigated the behaviour of a wet mill load based on the vibration signals of a laboratory scale ball mill shell. They studied the vibration characteristics of different grinding conditions such as dry grinding, wet and water grinding, and showed that the rheological properties of the pulp affect the amplitude and frequency of the vibration signals. A data acquisition system was developed, similar to one used by Si et al. (2009). The authors used a different accelerometer sensor that had a better bandwidth (0-22 000Hz). The difference of power spectral density (PDS) of shell vibration signals under different grinding conditions was used to develop a new parameter of ball mill load called charge volume ratio, which the authors claim accurately shows total volumetric filling of ball load, material load and water load in the wet mill (Tang et al., 2010). The collected data was saved on a personal computer and analysed with Matlab®. They observed that the grinding process produces strong vibrations and acoustic signals that are stable and periodic in nature over a given time interval.

This project aims to develop a data acquisition system which is able to accurately collect data from multiple sensors (up to 16) simultaneously, provided that the sensor transmits electrical signals using the industry standard 4-20mA protocol. This protocol has a very high noise immunity compared to other transmission methods used by some of the authors. It should also support a 0-10V protocol.

2.3 Sensors and Transducers Technology

Sensors and transducers are an essential part of any data acquisition system. They are always on the front end of any measurement and acquisition system (McGhee et al., 1999). Sclater (1999:17) defines a sensor as a *“device capable of sensing changes in magnitude of some*

physical or chemical variable such as frequency, radiation, heat, pressure, or salinity and responding with a proportional electrical output”.

The purpose of the sensor is to convert any physical properties, quantity or condition into an electrical signal, which could be voltage, current or charge. Thus, any sensor is an energy converter which converts some input properties into electrical properties (Fraden, 2010). It transforms data from one domain to another, relating the two domains in some sort of relationship (ibid.). This relationship must be based on some reference, and it will require some form of systematic calibration (Fowler, 2001).

There are two types of sensors used in measurement systems: direct sensors and complex sensors (Fraden, 2010). A direct sensor converts any physical stimulus or energy directly into an electrical signal, which can be fed to a signal condition circuit or system (ibid.). Direct sensors absorb energy directly from the measurand, which can lead to loss of information about the original state of the measurand (Regtien et al., 2004). These sensors rely on using some form of physical effect to convert energy into an electrical signal (ibid.).

A complex sensor requires one or more energy conversion stages before direct sensors can be used to generate an electrical signal output (Fraden, 2010). This implies that a complex sensor may consist of one or more transducers in the front end or input stages and a direct sensor at the output stages.

Sensors can be classified into two kinds: active and passive (Fraden, 2010). Sinclair (2001) defines an active sensor as a sensor that can generate an output signal without the need for an external power supply. He suggests that the sensors are self-generating. He also concludes that a passive sensor needs an external source of energy to operate. The definition of Sinclair (2001) conflicts with that of Wilson (2005) and Fraden (2010) regarding active and passive sensors. Wilson (2005) defines active sensors as sensors that require an external source of excitation and a passive sensor as self-generating sensors which generate their own electrical output signal, without requiring any external power source. Fraden (2010) defines active sensors as sensors that require external power or an excitation signal for their operation. They

modulate this signal according to some sensor parameter, and this modulated signal carries information of the measurand or stimulus value. He concludes that passive sensors do not require any external or additional energy source, and they directly generate an electrical output signal in response to a stimulus or measurand.

The definitions of Sinclair (2001) suggest that sensors that directly convert any form of energy into an electrical output are active or self-generating sensors, and sensors that need external power to perform such conversion are passive sensors. One of the meanings of the word active is to be physically energetic (Dictionary, 2014). This suggests that active sensors are physically active and only require a relevant stimulus to generate a proportional electrical signal.

Definitions by Wilson (2005) and Fraden (2010) conclude that active sensors in general require some form of power or excitation to generate an output signal from an input stimulus, and passive sensors are physically active and require only external stimulus to generate an output signal. In view of the above definitions, the industry convention is to classify the sensor or transducer with respect to external circuit requirements, i.e. the need or absence of external active circuitry to produce a sensor output signal (Jung, 2005).

A transducer is a device that converts or changes one form of energy to another (Fowler and Schmalzel, 2004). The transduction process involves sensing input energy with a sensing element and transforming it into another form by a transduction element (Coombs Jr, 2000). It allows transformation from input stimulus changes to electrical outputs that are suitable for data capturing and analysis (D'Amico and Di Natale, 2001). This suggests that a transducer must contain some form of a sensing element. A transducer in general must output signal in any industry standard such as 4-20mA protocol or a DC voltage between 0-5V and 0-10V (ibid.). Transducers can also be classified into active and passive types.

According to some authors, transducers are also called sensors, but this terminology is not standardised (Regtien, 2012). In some literature, transducers and sensors are differentiated in order to stress their differences in terms of their function, with a distinction made between an

element performing the physical conversion and the complete device (ibid.).

According to VIM (International Vocabulary of Metrology, 2012:36) standard, a transducer is defined as a “*device, used in measurement that provides an output quantity having a specified relation to the input quantity*”. From a measurement viewpoint, this definition suggests that a transducer produces an output based on some relationship with the input. This implies that the output and the input are not the same types of signal or measured quantity (Kamrul Islam and Haider, 2010). To add to the confusion, the ANSI (The American National Standards Institute) standard for an electrical transducer nomenclature and terminology defines transducers as a “*device which provides a usable output in response to a specific measurand*” (McGrath and Ni Scanail, 2013a:14). The main function of a sensor is to collect information from the real world (ibid.) and present this information in the form of electrical quantity (i.e. voltage or current). According to McGrath and Ni Scanail (2013a), the differences in these strict definitions will always be contentious and driven in part by the philosophical difference between engineers and scientists. In this research project, the word sensor or transducer will mean any sensing device which outputs an electrical signal using the industry standard 4-20mA protocol or a DC voltage between 0-5V and 0-10V in response to a stimulus and with the output and input having a directly proportional relationship.

In selecting a sensor or transducer, it is important to look at the fundamental characteristics and limitations applying to all types of sensors. Commercially available sensors used for measuring some physical quantity have fundamental limitations and imperfections that affect measurements.

2.3.1 DAS and Sensor Characteristics

The following are the key characteristics of any sensor: transfer function, accuracy, sensitivity, selectivity, span, resolution, linearity, threshold, hysteresis, noise, response time, repeatability, and dead band. The key characteristics relevant to DAS are studied in detail, specifically focusing on their effects on DAS performance.

2.3.1.1 Sensor Transfer Function

Sensor or transducers by definition must have some form of a relationship between the input and the output of the sensor. The transfer function defines a dependency relationship between the sensor electrical output and the input stimulus (Fowler and Schmalzel, 2004). This is the fundamental relationship between the sensor electrical output signal and input stimulus or measurand. The ideal sensor should have transfer characteristics dependent only on the measured variable (Soloman, 2010) and independent of any other variable. Theoretically, this relationship is represented by a transfer function that can be described statically and dynamically (Johnson, 2000). In sensor literature, the static transfer function can be stated or expressed in the form of table values, a graph, a mathematical equation or formula. The static transfer function describes an output and input relationship with the input not changing with respect to time. This means if the input is held constant and the output is obtained from a system, then the ratio between the output signal and input signal is the static transfer function. From a theoretical point of view, a sensor may be considered as a “black box” (see Figure 2.7) and only focuses on the relationship between the output electrical signal and input stimulus (Perez, 2002).

A sensor can be represented by a standard transfer function:

$$H = \frac{V_{out}}{V_{in}} \quad (2.1)$$

$$H = \frac{\text{Output Electrical Signal (I)}}{\text{Stimulus(S)}} \quad (2.2)$$

In practical applications, the stimulus (S) is the unknown parameter, while the output signal (I or V) is measured (Fraden, 2010). The measured parameter, which becomes known during measurement, could be a number such as a voltage or current that represents the value of the input stimulus (ibid.). Mathematically, this measured parameter is represented by:

$$I \text{ or } V = H \times S \quad (2.3)$$

where I or V = the output electrical signal current or voltage

H = the sensor transfer function

S = the input stimulus or the measurand

The measurement system must be able to decode the measured parameter V or I and deduce from the measured value the unknown value (stimulus) (Fraden, 2010). The measurement system actually performs the inverse function of the sensor. For the measuring system to infer the unknown value from the measured value, it must have some knowledge about the transfer function (Fraden, 2010). The transfer function of the sensor can be modelled mathematically, and the basis of the mathematical model must be based on some form of physical (physics law) or chemical law which defines the sensor operation (ibid.). In practice, the transfer function can easily become complicated if it accurately models the sensor behaviour (Schmalzel and Rauth, 2005). The solution is to use various function approximations to model the sensor's transfer function and inverse transfer functions (Fraden, 2010).

The inverse of the transfer function is represented by the following equation and can be used to obtain the input stimulus:

$$S = \frac{I \text{ or } V}{H} \quad (2.4)$$

In theory, if a sensor has a linear relationship between the output and input, it can be approximated by the following equation (Fraden, 2010):

$$V = A + BS \quad (2.5)$$

where V = the output signal

A = the output intercept

S = the input (stimulus or measurand)

B = the slope or sensitivity (Fraden, 2010)

This equation assumes a zero intercept of the transfer function when the input is zero (ibid.). This may mean that a sensor has no input offset. Actually, equation(2.5) represents a straight line equation ($y = mx + c$), where $A = c$, the y- intercept; $m = B$, which is the slope of the straight line; $x = S$ is the input; and $y = V$ is the output.

If the sensor is referenced to a non-zero input or there is some form of input offset, equation(2.5) can be rewritten as (Fraden, 2010):

$$V = v_0 + B(S - s_0) \quad (2.6)$$

where s_0 = the input offset or non-zero reference
 v_0 = the output signal at s_0

Fraden (2010) admits that there are very few sensors that are truly linear, and this view is also supported by Jung (2005). He also affirms that there will always be some small nonlinearity present, especially with a broader input range of the stimuli (ibid.). Perez (2002) maintains that a nonlinear transfer function can be represented by equations(2.7) and(2.8):

$$S = ae^{nI} \quad (2.7)$$

and

$$S = a + a_1 \ln I \quad (2.8)$$

where S = the output electrical signal
 I = the input stimulus, and n is constant in number

A sensor transfer function can also be described dynamically by the input and output relationship, with the input varying with respect to time, and this is actually the time response of the sensor (Johnson, 2000). In literature, this transfer function can be a model using first-order and second-order differential equations (Schmalzel and Rauth, 2005).

Theoretically, the dynamic transfer functions can be represented by differential equations in

the time domain (Ogata, 2002). The first-order equation actually describes a sensor that contains one energy storage component (Fraden, 2010). Moreover, it can be specified by a sensor manufacturer as the frequency response, thus describing how fast a first-order sensor can react to a change in the input stimulus (ibid.).

Because of manufacturing and other imperfections, all sensors will have a time lag between the output and input in response to a step change in the input (Fraden, 2010). In practice, this step or instantaneous changes rarely occur (Johnson, 2000) and may be present during worst-case conditions. Sensors must be able to track these changes in the physical dynamic variables in a time that is less than one time constant (ibid.), and this should be applicable to worst-case conditions.

When modelled with a first-order differential equation, a sensor response is represented by the following equation (Johnson, 2000):

$$b(t) = b_0 + (b_f - b_0)[1 - e^{-t/\tau}] \quad (2.9)$$

where b_0 = initial sensor output obtained from the static transfer function and the initial input
 b_f = final sensor output obtained from the static transfer function and the final input
 τ = sensor time constant

Equation(2.9) gives the sensor output response with respect to time in response to a step input (ibid.). According to Johnson (2000), this output is in error during transition time of the output from initial sensor output to final sensor output. Another critical parameter regarding the output response of the sensor is the time constant (τ) and is one of the sensor specifications (ibid.) which should be specified by the sensor manufacturer.

It is the measure of the sensor inertia (Fraden, 2010). This is actually the time it will take the sensor output value to change to approximately 0.63 or 63% of the total change (Christiansen et al., 2005a; Fraden, 2010; Johnson, 2000; Ogata, 2002). It can be determined by equating

$\tau = t$ and solving for the change in output from equation(2.9) (ibid.). According to Ogata (2002), the output response of a sensor modelled by equation(2.9) will only reach mathematical steadiness after an infinite time. It should be noted, though, that in practice, a reasonable estimate of the response time is given by the length of time the response curve must reach and stay within 2% of the line of the final value or the four-time constants (ibid.).

Fraden (2010) compiled equation(2.10) that he argues can be used to establish a relationship between the cut-off frequency and the time constant of first-order sensors:

$$f_c \approx \frac{0.159}{\tau} \quad (2.10)$$

where f_c = either the upper or lower cut-off frequency

The cut-off frequency specifies the lower and upper frequency of an input stimulus that the sensor can process (ibid.). The lower cut-off frequency indicates the slowest changing input stimulus the sensor can process, and the upper cut-off frequency indicates how fast the sensor reacts (ibid.). Another parameter for a sensor with a first-order response related to cut-off frequency is the phase shift. The phase shift at a specific frequency defines how the sensor output signal lags behind in representing the input change (ibid.).

Some sensors' dynamic transfer functions can also be modelled by a second-order differential equation (Schmalzel and Rauth, 2005). These equations describe a sensor that contains two energy storage components (Fraden, 2010). When the step input change is applied to this sensor, it will result in an oscillating output signal for a short period of time before settling to a steady state value, which will correspond to the input (Johnson, 2000). This oscillation is the function of the sensor (ibid.) and is one of the sensor's deficiencies. Johnson (2000) contends that it is impossible to describe the second-order response by a simple analytical expression. He developed equation(2.11) to describe the general behaviour in the time domain as:

$$R(t) \propto R_0 e^{-at} \sin(2\pi f_n t) \quad (2.11)$$

where $R(t)$ = the transducer or sensor output
 a = output damping constant
 f_n = natural frequency of the oscillation
 R_0 = amplitude

The damping constant and the natural frequency are the parameters of the sensors and must be specified by the manufacturer (ibid.). This second-order response is known as the frequency response of the sensor and is stated for a specific frequency range (Christiansen et al., 2005a).

Moshayedi et al. (2013) have shown that the transient rise time and the steady state for a second-order response of a sensor have more information and data regarding sensor behaviour. Their analyses and conclusion were based on experiments performed on air and ethanol sensors (ibid.). They also acknowledge that the development of these models will help in understating, simulating and improving the measurement system performance by utilising tools such as MATLAB® (ibid.). It is crucial to keep these models in mind when developing and designing a data acquisition system.

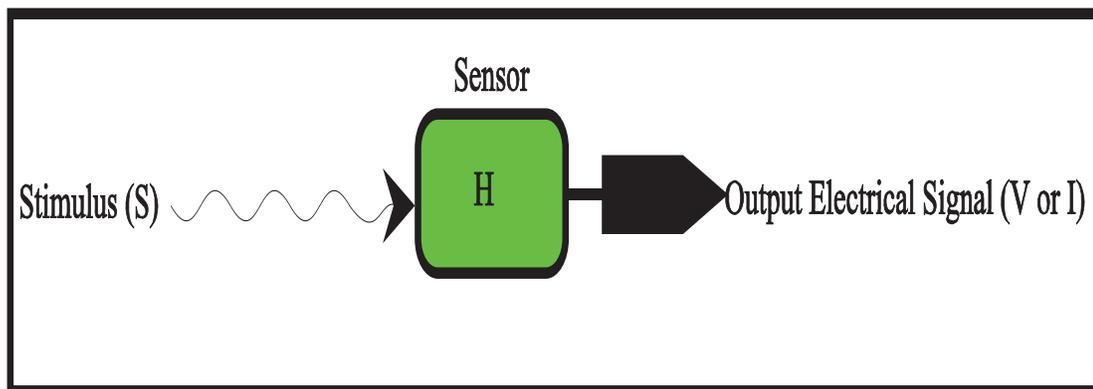


Figure 2.7 Sensor represented as a Black Box

Adapted from: Popovic (1995)

2.3.1.2 Accuracy

Accuracy in the context of sensors has to do with how much the sensor output value deviates from the actual measured value of the measurand or stimuli (Fowler and Schmalzel, 2004). This deviation must be referenced to a true value of the measurand that is traceable to some standard (ibid.).

According to Fraden (2010) and Johnson (2000), accuracy in terms of sensor error actually means inaccuracy. This is measured as the highest deviation of a value represented by the sensor from an ideal or true value of a stimulus at its input (Fraden, 2010). This ideal value is theoretical because it can never be said with absolute certain what this true or ideal value is in practice (Regtien et al., 2004). According Perez (2002), the deviation value can be described as the difference between the ideal input value and the actual input value that was converted by a sensor into a voltage or current. Mathematically, accuracy is equal to the ratio of the highest deviation of a value represented by the sensor to the ideal value (Bolton, 1992). Theoretically, accuracy describes the maximum expected error between the actual measured value and the ideal output signal (McGrath and Ni Scanaill, 2013b).

Bolton (1992) defines accuracy in the context of an instrument or measurement system as the extent to which the reading the instrument or measurement system gives might be wrong or the extent to which it differs from the true value. Moreover, the VIM (International Vocabulary of Metrology, 2012:21) standard defines the measurement accuracy as the *“closeness of agreement between a measured quantity value and a true quantity value of a measurand”*.

One should note that these accuracy errors are due to sensor imperfections, which are a result of manufacturer material variations, workmanship, design errors, manufacturing tolerances, and other limitations (Fraden, 2010), and are beyond the control of the user. Fraden (2010) submits that the accuracy of sensors is affected by the combination of part-to-part variations, hysteresis, dead band, and repeatability errors. This suggests that in practice, measures must be taken to reduce sensor accuracy errors. These measures may include calibrating each

sensor individually under specifically selected conditions and reducing reliance on manufacturer tolerances (ibid.). Again, Fraden (2010) suggests that this can be achieved by multi-point calibration of each sensor and using curve fitting techniques. He concludes that the specified accuracy limits will not be established around the theoretical ideal transfer function but around the actual calibration curve, which can be adjusted during sensor calibration.

In practice, accuracy or inaccuracy can be specified directly in different forms. It might be given in terms of the following:

- The measured value or variable which is given in percentage form. When specified in terms of measured value, this error is independent of the input signal magnitude and combines all other sensor errors (Fraden, 2010).
- Percentage of the sensor input span or full-scale reading. This form is useful if the sensor has a linear transfer function (ibid.).
- As a percentage of the measured signal or actual reading from the sensor (Fraden, 2010; Johnson, 2000).
- In terms of the output signal. This form is useful when specifying the error of a digital output (ibid.) from an analogue-to-digital converter (ADC). It is expressed as the percentage deviation of the analogue input per bit of the digital signal ($\pm n\%$ LSB) ADC (Johnson, 2000).

2.3.1.3 Sensitivity

Sensitivity of a sensor basically specifies the ability of a sensor to detect any change in input stimulus. It is mathematically expressed as the ratio of the change in sensor output signal in response to or resulting from a change in the stimulus (Bolton, 1992). It can be generally represented by the following equation:

$$\text{Sensitivity} = \frac{\Delta y}{\Delta x} \quad (2.12)$$

where y = the output

x = the input or stimulus

Fraden (2010) posits that coefficient B in equation (2.5) can be regarded as the sensitivity of a linear transfer function. This suggests that a sensor with a linear transfer function has a constant sensitivity across all different points in the interval of the input measurand (ibid.). This means that for a nonlinear transfer function, this coefficient is not constant. Sensitivity curve can be obtained by applying a derivative procedure from the response curve and considering a point-by-point derivative of the response curve with respect to the input (D’Amico and Di Natale, 2001).

According to Johnson (2000), sensitivity should be evaluated together with parameters such as linearity of the output to the input, sensor range and accuracy. This suggests that sensitivity has some form of relationship or dependence on these parameters. It can be argued that if a nonlinear response curve can be linearised by a piecewise linearisation procedure, then the sensitivity can be obtained on each linearised region of the curve (D’Amico and Di Natale, 2001). However, these require that the range of measurand that this linear region holds should be specified (ibid.). D’Amico and Di Natale (2001) postulate that a sensor with a nonlinear transfer function has a sensitivity that is a function of the operating point. It can be represented by the following equation:

$$\mathbf{Sensitivity}_{nonlinear} = \frac{\partial y}{\partial x} \quad (2.13)$$

This equation (2.13) suggests that sensitivity is applicable only to a particular input range of the nonlinear curve. In practice, it would be important to use a high sensitivity sensor, which will result in a large output signal in response to a small measurand change.

2.3.1.4 Resolution

Resolution may be defined as the smallest change in the measurand that can be detected by a sensor. It can also be defined as the “*smallest increment of the measurand that causes a change in output*” of a sensor (McGrath and Ni Scanaill, 2013b:3). This smallest change in

measurand will result in a sensor output that is not perfectly smooth, as a result of the output changing in small steps (Fraden, 2010). These changes in small steps in the output signal will cause quantisation errors (Li et al., 2009). It means that inherently any sensor contains quantisation errors on its output signal due to resolution limitations.

It should be noted that the smallest change in the measurand may not necessarily be due to the accuracy of the sensor because the sensor transfer function may be non-linear (Fowler and Schmalzel, 2004). Fundamentally, the sensor resolution is limited by the noise level internal to the sensor and the associated electronic circuits (D'Amico and Di Natale, 2001; McGrath and Ni Scanail, 2013b). D'Amico and Di Natale (2001) assert that the noise and sensitivity value limits of a sensor are the fundamental reasons why the sensor resolution will not approach zero for any practical sensor. They give a general mathematical form of resolution as follows:

$$\mathit{resolution} = \lim_{V_s \rightarrow V_{noise}} \frac{V_s}{S} = \frac{V_{noise}}{S} \quad (2.14)$$

where V_s is the sensor output, V_{noise} is the sensor noise level, and S is the sensor sensitivity. Both parameters, V_{noise} and S , specified the theoretical minimum noise and the maximum sensitivity of a sensor (ibid.). It can be argued from equation(2.14) that it will not be possible to determine the resolution without knowing the sensor noise level (ibid.) or the sensor sensitivity limits. Fraden (2010) argues that the resolution should be specified under specific conditions. This view is also supported by D'Amico and Di Natale (2001) who concluded that it will not be correct to specify the resolution of a sensor without specify its operating point. Bolton (1992) suggests that a resolution can also be expressed as the smallest change in the input – as a fraction of the maximum input value. However, this expression will not be useful in terms of sensors, as it does not take into account the noise limits of the sensor.

In practice, a resolution can be described as the threshold, provided that the smallest increments are measured from zero (McGrath and Ni Scanail, 2013b). This suggests that the noise levels should be the limits of the measurements.

A sensor with negligible steps in the output signal can be said to have a continuous or infinitesimal resolution (Fraden, 2010). When using such sensors, quantisation errors can cautiously be neglected or ignored.

2.3.1.5 Span

Span is the operating range of a sensor given by the upper and lower limits, within which the sensor will maintain its specified accuracy (Fraden, 2010). It specifies the limits of the input stimulus that the sensor can detect and produces the proportional output signal. It actually describes the difference between the maximum and minimum values of the input stimulus (McGrath and Ni Scanail, 2013b). The lower limit may be limited by the noise floor of the sensor.

The dynamic range of the sensor specifies the minimum and maximum values of a sensor that can be reliably measured (ibid.). If the dynamic range of inputs is very large or broad and the sensor has a nonlinear response characteristic, the dynamic range is expressed in decibels (Fowler and Schmalzel, 2004; Fraden, 2010). This is the logarithmic expression of two ratios and has a nonlinear scale. It may help in representing low-level signals with a high resolution and compressing high-level signals (Fraden, 2010). It can be argued that the dynamic range determines the total input energy that can be captured by the sensor (Suranthiran and Jayasuriya, 2003).

In the industry, either the full-scale range or dynamic range is used to specify the span in the manufacturer's datasheets (McGrath and Ni Scanail, 2013b). In some literature, the word range is used instead of span and is also called full-scale input (FS) (D'Apuzzo and Liguori, 1999).

2.3.1.6 Linearity and Nonlinearity

Linearity and nonlinearity specifies the maximum deviation of the sensor output from a best fit or approximation straight line over a specific sensor span (Fraden, 2010; Johnson, 2000; Perez, 2002; Regtien, 1992). Mathematically, they may be expressed as the difference between the actual and the ideal output line of a sensor in response to a stimulus. They both

specify some form of a proportionality relationship of the sensor output signal to the measurand input or stimulus (Fowler and Schmalzel, 2004). According to Fraden (2010), linearity must be specified with the type of straight line used to determine the linear relationship. There are several methods used to specify the nonlinearity of the sensor. One way is to use the least square method (D'Apuzzo and Liguori, 1999; Perez, 2002). Another way is to use the terminal points method, which determines the sensor output values from the smallest and highest stimulus values, and then a straight line is drawn through these two points (D'Apuzzo and Liguori, 1999; Fraden, 2010). These methods suggest that some form of calibration is required to reduce and minimise nonlinearity errors. Manufacturers of sensors usually specify nonlinearity or linearity expressed as a percentage of the full-scale input (Johnson, 2000; Regtien, 1992). Fraden (2010) contends that the manufacturer specifies nonlinearity using the smallest possible number without stating what method was used to obtain that value. This may result in measurement errors and misinterpretation of measured data.

Nonlinearity is inevitable and presents almost all sensors used in measurement systems (Suranthiran and Jayasuriya, 2003). In practical engineering application, several techniques can be used to improve, rectify and minimise nonlinearity. These techniques may include but not limited to the correction of nonlinearity using analogue signal conditioning, digital signal processing (Dias Pereira et al., 2007) and software algorithms.

2.3.1.7 Hysteresis

Fraden (2010:35) defines hysteresis error as the “*deviation of the sensors output at a specified point of the input signal when it is approached from the opposite directions*”. This basically means the sensor response consists of two transfer functions: one specifying an input-output relationship, with the input stimulus increasing from minimum to maximum, and the other with input stimulus decreasing from maximum to minimum (Johnson, 2000). Hysteresis is then the difference between the measured outputs obtained when the input stimulus is increasing and when the input stimulus is decreasing (Bolton, 1992; D'Apuzzo and Liguori, 1999; Johnson, 2000). In other words, hysteresis is a phenomenon with which the state of the

sensor output does not reversibly follow changes in the measurand (Perez, 2002).

According to some authors (D'Apuzzo and Liguori, 1999; Fowler and Schmalzel, 2004), the output response may depend on the previous stimulus input, whether it was higher or lower than the current input. Hysteresis error is due to inherent sensor limitations such as the geometry of the design, friction and structural changes of the sensing element (D'Apuzzo and Liguori, 1999; Fraden, 2010). In practice, sensor manufacturers specify hysteresis as a percentage of full-scale maximum deviation from the two transfer functions or the full-scale output (D'Apuzzo and Liguori, 1999; Johnson, 2000).

2.3.1.8 Noise

Noise is a fundamental phenomenon which arises due to small current or voltage fluctuations generated within devices (Lita et al., 2001) and random motion of electrons in resistive devices and random generation and recombination of holes and electrons within semiconductor (Leach, 1999).

An ideal sensor will not have any noise, and due to material deficiency, this is not a possibility. Numerous literature abound which discuss in detail the concept of noise in electronic devices. In the context of sensors, the noise within the sensor limits its performance and is the most significant noise source (Pottie and Kaiser, 2005). In practice, noise is inevitable (Suranthiran and Jayasuriya, 2003) with any sensor used for measurements. The inherent noise sources of a sensor can be due to thermal noise (Johnson noise), shot noise, photon noise and 1/f noise (flicker noise) (Bordoni and D'amico, 1990; Fraden, 2010; Goldie et al., 2009; Leach, 1999; Meyer et al., 2011; Pottie and Kaiser, 2005).

Johnson, shot and photon noise are fundamental in nature and have a power spectrum density which is constant over a broader frequency range (Bordoni and D'amico, 1990; Fraden, 2010). Moreover, it can be argued that the flicker (1/f) noise is more pronounced in the lower frequency range below 100Hz (Fraden, 2010); it coexists with the Johnson, shot and photon noise. According to Bordoni and D'amico (1990), 1/f noise exists in any physical devices, and its magnitude depends on the current passing through a resistive or semiconductor material

(Fraden, 2010; Meyer et al., 2011).

One of the fundamental limits of a sensor, which is as a result of noise, is that the sensor output signal will never truly represent the input stimulus (Fraden, 2010). In engineering applications, this will not be ideal. It should be noted that the external noise arising from the physical environment should not be considered as sensor limitations (Pottie and Kaiser, 2005). It can be argued (ibid.) that at the sensor input, the measurand signal and noise are indistinguishable. This suggests the noise and signal must have some similarity or noise must be embedded within the measurand signal or added at the sensor input (ibid.). Moreover, the measured data may be below the noise floor of the sensor, which is the level at which the signal is lost in the noise (Ida, 2013). In engineering applications, it is desired to have a sensor with a high signal-to-noise ratio (Fowler and Schmalzel, 2004), and it is also very important to understand the limits on signal detection imposed by noise (Cantley et al., 2012). In manufacturers' datasheets, noise is expressed in terms of root mean square (RMS) values at specific bandwidth and terms of signal-to-noise ratio usually given in decibels (D'Apuzzo and Liguori, 1999).

2.3.1.9 Repeatability

Repeatability is the ability of the sensor to reproduce the same output signal when the same input signal is repeatedly applied to it and under identical conditions (Fraden, 2010; McGrath and Ni Scanail, 2013b). Kularatna (2003:15) defines repeatability "*as the degree of agreement among independent measurements of a quantity under the same condition*". Further, Ida (2013) also maintains that repeatability indicates the failure of the sensor to reproduce the same stimulus value under similar conditions when measured at different times. In engineering application, sensor repeatability is the most important performance parameter (Ida, 2013; Wilson, 2005) which needs to be considered when using a sensor and DAS.

According to McGrath and Ni Scanail (2013b), poor repeatability may be a result of random fluctuations in environmental inputs. This suggests that noise signals superimposed on the input signal will cause poor repeatability. James (2000) posits that some sensors output signal

drift over time, due to instabilities as a result of changing operating temperatures and other environmental factors. These instabilities may result in poor sensor repeatability. According to Fraden (2010), some source of poor repeatability may be due to inherent sensor noise (i.e. thermal noise), charge build on the sensor and material deficiency (i.e. plasticity).

Fowler and Schmalzel (2004b) define precision as the repeatability of the measurement from the sensor. On the other hand, according to Bagajewicz and Chmielewski (2010), precision is formally known as repeatability or reproducibility by the International Society of Automation (ISA).

This means that for a sensor to be precise, it must have good repeatability. The word reproducibility is used interchangeable with repeatability in some literature (Bagajewicz and Chmielewski, 2010; Fraden, 2010; Ida, 2013). Reproducibility is defined as the sensor's ability to indicate the same values of the measurand each time a measurement is made, assuming that all environmental conditions are the same for each measurement (D'Apuzzo and Liguori, 1999). Bagajewicz and Chmielewski (2010) postulate that reproducibility is defined the same way as repeatability as per ISA standard. However, the definitions differ only on the approach of the input direction, where for repeatability, input direction is approached from one direction, while for reproducibility, the input direction is approached from both directions (ibid.).

In order for the words repeatability and reproducibility to be used interchangeable, it will be required that the environmental conditions used to measure the measurand be exactly the same. In practice, repeatability is given as the difference between two measurement values obtained at different times under identical input conditions (Ida, 2013) and is expressed in terms of percentage of full scale (Fraden, 2010; Ida, 2013).

2.3.1.10 Dead Band

International Vocabulary of Metrology (2012:42) defines dead band "*as a maximum interval through which a value of a quantity being measured can be changed in both directions without producing a detectable change in the corresponding indication*". This definition is

similar to that of D'Apuzzo and Liguori (1999) in which they define a dead band as a maximum interval through which an input stimulus changes in both directions without producing an output signal or response. Both definitions agree on one theme, which basically is the limit of input stimulus, i.e. a sensor can react to and produce a response. This essentially means the sensor will be insensitive or unresponsive to a specific range of input signals (D'Apuzzo and Liguori, 1999; Fraden, 2010; Ida, 2013; McGrath and Ni Scanail, 2013b), which could be very small or below the noise floor of the sensor. In some applications, the dead band is widened to prevent any changes in output response for small changes in the stimulus (D'Apuzzo and Liguori, 1999). It is then essential to understand this fundamental limit of a sensor when used in measurements. In practice, a dead band is expressed as a percentage of a full-scale range or span (Gill and Hexter, 1973; McGrath and Ni Scanail, 2013b).

2.3.2 DAS and Sensor Errors

The universal definition for an error is the difference between the measured value and the true value of the measurand (Bolton, 1992; Dunn, 2014; Ida, 2013; McGrath and Ni Scanail, 2013b; Regtien et al., 2004). Mathematically, it is expressed as:

$$e_r = \text{measured value} - \text{true value} \quad (2.15)$$

Kularatna (2003) defines an error as the difference between the measured value and the true value of the measurand after all corrections have been made. The definition of Kularatna (2003) suggests that some form of corrections or adjustment must be made on the measured value before the final error of measurement is computed. In the real world, there is no such a thing as a true value, as this is actually a theoretical concept (Salicone, 2013). This means the true value is unknown and, therefore, indeterminate (Regtien et al., 2004). Thus, the error magnitude will be unknown (Kularatna, 2003a), which would render equation(2.15) invalid. In practice, the true value is referenced to some absolute or agreed standard (ibid.). Through the referenced standard, this true value can be accepted as having a negligible uncertainty for measurement purposes (Joint Committee for Guides in Metrology (JCGM/WG 2), 2012;

Regtien, 2004). Uncertainty is defined as an estimate of the possible error in a measurement (Kularatna, 2003a). It can be argued that if the true value is estimated to have a very small error – thus negligible uncertainty – and this value is traceable to some standard, then the error can be obtained from equation(2.15).

In the context of sensors, there are two kinds of errors: systematic errors and random errors. They are important in engineering applications and contribute negatively to sensor performance. In practice, an error is expressed in terms of equation(2.15) or as a percentage of the input full scale (IFS) or percentage of the output full scale (OFS) (Bolton, 1992; Ida, 2013; Johnson, 2000).

2.3.2.1 Systematic Errors

Systematic error is defined “*as the closeness of agreement of the mean value of a number of consecutive measurements of a variable that maintains its value static*” (Bagajewicz and Chmielewski, 2010:4). In addition, the Joint Committee for Guides in Metrology (JCGM/WG 2), 2012:22) defines it “*as a component of measurement error that in replicate measurements remains constant or varies in a predictable manner*”. Both definitions essentially mean that systemic errors are errors that are reproducible (McGrath and Ni Scanail, 2013b), predictable, constant (Ida, 2013) and inherently present in any measurement done by a sensor. They may be classified as intrinsic errors (Finkelstein, 2014), which are the result of inherent imperfections in sensors.

The source of systematic error may be due to many factors such as manufacturing deficiency, material imperfections, signal transmission, aging, calibration errors, and operational errors (Beaty and Fink, 2013; Ida, 2013; McGrath and Ni Scanail, 2013b), and these error sources must be characterised by the manufacturer. Because of their predictable and constant nature, sensor errors resulting due to systematic errors can be corrected using compensation methods such as feedback (at system level), filtering and calibration (Joint Committee for Guides in Metrology (JCGM/WG 2), 2012; McGrath and Ni Scanail, 2013b).

2.3.2.2 Random Errors

Random errors are accidental by nature and fluctuate in an unpredictable manner (Beaty and Fink, 2013). They follow the laws of chance, and they do not have a consistent magnitude and polarity (Christiansen et al., 2005b). These errors will result in different error values each time a measurement is taken. Because of their unpredictable nature, they may be considered as noise because they carry no information (McGrath and Ni Scanail, 2013b). In a great deal of literature, random errors are viewed and are assumed to follow Gaussian distribution (Bich, 2012; Finkelstein, 2014; McGrath and Ni Scanail, 2013b). Random error can, therefore, be modelled and analysed using statistical tools such as averages, dispersion from the average and probability distribution of errors (Cooper and Helfrick, 1985; Finkelstein, 2014; Kularatna, 2003b).

The fact that noise signals are random signals by nature and they may be superimposed on the measured signal, taking averages of a number of measured values over a period of time might reduce the effect of noise (Bolton, 1992). This basically means the arithmetic average of various observations or measurements should be used to minimise the effects of random errors (Christiansen et al., 2005a).

The source of random errors within the context of sensors may be due to an external source (McGrath and Ni Scanail, 2013b) such as noise in the measurand, environmental noise, transmission noise and internal noise (D'Apuzzo and Liguori, 1999). In practice, manufacturers usually characterise the internal noise of the sensor.

The above sensor characteristics (errors) are usually specified for laboratory conditions. It is thus essential to know the effects of operating environment conditions, if sensors are used under conditions that are different from the conditions they were calibrated in (ibid.). In practice, these effects are not adequately specified and documented by manufacturer datasheets. Therefore, tests should be performed to characterise the environmental effects on the stated errors specified by the manufacturer (D'Apuzzo and Liguori, 1999; Fraden, 2010). Environmental effects usually affecting sensor specification consists of temperature effects,

pressure effects, acceleration effects, vibration effects and mounting effects (ibid.).

2.3.3 4-20mA Transmission Protocol

It is a fact that signal levels generated by some sensors are very low (i.e. thermocouple generate $10\text{-}50\mu\text{V}/^\circ\text{C}$ (Ida, 2013)) and not suitable for transmission. It is obvious that signal amplification is essential for a useable signal to be transmitted (Morris, 2001). Analogue voltage signals suffer signal attenuation, especially over a long distance, due to wire resistance and are easily susceptible to noise.

Because of transmission limitations of a varying voltage signal, in the industry, a varying current signal is used in sensor signal transmission and is known as a 4-20mA current loop interface (DeNatale et al., 2003; Dudojc, 2008; Johnson, 2000; Lohiya and Talbar, 1998; McMillan, 1999; Morris and Langari, 2012; Morris, 2001; Pereira, 2004). This interface has inherently high noise immunity (McMillan, 1999; Morris and Langari, 2012), and this is because it is a differential signal. This view is disputed by Dudojc (2008:5), who argues that it is only theoretical; he concludes that the *“level of noise immunity depends mainly on the properties of the transmitter”*.

The 4-20mA current loop interface uses current in the range of 4mA to 20mA to represent the output signal of a sensor (Morris and Langari, 2012). In practice, the output current is usually scaled and has a linear relationship to the measurand (DeNatale et al., 2003; Dos Reis Filho, 1989; Dudojc, 2008; Lohiya and Talbar, 1998; Pereira, 2004). The 4mA current is the lower limit or minimum value of the measurand, and the 20mA is the upper limit or maximum value of the measurand (International Society of Automation (ISA), 2012; Pereira, 2004). The interface has an inherent fault detection, which is indicated by zero current.

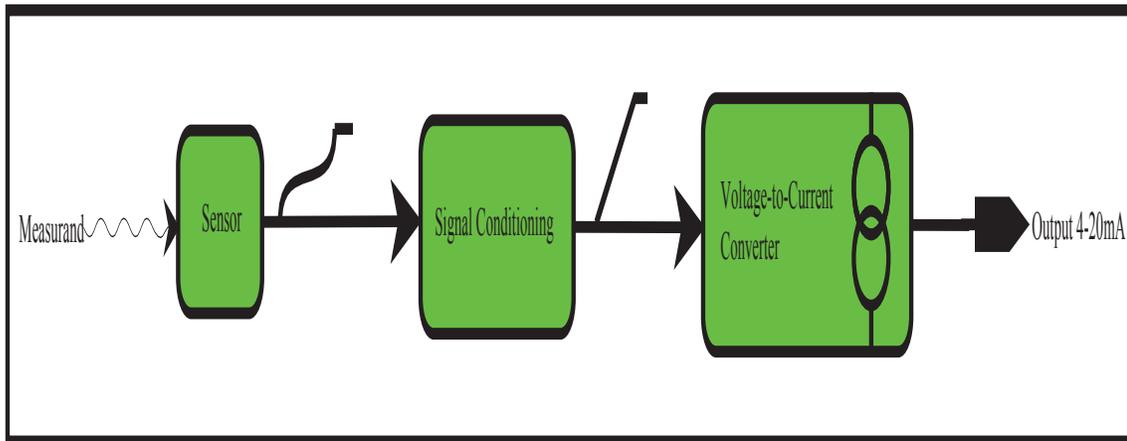


Figure 2.8 Sensor with 4-20mA current loop interface

A typical sensor with 4-20mA current loop interface may be represented by Figure 2.8. It consists of a sensor, which performs the sensing and produces an output electrical signal, which is usually a very small voltage or current (Fraden, 2010) and non-linear (de Barros Soldera et al., 2012; Hosticka, 2007; Niu, 2012) in response to input stimulus. It is followed by a signal conditioner that may perform functions such as signal amplification, filtering and linearisation (McMillan, 1999; Niu, 2012; Nuccio and Spataro, 2002). The signal conditioner is followed by voltage-to-current converter (Morris and Langari, 2012), which provides the 4-20mA output signal. In the industry, these sensors are known as 4-20mA current loop transmitters (Chaipurimas et al., 2010; International Society of Automation (ISA), 2012; Pereira, 2004) and must comply with the ANSI and ISA signal transmission standards, especially the ANSI/ISA-50.00.01-1975 (R2012) standard.

2.4 DAS and Signal Processing

In many scientific and engineering research projects, data acquisition systems are used to collect information from the physical world. This information is transmitted by sensors. Depending on the type of research, this system complexity may vary (Broggi et al., 2006; Harrison et al., 2012; Michalski and Makowski, 2012; Siskind, 2007; Y. Zhang et al., 2012). It is therefore important to determine the quality of information or data collected from sensors using such systems. DAS usually consists of signal measurement and signal processing chains, which may affect and

influence data integrity. Thus, it is essential to study the DAS theoretical foundations, components, limitations, accuracy and data integrity and measurement reliability.

2.4.1 Data Acquisition System (DAS)

DAS can be defined as a measurement system (Bolton, 1992; Regtien et al., 2004) that collects or measures and processes signals acquired from the physical world. According to Finkelstein (2014), instruments are defined as devices for the acquisition and processing of information acquired from the physical world. This definition suggests that a DAS can be classified as an instrument and thus an information machine (Finkelstein, 2014). Within the context of instrument science, data acquisition systems are a subset of information machines whose function is to acquire, process, and feed out information (ibid.).

There are numerous data acquisition systems in the industry that have diverse functions and form (Abdallah et al., 2011). They may differ on the type of measurand, characteristics of the signals, level of accuracy required, and environment of application (Finkelstein, 2014).

Data acquisition is a well-researched area with extensive literature regarding data acquisition systems, with different complexity and applications area. In literature, data acquisition systems are viewed as a set of interconnected components functioning as one unit (Finkelstein, 2014) to accomplish signal collection and processing function. This suggests that a data acquisition system can be regarded as a measuring system; thus, a system approach can be used for their design and analysis (Barwicz and Morawski, 1999). Within the context of system approach, a data acquisition system can be decomposed into individual components consisting of sensor, signal conditioning, signal acquisition and processing, data display, and storage subsystems (Austerlitz, 2003; Bolton, 1992; Finkelstein, 2014; Michalski and Makowski, 2012; Najarian et al., 2004; Park and Mackay, 2003; Regtien et al., 2004). This is illustrated in Figure 2.9.

Abdallah et al. (2011) suggest that in the industry, DAS systems are generally classified into three main categories: computer-based, embedded microcontroller-based, and field-programmable gate array (FPGA) based. The computer-based DAS will use the processing

power of the PC to perform the data acquisition, manipulation, visualisation and storage, which is an advantage for such a system. A disadvantage of the system is that it still requires data acquisition hardware that will connect to the PC via Peripheral Component Interconnect (PCI) slots or computer ports. The microcontroller-based DAS has the advantages of portability and high performance, and both systems have a fixed architecture. However, it can be argued that the embedded microcontroller-based class can be expanded to include digital signal processor (DSP) (Antonyuk et al., 2007) and a high performance microprocessor.

The FPGA-based system has the advantage of configurability and field programmability and easy redesign of the system to fine-tune and improve system performance (Abdallah et al., 2011). It can also be argued that this class can be expanded to include systems combining both DSP and FPGAs (Baofeng et al., 2010), or FPGA and microprocessors to perform data acquisition. This class may allow future expandability to cater for improvements in sensor technology and other technological advances. In practice, the decision respecting which category to implement will be based on system parameters such as sampling speed, accuracy, resolution, amount of data, data processing and display requirements (Austerlitz, 2003).

One important component of a data acquisition system which some authors fail to mention is software. It is used to control the collection of data, processing and display of data (Park and Mackay, 2003) and provides some form of human-machine interface.

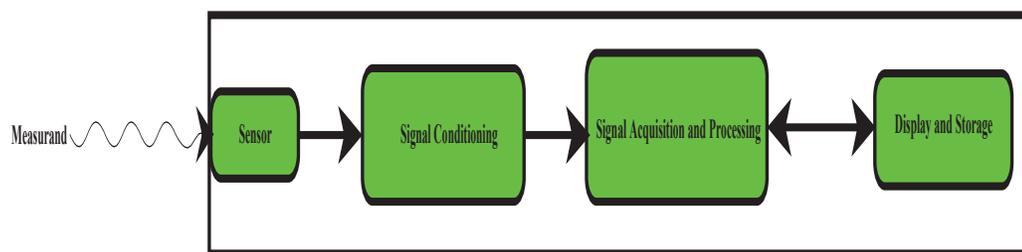


Figure 2.9 Simplified DAS

2.4.2 Signal Conditioning

Signal condition is one of the critical input stage components of any DAS, and its main function is to optimise signal transfer (Regtien et al., 2004). It converts a signal from a sensor

to a form that a data acquisition stage can handle (Dempster, 2001). This essentially involves making the sensor output signal compatible with the loading devices (Catunda et al., 2002; Fraden, 2010; McMillan, 1999) or data acquisition stage. Raw signals from the sensor are manipulated and transformed into a form suitable for the data acquisition stage, which will consist of an analogue-to-digital converter. Signal conditioning stage may be a source of measurement errors; thus, it is important to characterise and understand its limitations. Furthermore, it should not obscure or interfere with the accuracy of sensor data (Wilson, 2005). Generally, signal conditioning must enhance the performance and accuracy of the data acquisition system. In practice, signal transfer optimisation can be realised using amplification, filtering, impedance matching, isolation and linearisation (Dempster, 2001; McMillan, 1999; Regtien et al., 2004; Schmalzel and Rauth, 2005).

2.4.2.1 Amplification

The core of most signal conditioning stage consists of operational amplifiers (OPAMPs), which may be configured to perform amplification, attenuation, current-to-voltage conversion and DC level shifting (Broesch, 2009; Catunda et al., 2002; Christiansen et al., 2005a; McMillan, 1999; Regtien et al., 2004; Schmalzel and Rauth, 2005). In practice, the signal conditioning configuration applied will be governed by the relationship between the measurand and the sensor output signal (Christiansen et al., 2005a). One of the primary tasks performed by an amplification process is to increase the resolution of the signal. This improves signal-to-noise ratio, thus preventing loss of signal in noise (Dempster, 2001). It can be asserted that sensor signal must be protected against noise and interference, and noise suppression must be performed in the analogue domain (Regtien et al., 2004). This may aid to reduce the noise transformed into the digital domain by a data acquisition stage.

2.4.2.2 Filtering

The main task of filtering is to remove noise from sensor output signals before they are presented to the data acquisition stage (Dempster, 2001). In literature, this filters are usually known as anti-aliasing filters (Broesch, 2009; Oshana, 2007). From a theoretical context, they can be viewed as a mechanism to slow down the input signal, thus ensuring the data

acquisition stage will be able to track the input signal (*ibid.*). This essentially means that the fast-changing signals will be removed. It can be argued that this fast-changing signals may contain critical information about the measurand (Rieger and Demosthenous, 2008). In order not to remove such critical information about the measurand using a filter, it can be concluded that the signal of interest must be limited to the Nyquist frequency (Oshana, 2007).

Filtering is performed by filters, which are analogue circuits designed to pass signals with desired frequencies and attenuate unwanted signals. In literature, they are categorised into two types: passive and active filters. Passive filters – as the name suggests – are made only of passive components (i.e. resistors, capacitors and inductors). On the other hand, active filters are made of a combination of passive components and OPAMPs (Dempster, 2001). In theory, an ideal filter will have an ideal cut-off frequency, which will not be achieved in practice.

In most engineering applications, active filters are preferred to passive filters due to their sharp cut-off frequency (Dempster, 2001). It should be noted that filters can be implemented both in the analogue and digital domain (Oshana, 2007). To implement filtering in the digital domain, the signal must be transformed from the analogue to the digital domain using an analogue-to-digital converter. There is an abundance of written material dedicated to filter design and implementation.

2.4.2.3 Impedance Matching and Isolation

Impedance matching is important in measuring systems. If it is not implemented correctly, it will affect the overall performance of the system. From a theoretical context, the signal source can be considered an ideal voltage source with no internal impedance in series with output impedance, thus zero output impedance. Moreover, it can also be considered as an ideal current source with an infinite internal resistance (Bolton, 1992; Regtien et al., 2004; Sobot, 2012; Wang, 2010). This means with such ideal impedances, the signal source will not be affected by circuit loading, thus error-free signal transfer.

However, in practice, the voltage signal source will have some low impedance in the output, and current sources will have finite high impedance (*ibid.*). Because of the foregoing

limitations, it will be desirable to match the sensor signal output impedances with that of the signal conditioning input impedance.

In engineering applications, for optimal voltage signal transfer, the input impedance must be very large compared to the voltage source output impedance, and for optimal current signal transfer input, impedance must be very small compared to the current source output impedance (Regtien et al., 2004; Wang, 2010). What is more, to obtain maximum power transfer, the output impedance must be equal to input impedance (Regtien et al., 2004; Sobot, 2012). These theories suggest static impedances on the output and input stages, which is not necessarily true in practice.

Furthermore, isolation provides some form of galvanic isolation (Wilson, 2005), which passes a signal from sensor stages to the acquisition stage without any physical connection. It may prevent any loading effects from the filter stages (Doboli and Currie, 2011). Isolation may help with the control and removal of ground loops and rejection of possible high common mode voltages (Bansal et al., 2013; Dempster, 2001; Fraden, 2010; Kim et al., 2012; Wang et al., 2010) which provide some form of safety.

Isolation is usually implemented using either opto-isolation, magnetic or capacitive isolation (Dempster, 2001; McMillan, 1999). Depending on the application, opto-isolators are used mostly in the digital domain, due to their nonlinear transfer function (Fowler, 1996). Moreover, magnetic and capacitive isolation are used in the analogue domain because of their linear transfer function (ibid.).

2.4.3 Signal Acquisition and Processing

The signal conditioning stage is usually followed by a signal acquisition stage. The main fundamental function of the signal acquisition is to transform the analogue signal to the digital domain. This function is performed by an analogue-to-digital converter (ADC), which is the kernel of the signal acquisition stage. Thus, it is critical and important to understand the theory and limitations of the ADC.

DAS must have some form of intelligence (i.e. firmware and software) to control the acquisition, processing, displaying and storage of data. This may be performed by a dedicated system such as a PC or combination of PC and microprocessor.

2.4.3.1 Analogue-to-Digital Converter (ADC)

As mentioned in literature, signals collected from sensors must be transformed from the analogue domain to the digital domain to enable easy processing, displaying and storage of these signals. This function is performed by the ADC. Moreover, they might be a limiting factor in the performance of the data acquisition system (Lowdermilk and Harris, 2003) if an appropriate ADC is not chosen carefully. From a system context, the ADC can be decomposed into two main subsystems: sampler (Aziz et al., 1996; Rauth and Randal, 2005) and quantizer (Norsworthy et al., 1997).

Sampler

A sampler is basically a sample-and-hold circuitry made of a switch and capacitor (Rauth and Randal, 2005). As the name suggests, its main function is to sample the analogue input signal at specific time intervals. This is achieved by controlling the switching at a specific frequency (sampling frequency) and the capacitor holding that sampled value at that specific instant. This will result in discrete signals as a function of instant time. It is obvious that the sampler converts a continuous analogue signal into discrete samples at specific time intervals. The discrete samples will have the same amplitude as the continuous analogue signal, but only at integer multiples of the switching frequency (Ifeachor and Jervis, 2002; Rauth and Randal, 2005). Mathematically, the output of the sampler can be expressed as:

$$a_s(t) = \sum_{n=-\infty}^{\infty} a(nT_s) \times \delta(t - nT_s) \quad (2.16)$$

where n is an integer number, T_s is the sampling period, a_s is the sampler output, and a is the sampled analogue signal. It is clear from equation (2.16) that the output of the sampler is still an analogue signal, but not continuous in time, thus discrete-time analogue signal (ibid.).

Again, equation (2.16) clearly indicates that to have a faithful representation of the analogue

signal, the sampling period must be very short. If it is too long, critical information can be lost from the sampling process (Kularatna, 2000). This brings up an important concept in the data acquisition Nyquist sampling theorem (Nyquist rate), which basically means that the minimum sampling rate must be greater than twice the highest frequency component of the sampled input analogue signal (Engelberg, 2007; Ifeachor and Jervis, 2002; Phillips and Nagle, 1995; Rauth and Randal, 2005). It is expressed as:

$$F_s \geq 2f_{max} \quad (2.17)$$

where F_s is the sampling frequency and f_{max} is the highest frequency component of the input signal bandwidth. If conditions in equation (2.17) are met, it will in theory allow an exact reconstruction of the sampled signal and may even contribute in avoiding aliasing (ibid.). One of the disadvantages of a sampling process is that it '*replicates the frequency spectrum of the analogue signal at the integer multiples of the sampling frequency*' (Rauth and Randal, 2005:45).

In practice, this problem is mitigated by using anti-aliasing filters and oversampling the analogue signal. The increase in sampling frequency, which is more than what is required by equation (2.17), will aid in widening the gap between the signal and spectral replicas (Ifeachor and Jervis, 2002). In engineering applications, oversampling may be used with low-resolution ADC to improve resolution (Aziz et al., 1996).

Sample-and-hold and sampling clock sources limitations result in an error or noise known as aperture jitter noise (Analog Devices Inc., 2002; Guan, 2012; Rapuano et al., 2005). This error is the function of the input signal slew rate and is directly affected by the frequency and amplitude of the signal (Guan, 2012). It is actually due to the input signal changing during the ADC conversion time (Fischer-Cripps, 2002) and sampling clock jitter. The sampling clock jitter will affect the SNR of the ADC negatively due to their influence on the sampling uniformity of the sample-and-hold circuitry (M. Zhang et al., 2012).

This error can be represented as:

$$\Delta V = \frac{dV}{dt} \times \Delta t \quad (2.18)$$

where ΔV is the aperture jitter error and Δt is the aperture jitter. The $\frac{dV}{dt}$ is the maximum rate of change of the input signal (Perez, 2002). It is actually the rate at which a converter can resolve the 1 LSB value of the binary output (ibid.). It can be represented as:

$$\frac{dV}{dt} = \frac{2^{-n}V_s}{T} \quad (2.19)$$

where n is the number of bits, V_s is the full-scale span of the ADC, and T is the time between conversion (ibid.). According to Guan (2012), the maximum aperture jitter, which is the sample-to-sample uncertainty, is represented by:

$$\Delta t = \frac{1}{2^n \times \pi \times F_{max}} \quad (2.20)$$

where n is the number of bits and F_{max} is the maximum input frequency. The current practice by manufactures is not to specify the aperture jitter of the integral SHA, but to specify the SNR or ENOB of the ADC (Analog Devices Inc., 2002). Kester (2005) posits that SNR or ENOB specifications indirectly include and guarantee the aperture jitter specifications.

Quantiser

For the sampled data to be of some use in a data acquisition system, the sampled signal must be quantised. This basically means the sampled signal output is assigned a digital value which represents the amplitude of the signal at that instant. In other words, it maps the discrete signal into a finite set of amplitudes represented by binary values (Kester, 2005a). It is fundamentally clear that a quantisation process is irreversible (Cvetkovic and Vetterli, 1998). Signal quantisation is inherently a nonlinear process (Al-Eryani et al., 2011; Aziz et al., 1996; Gray, 1990). This is due to limitations in representing an infinite amplitude signal with a finite number of output values (Aziz et al., 1996; Fischer-Cripps, 2002) as a result of digital code width limitations. The process results in an error known as quantisation noise. This error is due to limitations in representing quantised output amplitude with a finite number of bits (Rauth and Randal, 2005). This means not all of the theoretical infinite analogue amplitudes will be represented by the finite quantiser output. Also, those not represented will be limited by the quantisation interval (Kehtarnavaz, 2008) or steps which constitute an error, thus leading to loss of information. It can be concluded that this error is actually the difference between the input signal and the quantiser output (Leis, 2011). Furthermore, the error is also dependent on the quantisation method, whether a rounding or truncation method is used (Chen, 2004). With a truncation method, the digital output changes when sampled amplitude exceeds the LSB value; thus, the error is biased around $\frac{1}{2}$ LSB and the rounding method of the digital output changes when sampled amplitude exceeds $\frac{1}{2}$ LSB (Rauth and Randal, 2005). This then sets a quantisation error of rounding method to:

$$-Q_{LSB}/2 \leq e_q \leq Q_{LSB}/2 \quad (2.21)$$

It can be concluded that a rounding method is thus the preferred method as it gives a lower quantisation error (ibid.).

Assuming a uniform quantiser, the minimum step size or the resolution is expressed as:

$$Q_{LSB} = \frac{V_{fs}}{2^N} \quad (2.22)$$

where Q_{LSB} is the minimum step size, V_{fs} is the full-scale range or reference voltage in volts, and N is the number of bits or word length limits of the quantiser or digitiser. In literature, equation (2.22) represents the smallest ADC step which is equal to the least significant bit (LSB) value of the digitiser and is known as the resolution of the ADC. It is evident from equation (2.22) that the reference voltage (V_{fs}) is critical in obtaining the required measurement resolution. It is also evident that a noisy and unstable reference voltage (V_{fs}) will negatively affect the accuracy and resolution of the measurement.

Provided the quantisation error is random, uniformly spread across frequencies over $\pm \frac{1}{2} LSB$ and not correlated to the quantiser input (Aziz et al., 1996; Fischer-Cripps, 2002; Fowler, 2003; Kularatna, 2000; Perez, 2002), the theoretical quantisation noise is given as:

$$e_{rms} = \frac{Q_{LSB}}{\sqrt{12}} \quad (2.23)$$

It is evident from equation (2.22) and (2.23) that to reduce the quantisation noise, one must either reduce step size by increasing number of bits (N) or reducing V_{fs} . Furthermore, equation (2.23) sets the theoretical noise floor of an ADC (Rauth and Randal, 2005).

In literature, the quantisation noise can be expressed in terms of signal-to-noise (S/N) ratio:

$$S/N = 6.02N + 1.76 \text{ dB} \quad (2.24)$$

where N is the number of bits. Equation (2.24) actually represents the theoretical S/N ratio of an ideal ADC and takes into account only the quantisation error (Aziz et al., 1996; Fischer-Cripps, 2002; Fowler, 2003; Kularatna, 2000; Perez, 2002; Qaisar et al., 2013; Rauth and Randal, 2005).

It is clear from equation (2.24) that if the S/N ratio of an ADC can be measured, then N can

be computed. In literature, the computed N is known as effective number of bits (ENOB) and is given as:

$$ENOB = \frac{S/N_{measured} - 1.76}{6.02} \quad (2.25)$$

which gives the actual resolution of the ADC, thus degraded resolution.

In practice, due to ADC non-ideal characteristics caused by nonlinear transfer function, S/H, quantiser and other errors (ibid.), ENOB is given as:

$$ENOB = \frac{SINAD - 1.76 \text{ dB}}{6.02} \quad (2.26)$$

where SINAD is the signal-to-noise ratio plus distortion, which is measured from real ADC. According to Fowler (2003) and Rauth and Randal (2005), ENOB represents the accuracy of the ADC and will always be less than the resolution specified by the manufacturer.

Rauth and Randal (2005) developed a slightly modified ENOB expression that takes into account the peak-to-peak amplitude of the input signal and the full-scale range of the ADC:

$$ENOB = \frac{SINAD - \log_2(A/R) - 1.76}{6.02} \quad (2.27)$$

where A is the peak-to-peak amplitude of the input signal, and R is the full-scale range of the ADC. This equation highlights the importance of the input signal amplitude, the full-scale range and inherent nonlinearities in determining the accuracy of the ADC.

ADC Errors

There are several other errors in addition to the ones already mentioned that contribute and limit ADC performance. Depending on the type of application, some errors will be of critical importance in determining the ADC performance. According to IEEE Standard for Terminology and Test Methods for Analog-to-Digital Converters (2011), if ADC is to be used

in data acquisition systems, the following errors outlined are critical parameters in evaluating a suitable ADC:

Differential nonlinearity (DNL) error is due to the difference between the specified code bin width and the ideal code bin width divided by the ideal code bin width and with the static gain correct (IEEE Std 1241, 2011). The error arises due to the ADC output code width deviating from an ideal 1 LSB code width. The ideal step size is actually equal to 1 LSB, which is represented by equation (2.22). This ideal step size is not achievable in practical ADCs, thus resulting in a DNL error. The DNL error is represented by:

$$DNL(k) = \frac{W(k) - Q}{Q} \quad (2.28)$$

It is evident from equation (2.28) that a zero DNL error is ideal, where $W(k)$ is the actual specified code bin width and Q is the ideal code bin width (1 LSB).

It is desirable to have a DNL error that is small and has a positive value. Moreover, it represents the maximum DNL at any step change (Analog Devices Inc., 2002; Jenq, 2003). A smaller DNL error means that the actual code bin width is closer to the ideal code bin width at that specific transition step. It should be noted that a negative DNL error represents a missing code in an ADC transition step, which is highly undesirable. This means that there will be some analogue input values that are not mapped to a corresponding digital code value, hence resulting in missing codes.

The integral nonlinearity (INL) error is defined as the “*maximum difference between the ideal and measured code transition levels after correcting for static gain and offset*” (IEEE Std 1241, 2011:14). It represents maximum deviation from the straight line crossing through the midpoint of each quantisation step (Jenq, 2003; Rapuano et al., 2005).

This can be represented mathematically (Pelgrom, 2010) by:

$$INL = \frac{A(i) - i \times A_{LSB}}{A_{LSB}} \quad (2.29)$$

where $A(i)$ is the analogue input, i is the digital code width, and A_{LSB} is the resolution of the ADC, which is given by equation (2.22). It measures the straightness of the ADC transfer function (Gray, 2006); thus, the smaller the INL error, the closer the actual ADC transfer function is to the ideal ADC transfer function.

According to Jenq (2003) and Pelgrom (2010), an ideal ADC has an INL error of 0.5 LSB. This means one should not expect to get an INL error of less than 0.5 LSB in practice. However, manufacturers measure the INL error using either the best fit or endpoint method (IEEE Std 1241, 2011). It is argued that the endpoint method is a much useful method to measure the ADC INL error (Analog Devices Inc., 2002; Gray, 2006). This is because the method gives the worst-case INL error, thus being rendered useful for error budgets in measurement applications (ibid.).

Some literature define an INL error as integral (Integrated, 2002) or a summation of DNL errors (Mishra and Gamad, 2011), which suggests that an INL error will always be greater than a DNL error. Both DNL and INL are inherently fundamental errors in any ADC; thus, it is very or even impossible to correct these errors in practical applications (Kester, 2006). Another important point to note is that DNL and INL errors seem to worsen if the input signal bandwidth approaches the Nyquist rate (Kularatna, 2000). In practice, this will limit the bandwidth of signal that the DAS will acquire.

Gain error results from the slope of the transfer characteristic differing from the ideal transfer characteristic, which usually has a value of one (Rauth and Randal, 2005). Theoretically, the transfer characteristics of ADC can be represented (Analog Devices Inc., 2002; Zumbahlen, 2008) by:

$$D = K + GA \quad (2.30)$$

D represents the digital output code, A is the analogue input signal, K is the intercept or

offset, and G is the slope or gain of the transfer function. As mentioned in literature, equation (2.30) is the same as a straight line equation ($y = mx + c$). The gain error thus gives the amount or measurement of G , which deviated from the ideal value (ibid.). If the actual G is more or less than one (unity), then the difference between the ideal G and actual G is the gain error. This basically means the error represents a change in slope of the transfer function. This error may result in the premature saturation of the ADC output if the slope is steeper than the ideal slope (Gaydecki, 2004). It thus affects the full-scale resolution of the ADC, by limiting the full-scale range to be more or less than the reference voltage (ibid.). According to IEEE Standard for Terminology and Test Methods for Analog-to-Digital Converters (2011) standard, this error is defined at a specific frequency, which suggests that the gain error is frequency dependent. A user must trim or calibrate out gain error in engineering applications, and in datasheets, it is normally expressed in terms of mV, LSB or percentage of the ideal full-scale range (Analog Devices Inc., 2002; Gray, 2006).

Offset error is due to value K from equation (2.30), which differs from the ideal value. The ideal value is usually 1 LSB or $\frac{1}{2}$ LSB for a rounding quantiser (Rauth and Randal, 2005). This error is actually zero if the value or transition point that is required to cause the ADC to produce a digital code is 1 LSB or $\frac{1}{2}$ LSB. However, in practice, ADC is not exactly 1 LSB. It is clear from equation (2.30) that this error would result in the slope not passing through the origin (Gaydecki, 2004). This error is a constant and it can be easily trimmed or calibrated out by the ADC user (Analog Devices Inc., 2002). Offset error is normally expressed in terms of mV, LSB or percentage of the ideal full-scale range in datasheets (Gray, 2006). In practice, it would be desirable to choose an ADC with the lowest offset error.

Noise is one of the fundamental limitations of any electronic device. It is due to a current flow in semiconductor material (Pelgrom, 2010), which is the limitation of the material physics. Noise within ADC specified the deviation of output signal from input signal with linear errors excluded and nonlinear errors taken into consideration (IEEE Std 1241, 2011). It should be noted that this noise is obtained with the ADC output units converted to input signal units (ibid.). Within the context of ADC, intrinsic noise sources in ADC are due to the contribution

of internal circuitry (Fowler, 2003) (i.e. Op-amps, comparator, and resistors) and other ADC fundamental noise sources detailed in the IEEE Standard 1241-2011. Extrinsic noise source contributions could be due to external circuitry interfaces to ADC, which can include power supply noise (ibid.), reference voltage and signal conditioning circuitry noise.

The out-of-range recovery error basically defines how fast the ADC recovers from an input signal surge that is beyond its reference voltage or full-scale value or absolute input range (Baker, 2011; IEEE Std 1241, 2011). The recovery time is measured from the instant the input signal surge returns within the specified ADC input range (ibid.). It is clear that in practice, the aim should be to prevent an analogue input signal from reaching the absolute input range or full-scale value.

A non-ideal ADC will take finite time to respond to a change in input signal. Settling time error specifies this finite time required by the ADC output to settle its final value in response to a changing input signal (Rapuano et al., 2005). The error is measured using a step input signal, and the final value is defined to occur one second after the step input (IEEE Std 1241, 2011). The IEEE Standard 1241-2011 specifies two settling time errors: short-term settling time and long-term settling error. Short-term settling time specifies the final value when taken in less than one second. The long-term settling error is then the difference between the final value taken at one second and the final value taken at short-term settling time (ibid.). It is obvious that the step input signal is the worst-case signal which will result in the largest settling errors. Thus, it is advisable to choose an ADC with the lowest settling time in practical applications.

The full-scale step response error defines the time it takes for the ADC output code to be between the specified accuracy as the input signal is varied from low to high or high to low (Baker, 2011). By definition, it is the measurement of the response of an ADC to a fast changing input signal. It is desired that ADC should have well-behaved step responses with minimal overshoots and undershoots and with a fast response.

Crosstalk is unwanted coupling of signals between channels and is due to non-ideal effects

such as inherent undesired capacitive, inductive and conductive coupling between channels (Davis, 2006; Feucht, 2010; Peterson and Durgin, 2008). Within the context of an ADC, it specifies the channel-to-channel interference on multichannel ADCs (Baker, 2011). It is clear that in engineering applications, the aim should be to choose an ADC with a high channel-to-channel signal rejection.

As indicated in literature, the aforementioned errors are critical parameters in evaluating the performance of a data acquisition system and must be taken into consideration during the design of a DAS. There are other errors which are detailed by various authors (Analog Devices Inc., 2002; Baker, 2011; Fowler, 2003; Kester, 2006; Kularatna, 2000; Rapuano et al., 2005) that should also be taken into account during the design and implementation phase. It is assumed that their effects in degrading the performance of a data acquisition system are minimal.

ADC architectures can be divided into two main categories: high-speed ADC architectures and high-precision ADC architectures (Kook, 2011). High-speed ADC architectures are mostly used in video, communication and instrumentation, whereas high-precision ADC architectures are used in industrial and scientific measurements (Fowler, 2006; Rapuano et al., 2005). There are seven major architectures used in the industry, namely, flash converter; pipelined (sub-ranging) converter; successive-approximation converter (SAR); sigma-delta converter; integrating converter; time-interleaved converter; and folding and interpolating converter (IEEE Std 1241, 2011; Rapuano et al., 2005). The detailed description of these architectures is given in Kularatna (2003b).

With high-speed ADC architectures, the category consists of flash, pipelined and sub-ranging, and successive approximations converters (Perez, 2002). Their advantage is the speed of conversion, while their disadvantages are low resolution and high power dissipation (Fowler, 2003; Rapuano et al., 2005). Moreover, high-precision or resolution ADC architectures are dominated by a sigma-delta converter and an integrating converter (Kester, 2005a). This architecture sacrifices speed or sampling rate as the resolution is increased. Successive approximations converters can also be used in high-precision or resolution data acquisition

system, as they can provide both resolution and speed (Rapuano et al., 2005; Robert Pease, 2008). However, it is evident – and depending on application – that there will always be a trade-off between speed (sampling rate) and precision or resolution.

2.4.3.2 Signal Processing

After the analogue signal is sampled and digitised, it must be processed to be of some use. Fundamentally, signals are due to charge (electrons) movement carrying information and moving energy (Eccles, 2011) from point to point via some kind of medium.

Chen (2004) describes signal processing as a process of modifying and extracting information from a signal. In other words, signal processing helps in enhancing signal measurements and understating of signal properties (Leis, 2011) by using mathematical tools. This means an input signal from the sensor is modified or transformed and information extracted from this modified form. Mathematically, a signal can be considered a function with one or more variables – a variable which can be dependent or independent of time (Chen, 2004; Dunn, 2014; O’Shea et al., 2011). However, a signal can be viewed as one dimensional, describing a signal with varying physical quantity as a function of a single independent variable (Oshana, 2012a). It means that some variables (i.e. temperature and position) will vary with respect to time as an independent variable. It can be argued that a signal can also be multi-dimensional, with varying physical quantity as a function of two or more independent variables.

In literature, signals are classified into seven categories (Dunn, 2014; O’Shea et al., 2011):

- Analogue, discrete and digital signals
- Real and complex signal
- Periodic and non-periodic signals
- Deterministic and random signals
- Single and multi-channel signals
- Power and energy signals
- Mono-component and multi-component signals

Within the context of DAS, the analogue, discrete and digital signal category is important, as these signals are handled by such systems:

1. Analogue or continuous-time signals are signals which occur in nature and defined at all points in time (Jones and Watson, 1990) and represented by an infinite number of values (Broesch, 2008, 1997) or amplitude and resolution. Infinite amplitude and resolution are theoretical assumptions not realised in practice.
2. Discrete time signals are signals defined at a discrete instant (Meddins, 2000) or points in time. As mentioned in literature, discrete signals are an output of a sampler sampling a continuous-time signal at a specified sampling interval. In theory, their amplitude and resolution is the same as the sampled input signal, but only defined at a specific instant in time. For ease of processing, this signal must be defined at an equally spaced instant in time (Gaydecki, 2004; Jones and Watson, 1990). Mathematically, it is expressed as:

$$x_s[n] = x_s(nT) \quad (2.31)$$

$x_s[n]$ indicates a sampled continuous signal, at sampling period T .

3. Digital or binary signals are signals representing a discrete time signal amplitude or value at that time instant. Chen (2004) maintains that the digital signals are a special case of a discrete signal, but with a finite range or amplitudes. These are equivalent digital values of the sampled analogue signal values (Meddins, 2000). It is quite clear that the digital signal is the output of the quantisation process within an ADC. The ADC's main function is to transform or allocate binary numbers (Jones and Watson, 1990; Leis, 2011; Meddins, 2000) to sampled input signal. It should be noted that in practice, sampling and quantisation processes happen together within an ADC, but in theory, they can be viewed as two separate processes (ibid.). In the context of system theory, analogue to digital conversion can be viewed as a linear system, with a discrete time input signal transformed into discrete time output signal with defined system rules (Chen, 2004; O'Shea et al., 2011).

A DAS can be viewed as a linear system which consists of three linear subsystems:

- a) Signal Conditioning
- b) ADC
- c) Digital Signal Processing (DSP)

The analogue to digital conversion is thus followed by the digital signal processing stage. It is obvious that this processing is performed in the digital domain because of the sampled and digitised signal.

The DAS developed in this project will involve the transformation of 4-20mA signal generated by sensors to a voltage signal (i.e. signal conditioning), acquiring, pre-processing and transferring the raw digital data (i.e. ADC) to the host PC for storage. Thus, the main aim is to accurately acquire, pre-process, transfer and store this signal on a PC. The signal analysis performed on thi signals is beyond the scope of this research project and highly dependent on preference of the user of the DAS.

Chapter 3 Design, Testing and Calibration

Methodology

3.1 Introduction

The preceding chapter was a review of literature relevant to this study. This chapter first presents the design methodology and the test methods followed to verify the system hardware and software components. The data collection and system calibration procedures are also described in detail in this chapter. The instruments and experimental set-up are presented. The difficulty experienced in achieving high sampling rates is discussed.

3.2 Design Methodology

This project aims to develop a DAS that will acquire and process signals from multiple sensors embedded in a pilot mill. Sensors are assumed to transmit the analogue signal using 4-20mA or the 0-10V protocol. The system must thus convert the 4-20mA and 0-10V analogue signals to digital signals for further processing, storage, and displaying by a personal computer or laptop. To simplify the DAS design, the system is decomposed into three important subsystems:

1. Signal condition subsystem
2. ADC subsystem
3. Controlling and Processing subsystem

The above-mentioned is a system engineering technique in which a complex system is decomposed into smaller subsystems with distinct inputs and outputs. This approach allows for the design and verification of each subsystem independently before they are integrated together as a final system.

3.2.1 DAS System Specification

The system was developed based on the requirements listed in Table 3.1. These specifications

were developed based on the inputs from the literature review and the IEEE Standard for Digitizing Waveform Recorders (IEEE Std 1057TM-2007, 2008) requirements. This provided the minimum requirements to start with system design.

Table 3.1 DAS minimum requirements

| Parameter | Value |
|--------------------|---------|
| Input Channels | 16 |
| Bandwidth | 10kHz |
| Resolution | 10 Bits |
| Sample Rate | 40kS/s |
| Response time | 1ms |
| Input Signal Range | 4-20mA |
| Input Impedance | 100Ω |
| Input Signal Range | 0-10V |
| Input Impedance | 1MΩ |

3.2.2 Signal Conditioning Card Design

The signal condition subsystem was decomposed into four main subsystems:

1. Input stage (4-20mA Receiver)
2. Buffer Stage
3. Differential Amplifier Stage
4. Filter Stage

The signal condition circuit development was based on DAS requirements (Table 3.1) and the (International Society of Automation (ISA), 2012) ANSI/ISA-50.00.01-1975 (R2012) standard

requirements. These requirements specified the DAS channels input specifications. The specifications are listed in Table 3.2. The filter stage circuit was developed using the FilterPro filter design tool from Taxes Instruments®. The entire system circuit was designed, simulated and verified using Tina-TI V9 circuit design and simulation application.

Table 3.2 4-20mA receiver input stage requirements

| Requirement | Value | Tolerance | Notes |
|------------------|-------|--------------------|---|
| Input Resistance | 250Ω | ±0.1% or ±0.25Ω | Resistor temperature coefficient <0.01%/°C. |
| Input Current | 40mA | | Maximum current and receiver should not be damaged. |
| Input Voltage | 10V | | Maximum voltage and receiver should not be damaged. |
| Voltage drop | ±4mV | | Maximum voltage drop due to receiver input impedance. |

3.2.3 PCB Design

The signal conditioning schematics was captured with DesignSpark PCB⁴ electronic design automation (EDA) tool, and it was used to design the printed circuit board (PCB). Once the design was finalised, the design files (Gerber files) were sent to a PCB manufacturing company (www.pcbtrain.co.uk) to produce the PCB.

The bare board is shown in Appendix C. The PCB was designed using generally accepted criteria and techniques used in the electronic engineering industry. The criteria and techniques

⁴ It is a free-of-charge schematic capture and PCB layout tool for electronic design automation (EDA), for use by professional electronics design engineers, hobbyists, educators and students (<http://www.rs-online.com/designspark/electronics/eng/>)

included board stack-up scheme, grounding scheme, PCB partitioning and trace routing schemes; these techniques are described by various authors (Carter and Mancini, 2009; Grout, 2008; Zumbahlen, 2008). Once the PCB is fabricated, the components were assembled on the board, and visual inspection was performed on the board. Thereafter, the board was ready for testing. The assembled board is shown in Appendix D.

3.2.4 DAS ADC

The aim was not to design an ADC from scratch but to identify and choose the appropriate ADC architect and technology. The criteria were based on the DAS acquisition requirements listed in Table 3.1 and ADC specifications discussed in literature. In implementing the ADC and embedded system, a system-on-chip (SoC) concept was chosen. With an SoC platform, the ADC and CPU are fabricated on a single chip. This is illustrated in Figure 3.1.

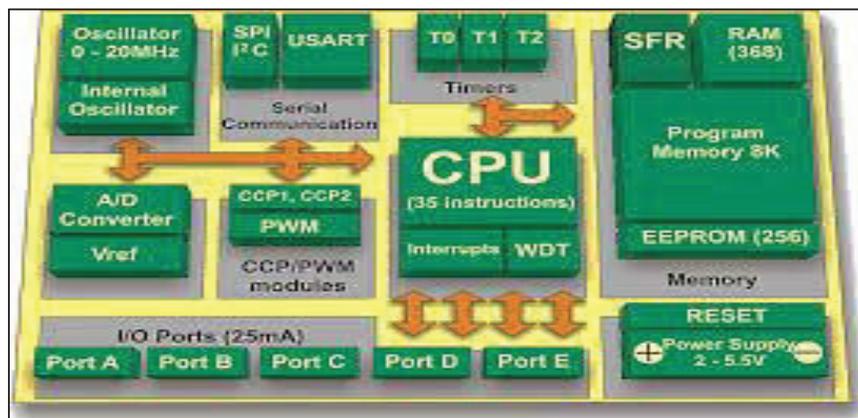


Figure 3.1 Typical μ P or μ C SoC block diagram

Source: www.mikroe.com

3.2.5 DAS Controller Embedded Platform

The controller is actually a combination of hardware and software modules. In this project, there are two controllers that will run and be implemented on two different platforms. The one platform will be implemented on an embedded system and the latter on a PC/laptop system.

There are various embedded system platforms available commercially that could have been used in the implementation. However, the choice was narrowed down to ones that support rapid development and that use the latest model-based design methodologies (i.e. Arduino®). In this project, the MATLAB® Simulink® will be used to develop and implement algorithms required by the embedded system to perform all the data acquisition, pre-processing and communication of data. The firmware was developed using model-based design, as an alternative to traditional development using high-level languages such as C or C++. This novel approach used Simulink® blocks and an SoC platform. The Simulink® environment uses blocks interconnected together, describing the algorithm for the system. This defined how the system firmware must function. These blocks were then compiled and programmed to the embedded system and executed by the target platform. This is illustrated in Figure 3.2. One of the advantages of model-based development in Simulink® is how the design of the software algorithm is modelled, and after optimising and successful simulation, the software is generated automatically and downloaded onto the target platform (Ebookaktiv, 2013). This can be done without detailed knowledge of the low-level implementation of the target hardware.

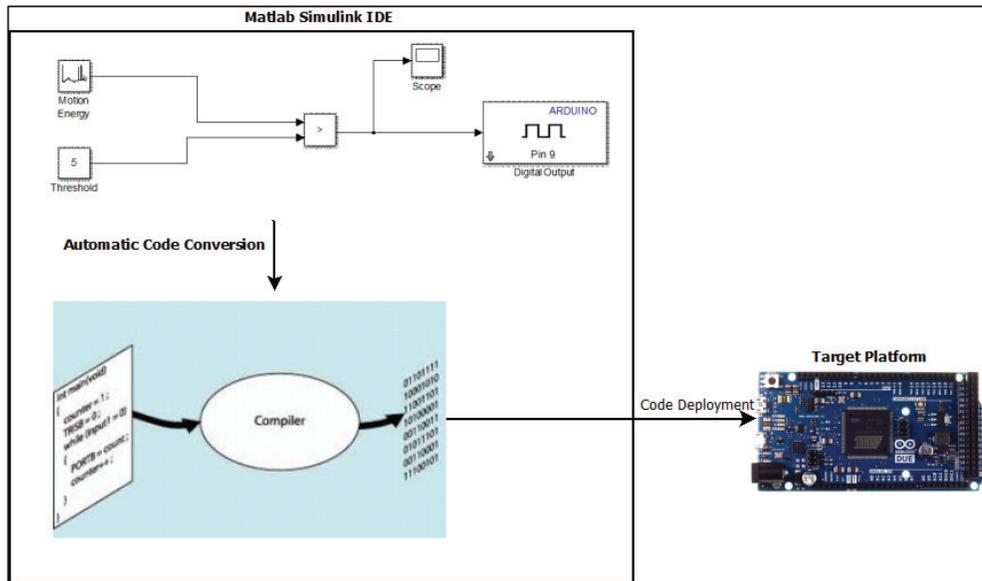


Figure 3.2 Model-based development

Initially, five embedded system platforms were chosen to be implemented in this project (see Table 3.3). The ZedBoard⁵ platform is the most advanced, complex and costly of the five platforms and can be used as a stand-alone data acquisition system, without any need of the host system. The other four platforms are reasonably priced and readily available. The Arduino⁶ platform meets requirements in terms of MATLAB® and Simulink® tools available to implement the system; this is shown in Table 3.3. The Arduino-Due platform was used in the implementation of the data acquisition system, as it meets the minimum embedded system requirements (i.e. sampling rate of 1MSPS, I/O, ADC). However, it only has 12 ADC channels accessible for external connection.

⁵ www.zedboard.org

⁶ An open-source software and hardware platform for rapid product development and electronic prototyping (<http://www.arduino.cc/>)

Table 3.3 Embedded system platform comparison

| Parameter | Arduino Mega 2560 | Arduino-Due | dsPICDEM 80-Pin Starter Development Board | STM32F4DISCOVERY | ZedBoard |
|---------------|---|--|--|--|--|
| CPU | ATmega2560 8 bit | ATSAM3X8EA-AU ARM® Cortex®-M3 revision 2.0 32 bit | dsPIC30F6014A 16 bit | STM32F407VGT6 ARM 32-bit Cortex™-M4 CPU | Xilinx® XC7Z020-1CLG484C Zynq-7000 AP SoC Programmable SoC (AP SoC dual Core-A9 Processing System (PS)) |
| RAM | 8kB | 64kB + 32kB | 8kB | 196kB | 512MB |
| FLASH | 256kB | 512kB | 144kB | 1MB | 256Mb |
| EEPROM | 4kB | N/A | 4kB | N/A | N/A |
| ADC | 16 Channels 10-bit SAR 76kSPS (max) | 12 Channels 12-bit SAR 1MSPS (max) | 16 Channels 12-bit SAR 200kSPS (max) | 16 Channels 12-bit SAR 1.2MSPS (max) | > 16 Channels 12-bit SAR > 1.5MSPS |

| Parameter | Arduino Mega 2560 | Arduino-Due | dsPICDEM 80-Pin Starter Development Board | STM32F4DISCOVERY | ZedBoard |
|------------------|---|---|--|---|--|
| Comm Port | 2 x Serial USARTs | 4 x Serial USARTs 1x USB | 2 x Serial USARTs | 6 x Serial USARTs 1 x USB (OTG) | 1x 10/100/1G Ethernet 1 x Serial USARTs 1 x USB (OTG) |
| I/O | 86 | 103 | 80 | 140 | ≥200 |
| Simulink | Yes, Simulink Support Package Available | Yes, Simulink Support Package Available | Yes, require Microchip dsPIC DSC Blockset for Simulink | Yes, Simulink Support Package Available | Yes, not supported on MATLAB and Simulink Student Suite |

| Parameter | Arduino Mega 2560 | Arduino-Due | dsPICDEM 80-Pin Starter Development Board | STM32F4DISCOVERY | ZedBoard |
|-----------|-------------------|-------------|--|--|--|
| Matlab | Yes | Yes | Yes, require Matlab add-ons Embedded coder, Matlab coder, Simulink coder and MPLAB Packages ⁷ | Yes, require Matlab add-ons Embedded Coder ⁸ , Matlab coder ⁹ , Simulink coder ¹⁰ | Yes, require Matlab add-ons Embedded Coder, Matlab coder, Simulink coder |
| CPU Clock | 16MHz | 84MHz | N/A | 120MHz | > 100Mhz |
| MIPS | 16 | N/A | 30 | 210 DMIPS ¹¹ | N/A |

3.2.6 DAS Controller Host Platform

This platform is actually a PC or laptop. The platform offers various advantages, including flexibility, low cost (depending on specifications) and multi-interconnectivity (i.e. UBS, Wireless LAN, Ethernet, and Bluetooth) (Sinclair, 2011). Due their (i.e. DAS controller host platforms) relative low cost, they can be found in most households, work environments and laboratories, therefore, making them ideal host platforms. The host system provides the required interface (i.e. GUI, processing and displaying hardware). This is shown in Figure 3.3. These PC/laptop platforms must have impressive hardware performance. They must have processors with 64-bit bus, running at 3GHz, and containing large RAM (> 4GB) and ROM

⁷ Propriety software from Microchip costing about R17 000

⁸ Not available as an add-on for MATLAB and Simulink Student Suite

⁹ Not available as an add-on for MATLAB and Simulink Student Suite

¹⁰ Not available as an add-on for MATLAB and Simulink Student Suite

¹¹ <http://e2e.ti.com/support/microcontrollers/msp430/f/166/t/188192>

(Hard drive) memory (> 500GB). They must have an operating system installed (i.e. Windows 8) and ready for customised GUI development and deployment. The GUI was developed using the MATLAB® GUIDE IDE¹² environment. The code used to process the data is developed with MATLAB® programming language.



Figure 3.3 Typical host platform

3.3 DAS Testing

A variational experimental (Dunn, 2014) approach will be employed in testing the DAS system and subsystems.

3.3.1 Instruments

The following equipment was required to perform tests: digital multimeter, digital storage scope or PC-based scope and a signal generator. The tests were performed at room temperature ($+25^{\circ}\text{C} \pm 10^{\circ}\text{C}$). The instrument list and some of their basic specifications (i.e. accuracy) are listed in Table 3.4. These instruments were purchased specifically for this project, due to their cost-effectiveness.

¹² MATLAB® Creating Graphical User Interfaces R2014b User Manual

Table 3.4 Instruments list

| Instrument | Model | Accuracy | Notes |
|---|------------------------------------|--------------------------|--|
| Digital Multi-meter | UT50B ¹³ | $\pm 0.5\% + 1$ digit | DC measurements |
| PC-based Digital Scope | 2202A ¹⁴ | $\pm 3\%$ | Of full-scale DC measurements |
| Function Generator | 2202A AWG ¹⁵ | $\pm 1\%$ | Of full-scale DC measurements |
| Spectrum Analyser | 2202A SA ¹⁶ | N/A | |
| Bench Top PSU | SP303EGC | $\pm 0.5\% + 1$ digit | Voltage and current reading display |
| Current and Voltage Loop Calibrator and Tester | 7006 Loop- Mate 1 ¹⁷ | $\pm 0.1\%$ | Of span |

3.4 Signal Conditioning Card Tests

The signal conditioning card was tested first, and all the critical parameters such as the DC voltages and signal path were measured and compared to simulation results. The signal

¹³ <http://uni-trend.com/UT50B.html>

¹⁴ Pico 2000 Series Scope Model: 2204A, SN: CZ4131230, PicoScope 6 PC Oscilloscope Software Ver: 6.10.11.15, Calibrated 18 June 2014, www.picotech.com

¹⁵ Arbitrary Wave Generator feature of Pico 2000 Series Scope Model: 2204A, SN: CZ4131230

¹⁶ Spectrum Analyser feature of Pico 2000 Series Scope Model: 2204A, SN: CZ4131230

¹⁷ <http://www.timeelectronics.com/portable-voltage-current-instruments/7006-loop-simulator-source/>

conditioning card was tested to verify its operation. All the channels were tested and verified. The following tests were performed on each channel:

- DC Test
- Signal Chain Test

These tests were performed on an assembled signal conditioning card, shown in Appendix D.

The set-up for the tests is shown in Appendix E. The bench PSU was used to provide the 24V to power the signal conditioning card.

3.4.1 DC Tests

Power Supply Unit (PSU) Ground Isolation Test

The tests start with the testing isolation between the power supply rails and return grounds. This was done with no power applied to the card. A digital multimeter was used to measure the isolation resistance between power rails. Test points were provided on the board. The test specifications are listed in Table 3.5.

Table 3.5 Power supply and ground isolation specifications

| Isolation Resistance/Impedance Requirements | Min (k Ω) |
|---|----------------------|
| +12V and Power Return Ground | 10 |
| +12V and Signal Return Ground | 10 |
| -12V and Power Return Ground | 10 |
| -12V and Signal Return Ground | 10 |

PSU Voltage Test

This test tests the power supply voltages used to power the op-amps and bias circuits implemented on the signal conditioning card. The signal conditioning card was powered by 24V.

The PSU voltages were measured with no signal applied to the inputs of the signal conditioning card channels. The specifications are shown in Table 3.6.

Table 3.6 PSU voltage test specifications with no input signal

| PSU Voltages | Min (V) | Typ (V) | Max (V) | Test Condition |
|--------------|------------|------------|------------|------------------|
| +12V | 11.50 | 12 | 12.5 | No input signals |
| -12V | -11.50 | -12 | -12.5 | No input signals |

PSU Voltage Test with DC Input

A DC voltage was applied to the inputs, and the PSU voltages were measured. A new AA battery cell was used as a DC voltage input source. This provided a clean DC voltage source. The specifications are shown in Table 3.7.

Table 3.7 PSU voltage specification with +1.5V DC input signal

| PSU Voltages | Min (V) | Typ (V) | Max (V) | Test Condition |
|--------------|------------|------------|------------|--|
| +12V | 11.50 | 12 | 12.5 | +1.5V DC input applied to the channel inputs |
| -12V | -11.50 | -12 | -12.5 | +1.5V DC input applied to the channel inputs |

PSU Voltage Test with AC Input

An AC signal was applied to the channel inputs, and the PSU voltages were measured. A signal generator was used to provide a sinusoidal signal with an amplitude of 1V at a frequency of 50Hz. The signals were applied to channel inputs. The test specifications are reflected in Table 3.8.

Table 3.8 PSU voltage specifications with 1V AC input signal

| PSU Voltages | Min (V) | Typ (V) | Max (V) | Test Condition |
|--------------|------------|------------|------------|--|
| +12V | 11.50 | 12 | 12.5 | 1V sinusoidal signal applied to the channel inputs |
| -12V | -11.50V | -12V | -12.5 | 1V sinusoidal signal applied to the channel inputs |

3.4.2 Signal Chain Tests

These tests verify the signal chain path. The input signals were applied, and output signals were measured and compared to the expected simulation and theoretical results. The set-up for the tests is shown in Appendix E. The DC measurements were made using the UT50B multimeter, due to its good DC measurement accuracy compared to the PC-based scope. The following tests were performed:

DC Offset Measurement with Inputs Grounded

The channel input was connected to ground, and the output was measured. The test conditions are shown in Table 3.9.

Table 3.9 Channel DC offset voltage with input at 0V

| Channel No. | Test Condition | Notes |
|-------------|----------------|------------------------|
| Channel 1 | Input grounded | 4-20mA channel (5kHz) |
| Channel 2 | Input grounded | 4-20mA channel (5kHz) |
| Channel 3 | Input grounded | 4-20mA channel (5kHz) |
| Channel 4 | Input grounded | 4-20mA channel (5kHz) |
| Channel 8 | Input grounded | 0-10V channel |
| Channel 9 | Input grounded | 0-10V channel |
| Channel 11 | Input grounded | 4-20mA channel (50kHz) |
| Channel 12 | Input grounded | 0-10V channel (50kHz) |

DC Offset Measurement Input Floating

The channel inputs were left open, and the outputs were measured. The test conditions are highlighted in Table 3.10.

Table 3.10 Channel DC offset voltage with input open (floating)

| Channel No. | Test Condition | Notes |
|-------------|----------------|------------------------|
| Channel 1 | Input open | 4-20mA channel (5kHz) |
| Channel 2 | Input open | 4-20mA channel (5kHz) |
| Channel 3 | Input open | 4-20mA channel (5kHz) |
| Channel 4 | Input open | 4-20mA channel (5kHz) |
| Channel 8 | Input open | 0-10V channel (5kHz) |
| Channel 9 | Input open | 0-10V channel (5kHz) |
| Channel 11 | Input open | 4-20mA channel (50kHz) |
| Channel 12 | Input open | 0-10V channel (50kHz) |

1.5V DC Input Test

A 1.5V DC voltage was applied to the input, and the output signals were measured. A new AA battery cell was used as a DC voltage input source. The cell open voltage measured was 1.604V. The actual input voltage seen by the channel was measured, and the output generated by the signal chain was also measured. The test conditions are shown in Table 3.11.

Table 3.11 Signal chain 1.5V DC input test

| Input Channel | Test Condition Actual Measured Input Voltage | Notes |
|---------------|--|---|
| Channel 1 | Input +1.5V | 4-20mA channels (5kHz) are scaled by 0.2194 |
| Channel 2 | Input +1.5V | 4-20mA channels (5kHz) are scaled by 0.2194 |
| Channel 3 | Input +1.5V | 4-20mA channels (5kHz) are scaled by 0.2194 |
| Channel 4 | Input +1.5V | 4-20mA channels (5kHz) are scaled by 0.2194 |
| Channel 8 | Input +1.5V | 0-10V channels (5kHz) are scaled by 0.2424 |
| Channel 9 | Input +1.5V | 0-10V channels (5kHz) are scaled by 0.2424 |
| Channel 11 | Input +1.5V | 4-20mA channels (50kHz) are scaled by 0.2194 |
| Channel 12 | Input +1.5V | 0-10V channels (50kHz) are scaled by 0.2424 |

AC Input Test

An AC signal set at different frequencies was applied to the channel inputs, and the output signals were measured with a scope. A signal generator was used to provide a sinusoidal signal with an amplitude of 1V, and the test signal frequencies are indicated in Table 3.12 for both 5kHz and 50kHz channels.

Table 3.12 AC input test frequency

| Channels | Test Frequency 1 | Test Frequency 2 | Test Frequency 3 | Test Frequency 4 | Test Frequency 5 |
|----------|------------------|------------------|------------------|------------------|------------------|
| 5kHz | 5Hz | 50Hz | 5kHz | 50kHz | N/A |
| 50kHz | 5Hz | 50Hz | 5kHz | 50kHz | 100kHz |

Common Mode Signal Input Test

An AC signal (sinusoidal) set at a frequency of 50Hz, with amplitude at 1V was applied to both the channel inverting (+) and non-inverting (-) inputs, and the output signals were measured.

3.5 Embedded Controller Tests

The traditional software testing techniques were used to test the DAS controller model. This included testing each sub-model and verifying each sub-model operation using simulations (i.e. external mode feature). The DAS controller model implementation was simulated using a discrete solver option with a fixed step size. This was because the controller operates in the digital domain, with no continuous states. The fixed step size was set at 1μ (0.000001) steps. A smaller step size gives better accuracy, at the expense of simulation duration (i.e. it takes longer to simulate).

The DAS controller model indicates the high-level model of the system operation and interface to the DAS signal conditioning card. The model was tuned and evaluated, running from the target hardware using external mode feature of the Simulink[®] environment. In this mode, the model was compiled, loaded and executed on the Arduino[®] Due platform. The platform then runs the model in real time, and this allows the model to read and see what the target hardware was reporting. The Simulink[®] sink library blocks (i.e. display, scopes, simout, and others) are then used to display data reported by the target hardware. This was made possible by the server that was loaded automatically by Simulink[®] on the model running on the target hardware in external mode. This is described as hardware-in-the-loop simulation

(Xue and Chen, 2013). The data collected was compared and verified with multimeter measurements. The test set-up consists of the Arduino[®] Due platform, with channel inputs injected with DC voltage 0V, 1.5V and 3.3V. Measurements taken by the model were compared with measurements taken by the digital multimeter. The test set-up is shown in Figure 3.4.

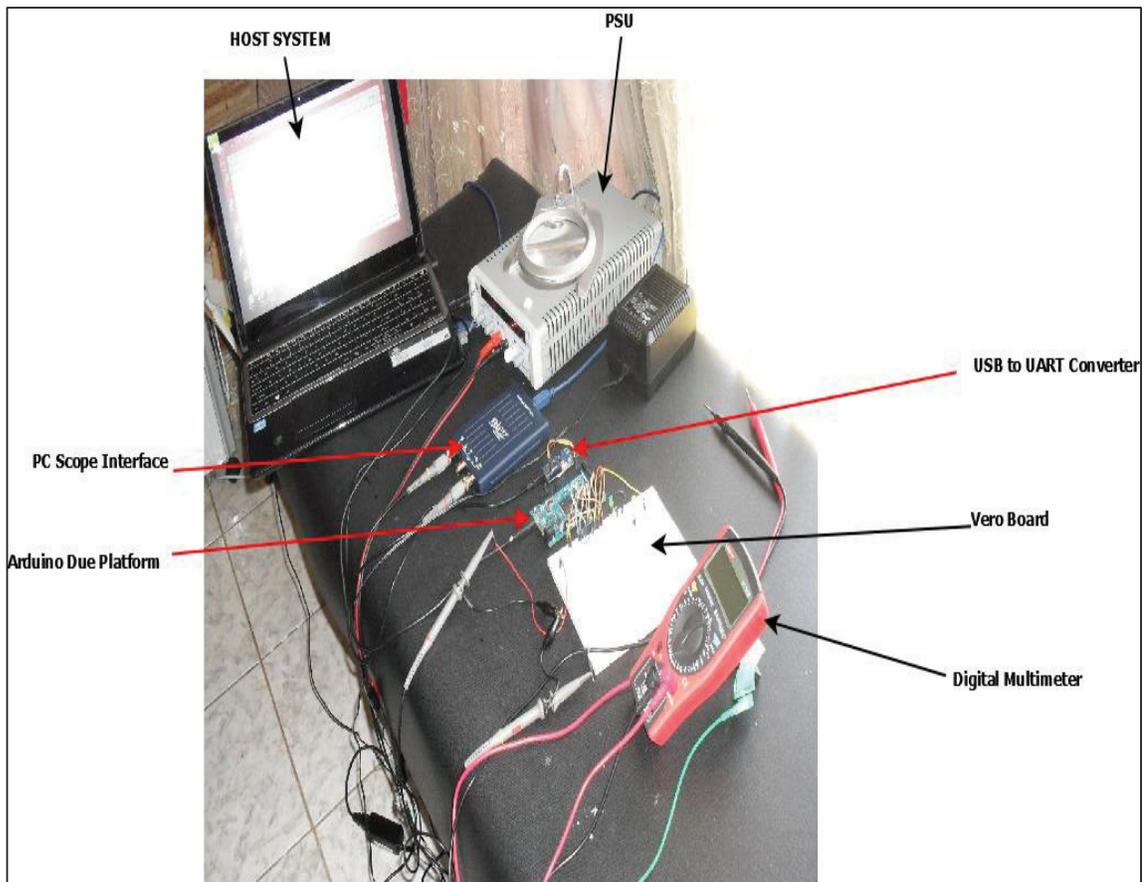


Figure 3.4 DAS controller model test set-up

The DAS controller model was made of subsystems which perform different tasks to accomplish the data acquisition process. The DAS controller model is depicted in Figure 3.5.

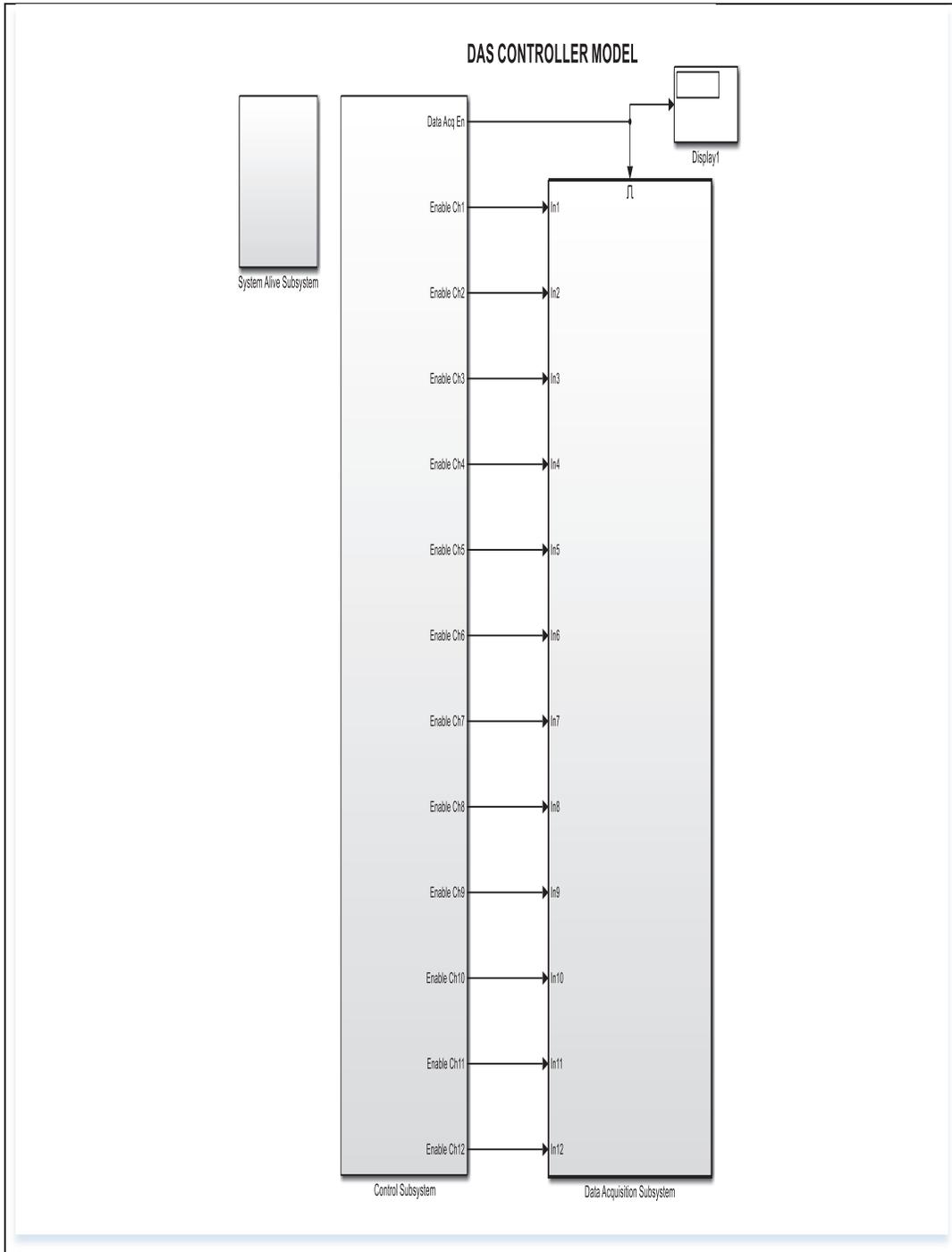


Figure 3.5 DAS controller model

The vector concatenate block was used to pack the measured data into a 12 x 1 vector matrix. The data is packed into contiguous locations in memory. Therefore, the output of the block is a contiguous output signal with 12 elements containing the measured signals. The data format sent by this model through the serial port is indicated in Figure 3.6.

Data Transmission subsystem is used to transmit data through the serial port to the host system. The Arduino® Due hardware platform provides access to two serial ports. Port1 interface is shown in Figure 3.7. The signal generated and received on the serial port RX1 and TX1 pins are at 3.3V transistor-transistor logic (TTL) levels. A signal level shifter or UBS-to-Serial converter (see Figure 3.4) was required to directly connect the Arduino® Due serial port pins and the host system serial port.

One of these subsystems is the Data Acquisition subsystem. The Data Acquisition subsystem model is highlighted in Figure 3.8. It consists of 12 inputs and one enable input. It contains other subsystems and MATLAB® function blocks to process acquired data. It contains the ADC Channels, Relay Data, Data Formatting and Data Transmission subsystems.

The enable input was used to control the Data Acquisition subsystem block. The whole subsystem can be enabled or disabled. The display and simout blocks are used to view, debug and verify data collected by the model.

The ADC Channels subsystem's task was to read analogue inputs and pack sampled data. It was made of 12 analogue input blocks, which measure signals connected to channel A0 to A11 on the Arduino® Due hardware platform and the vector concatenate¹⁸ block. The ADC Channels subsystem model is shown in Figure 3.9. The display and simout blocks are used to view, debug and verify data collected by the model. The Arduino® Due hardware platform illustrating the ADC channels is shown in Figure 3.10.

¹⁸ <https://www.mathworks.com/help/simulink/slref/vectorconcatenate.html>

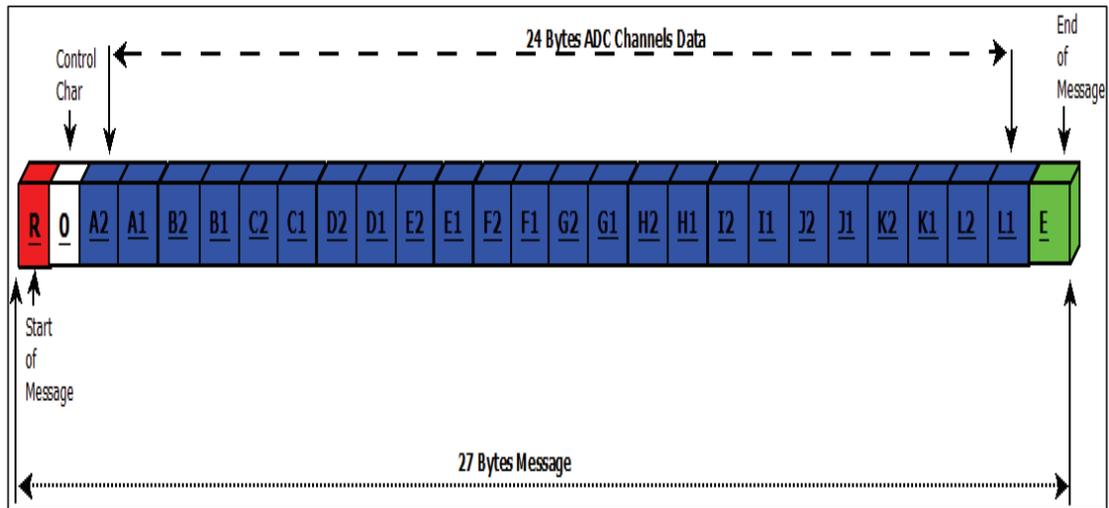


Figure 3.6 Data message structure

The control subsystem's task was to control the data acquisition process. The model receives a command sent by the host system, then decodes the command and executes the instruction.

The model consists of Arduino® serial receiver block¹⁹, a Command processing and Control signals subsystems. The control subsystem model can be seen in Figure 3.12. A

HyperTerminal application was used to send predefined command messages (Figure 6.9 and Figure 6.10) to the control subsystem model. The display and simout blocks were used to view, debug and verify data collected by the model.

¹⁹ <http://www.mathworks.com/help/supportpkg/arduino/ug/serialreceive.html>

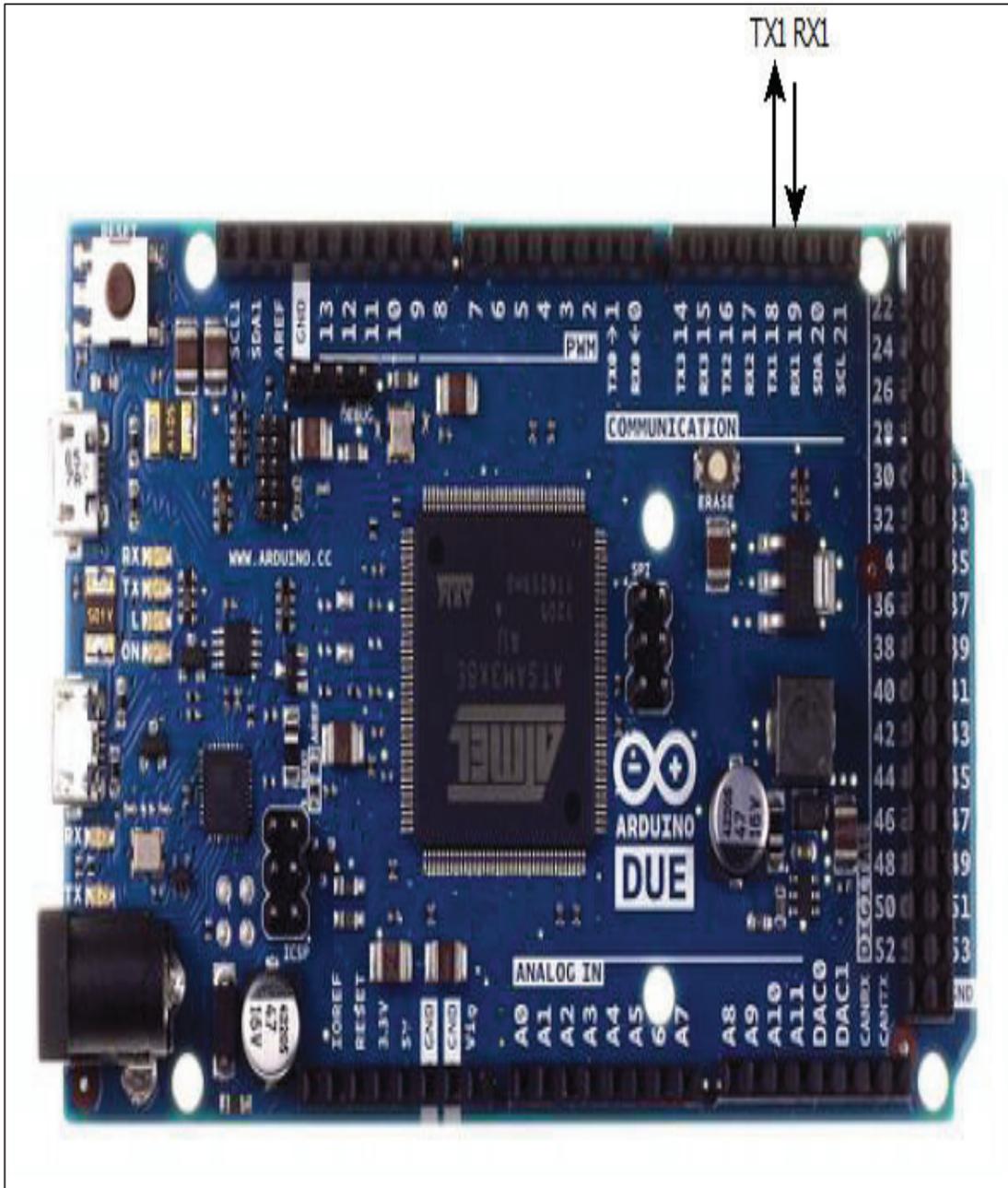


Figure 3.7 Serial port interface

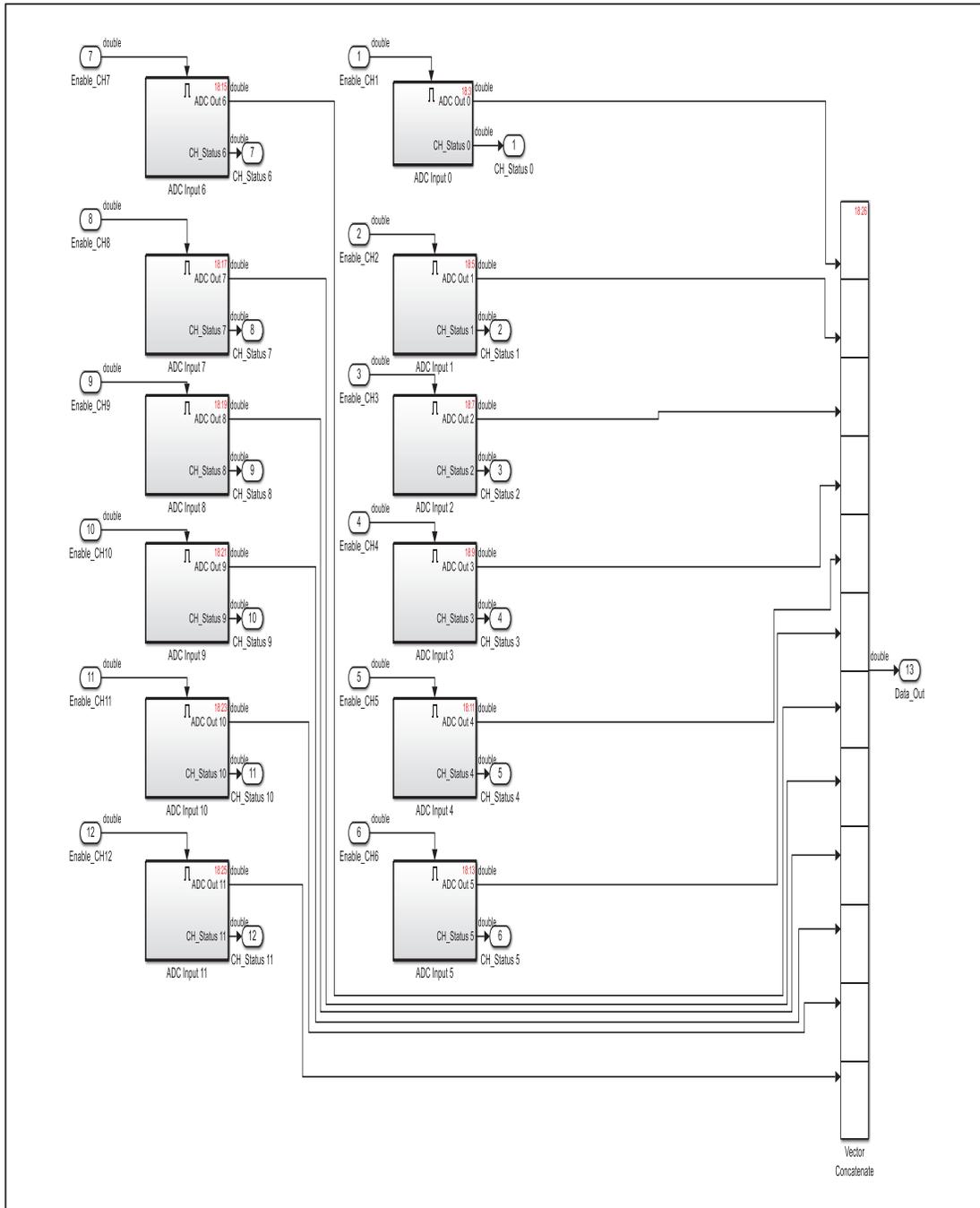


Figure 3.9 ADC channels subsystem

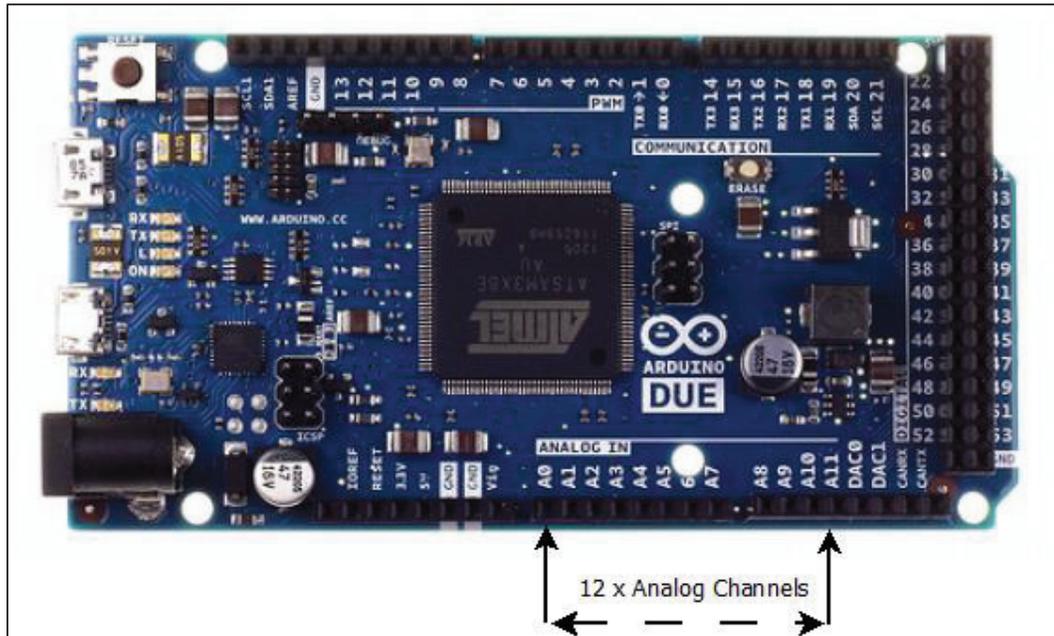


Figure 3.10 Arduino® Due hardware platform ADC channels

The System Alive subsystem was tested by measuring the signal generated on pin 51. This is shown in Figure 3.11

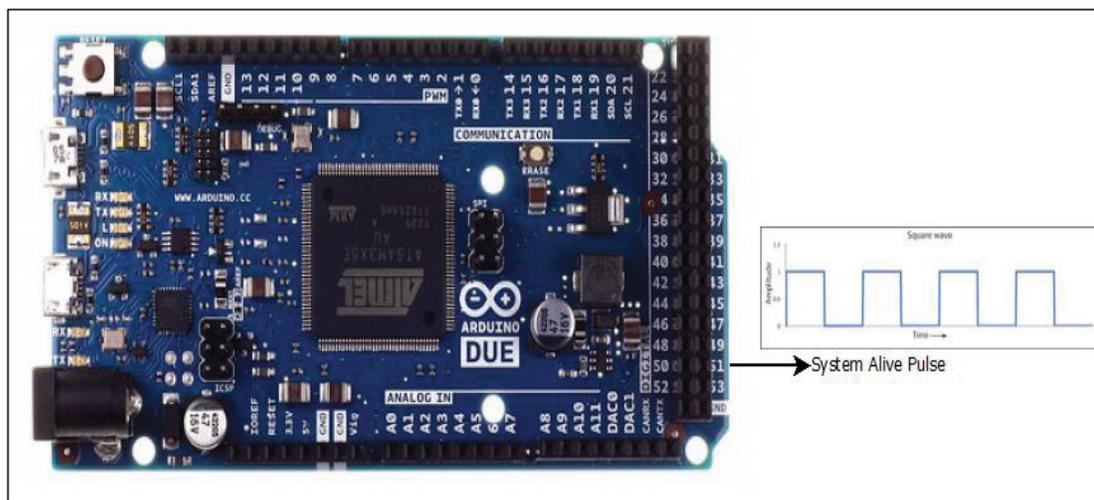


Figure 3.11 System Alive interface

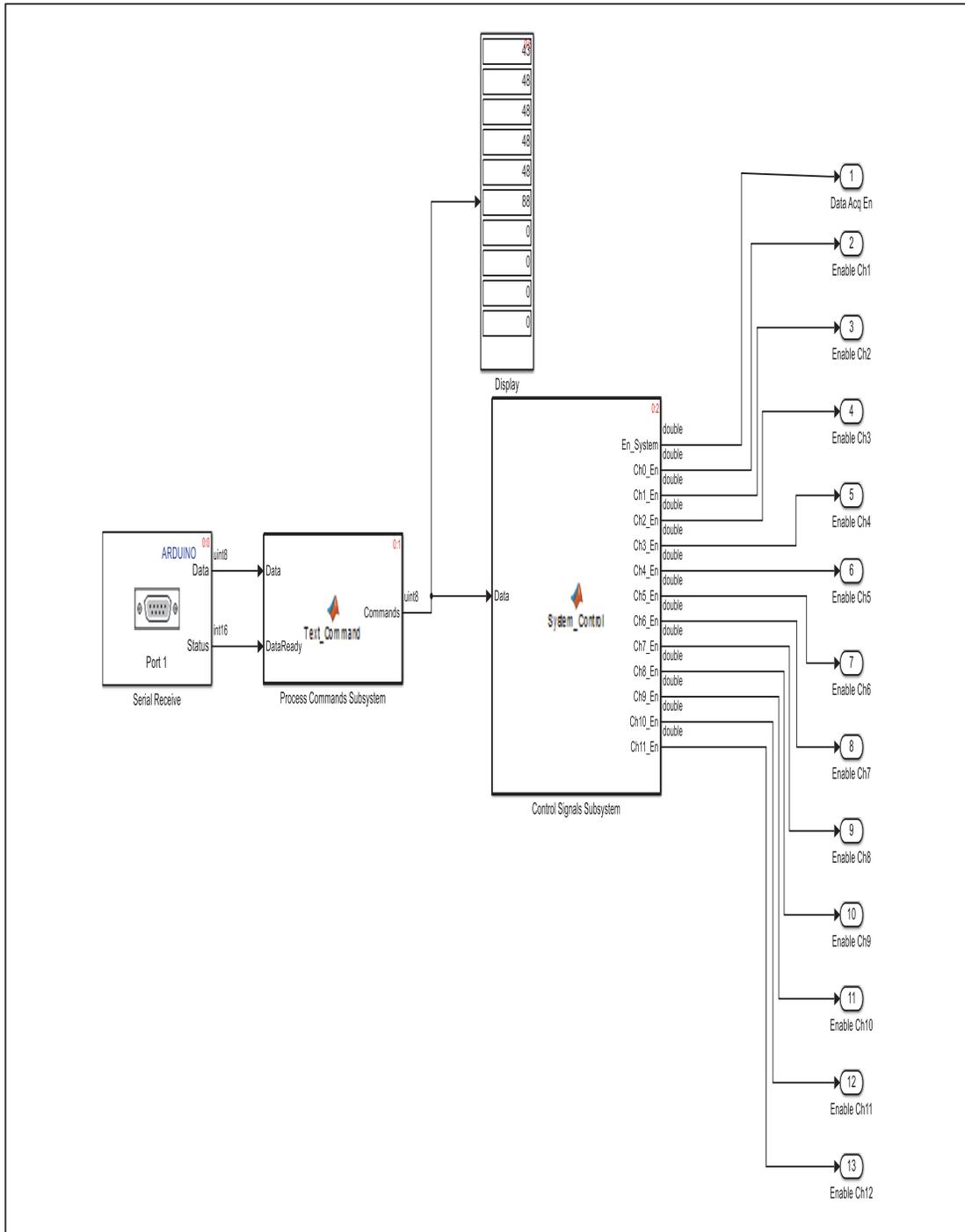


Figure 3.12 Control subsystem model

3.6 Host Controller Tests

The host system consists of a laptop/PC and GUI designed to interact with the DAS controller. Traditional software testing techniques are used to test the DAS Host controller application. This included testing each sub-module and verifying each sub-module's operation using simulations. The components making up the DAS host user interface and layout are shown in Figure 3.13. The functions of each user interface (UI) components were tested using the set-up shown in Figure 3.4.

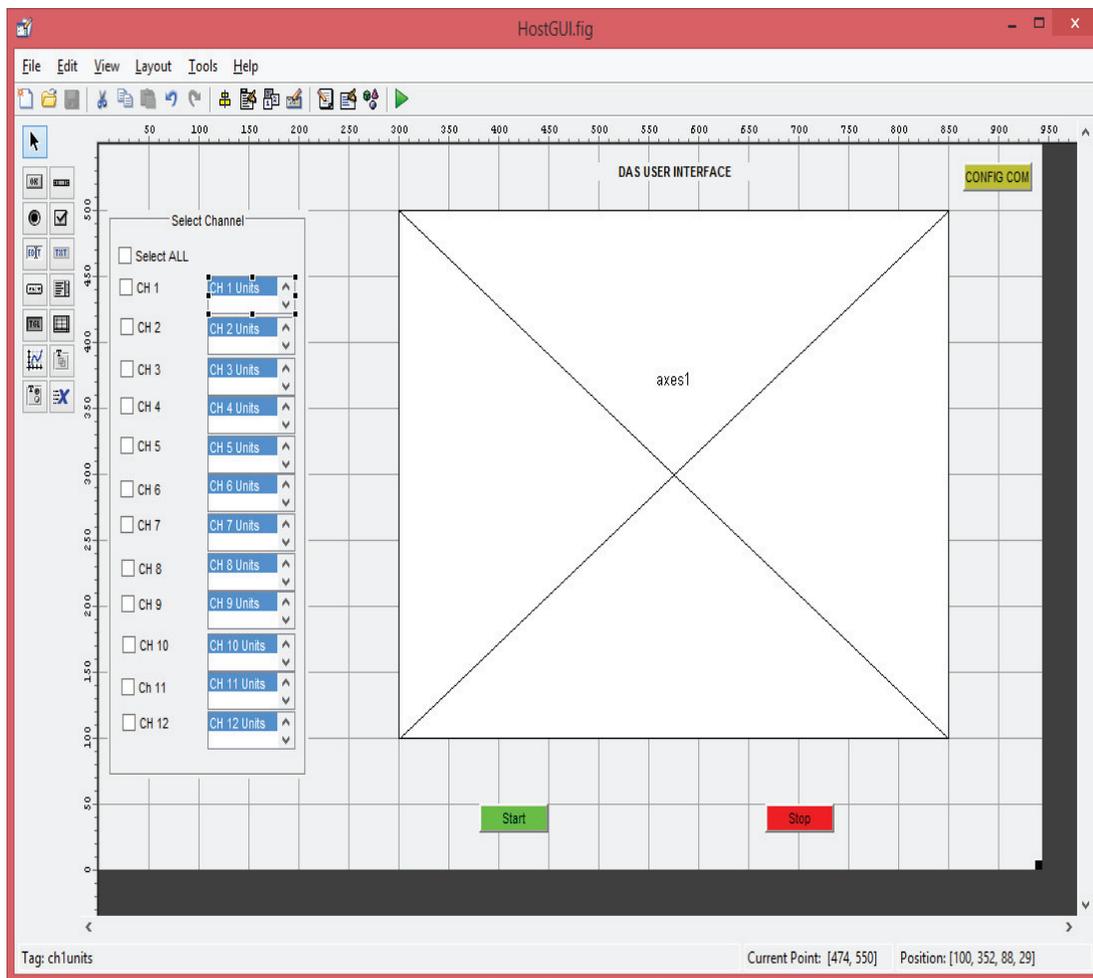


Figure 3.13 DAS user interface components layout

First, all buttons and checklist modules were tested. The host system UI generates events as the components were clicked or selected. These events contain algorithms that will perform operations on the data and generate messages to provide information to the user. These modules were tested using the simulation and debug feature of the MATLAB® GUI programming environment.

The command and communication modules were then tested. The DAS controller receives instructions from the host in response to events generated by the start or stop buttons. The DAS controller responds to the instruction and then sends data requested by the host system. These modules are tested using the simulation and debug feature of the MATLAB® GUI programming environment with USB-Serial Converter connected to the host system. This enabled the application to open the serial port used to send or receive data. A HyperTerminal application was used to send commands to the host controller GUI to evoke events generated by data received on the serial port. The HyperTerminal application was also used to verify data sent by the host controller GUI. This allowed verification of the commands structure Figure 3.14.

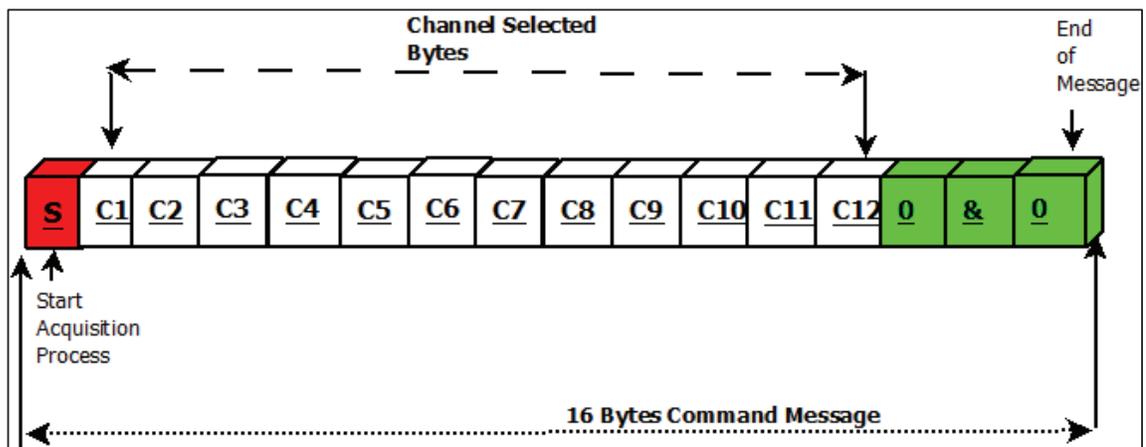


Figure 3.14 Start of data acquisition process command message structure

As already indicated, the DAS controller receives instructions from the host in response to events generated by the start or stop buttons. Thereafter, the DAS controller responds to the

instruction and then sends data requested by the host system Figure 3.15.

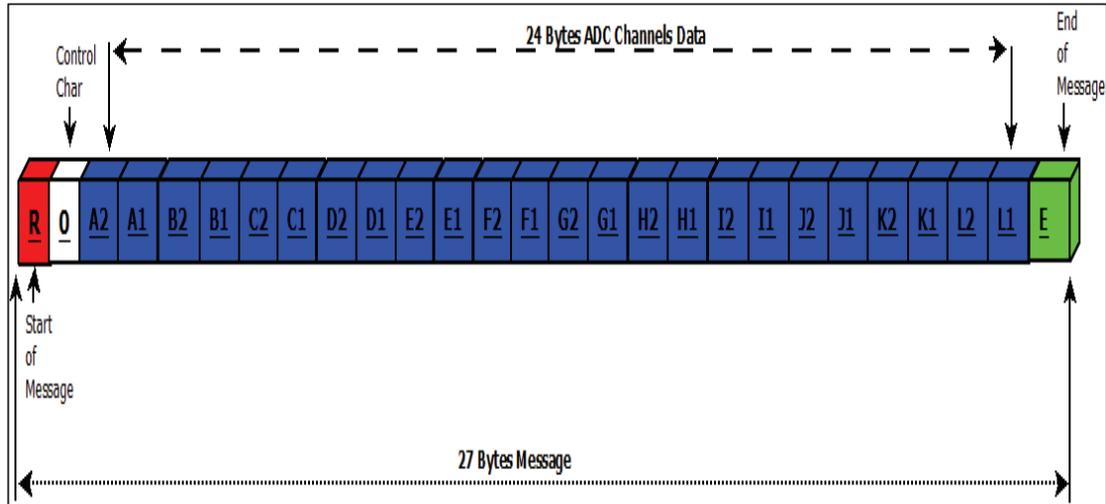


Figure 3.15 Data message structure

The data storage and displaying modules were tested last. A HyperTerminal application was used to send data to the host controller GUI to evoke events used to plot the captured data on the axes component and the storage of data in a text file. The file was saved in text format, with data separated by commas. The data was saved in the following format: first number was the sample number, which was followed by a comma and the actual processed ADC data (equation 3.1) in fixed-point notation form, with six points precision.

$$V_{out} = \left(\frac{V_{ref} * ADC_{output}}{FS} \right) * Scale_{factor} \quad (3.1)$$

3.7 ADC Tests

These tests are performed to verify the measured data and the accuracy of data provided by the ADC on the Arduino® Due hardware platform, as implemented on the DAS controller model. The tests are performed with only the Arduino® Due hardware platform, with test signals applied directly to the ADC channels. The test set-up is shown in Figure 3.4. The

following tests were performed: ADC DC and AC tests. The tests require a functional DAS Embedded and Host Controller.

3.7.1 ADC DC Test

A DC voltage was applied to the inputs, and ADC outputs were captured and stored in a text file using the host GUI application. A new AA battery cell was used to provide +1.5V DC voltage input source. Next, the inputs were then grounded (0V) and ADC outputs were captured and stored in a text file. Lastly, the +3.3V DC voltage was applied to the inputs, and ADC outputs were captured and stored in a text file. A digital scope and multimeter were used to measure the input test signals. The file can be opened in an Excel spreadsheet to plot and analyse the data.

3.7.2 ADC AC Test

An AC signal set at different frequencies was applied to the channel inputs, and the output signals were measured. A signal generator was used to provide a sinusoidal signal with an amplitude of 250mV, 500mV and 1V at a frequency of 1Hz, 5Hz, 10Hz, 500Hz, 5kHz and 50kHz. The DAS Host GUI program was used for ADC data capturing. These signals were applied to the ADC inputs, and the ADC output were captured and stored in a text file. The files can be opened in an Excel spreadsheet to plot and analyse the captured data.

3.8 System Calibration

The system was calibrated to quantify system errors (static errors). A standard input signal was applied to the system inputs, and the outputs of the system were then measured and data collected and saved on a file. The raw data obtained from the system averaged over 50 data points. The scope was used to measure the output signals for comparison (the scope was the reference measuring instrument). The calibration was performed on a complete integrated DAS system. The calibration set-up is shown in Figure 3.16. The standard calibration signals were obtained from the current loop and voltage simulator, the 7006 Loop-Mate 1. It can generate the required 4-20mA loop currents and 0-10V loop voltage signals. The system

calibration was limited to using static and sequential calibration techniques and only performing up-scale calibration (Dunn, 2014).

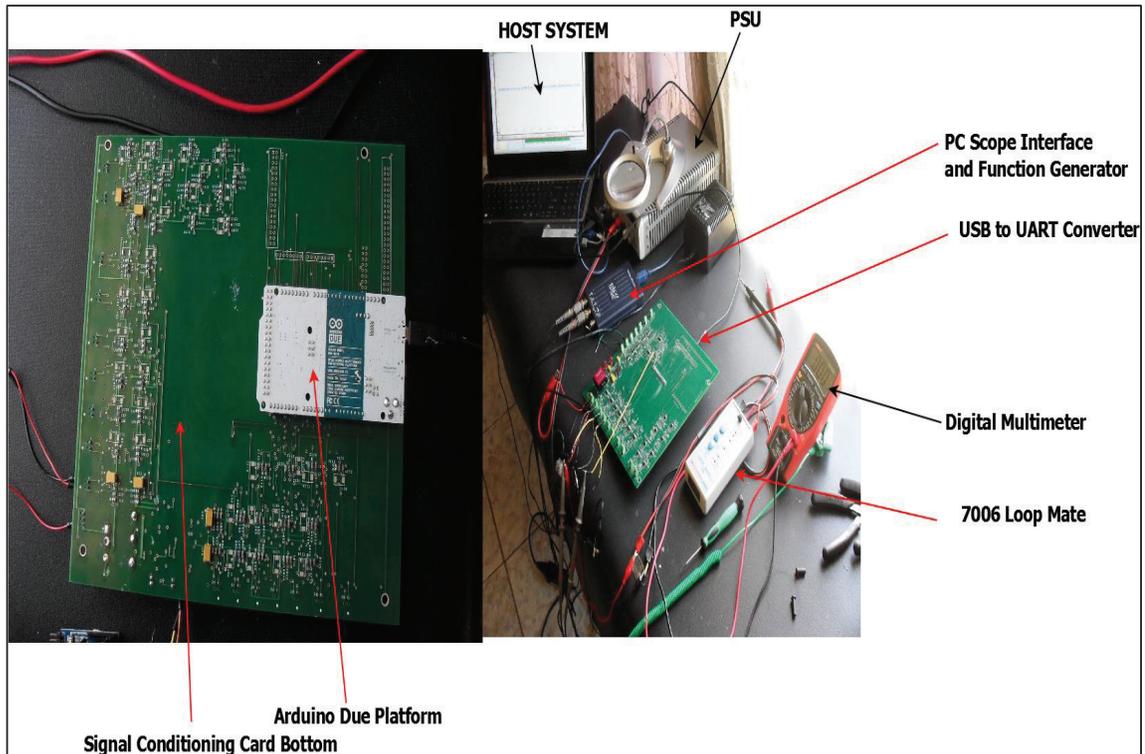


Figure 3.16 DAS system calibration set-up

3.8.1 Procedures

First, the current loop simulator was used to manually step up through seven set points listed in Table 3.13; this signal is applied to 4-20mA channels. The scope was used to measure the output signal of each channel, and this data was recorded. The DAS Host GUI program was used for ADC data capturing, and the data was stored onto a text file. The file can be opened in an Excel spreadsheet to plot and analyse the data.

Table 3.13 Current calibration points

| Cal Point 1 | Cal Point 2 | Cal Point 3 | Cal Point 4 | Cal Point 5 | Cal Point 6 | Cal Point 7 |
|----------------|----------------|----------------|----------------|----------------|-----------------|----------------|
| 0% (4mA) | 10% (5.6mA) | 25% (8mA) | 50% (12mA) | 75% (16mA) | 90% (18.4mA) | 100% (20mA) |

The voltage simulator was used to manually step up through seven set points listed in Table 3.14. These signals were applied to 0-10V channels. The scope was used to measure the output signal of each channel, and this data was recorded. The DAS Host GUI program was used for ADC data capturing, and the data was stored on a file. The file can be open on an Excel spreadsheet to plot and analyse the data.

Table 3.14 Voltage calibration points

| Cal Point 1 | Cal Point 2 | Cal Point 3 | Cal Point 4 | Cal Point 5 | Cal Point 6 | Cal Point 7 |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| 0% (0V) | 10% (1V) | 25% (2.5V) | 50% (5V) | 75% (7.5V) | 90% (9V) | 100% (10V) |

3.9 Difficulties Encountered

There were challenges in obtaining the required real-time processor speed (96MHz) on the Arduino® Due hardware platform. This affected the maximum sampling rate negatively. It also limited the input signal bandwidth severely. A considerable amount of time was spent trying to solve this limitation, however, with no success. Due to the limited sampling rate, test signals bandwidth were limited to a maximum of 2Hz, which will assist in reducing measurement errors.

Chapter 4 System Description and Model

4.1 Introduction

The research methodology employed in this study was discussed in the foregoing chapter. The focus of this chapter is to describe the function of each DAS subsystem, and model these subsystems. Their working principles and characteristics are presented.

4.2 DAS System

A data acquisition system is basically a digital recorder, the main purpose of which is to acquire analogue signals from multiple input channels and transform them into digital signals for storage, viewing and analysis. The values indicated in Table 3.1 are bare minimum requirements of the system. However, the input signal ranges are specified to limit system input range capabilities. If such limits are omitted, it will mean in theory the system is capable of measuring an infinity input range, which obviously is not achievable in practice, hence is a theoretical concept.

The system can be viewed as a linear system consisting of linear subsystems interconnected together. This is illustrated in Figure 4.1. Each subsystem will have distinct inputs and outputs. The DAS system consists of the following subsystems:

- Signal Conditioning Card
- ADC
- Controller

The host system is used to control the data acquisition system. The PSU is an external bench power supply used to power the signal conditioning card. The ADC and Controller subsystems are fabricated into one integrated circuit (IC) for practical application (see Figure 3.1).

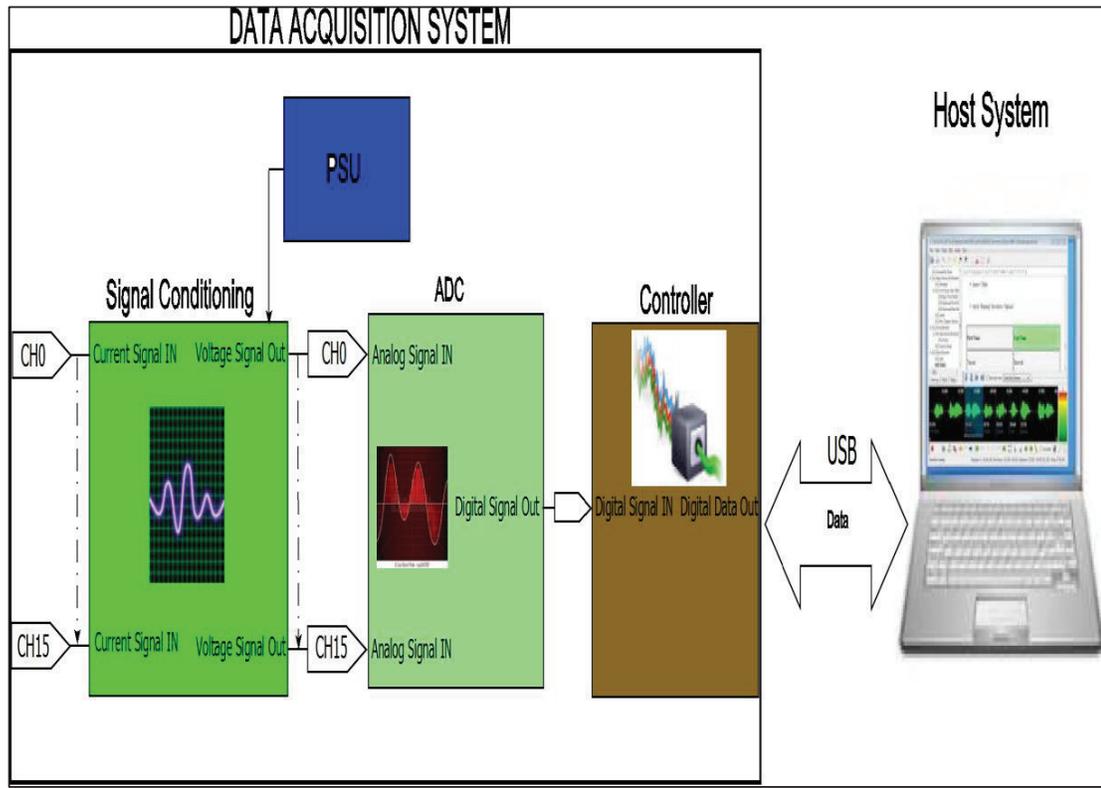


Figure 4.1 System level DAS diagram

4.3 Signal Conditioning

The main purpose of this subsystem is to pass signals generated by sensors to the ADC subsystem for conversion to digital form. These signals are first converted to voltage signals if the input signal is in 4-20mA form. They are then filtered (bandlimited) and scaled to the correct level for the ADC inputs. The 0-10V signals are scaled and bandlimited only before they are fed to ADC inputs.

4.3.1 Parameters to Be Measured

In mill measurements, the following parameters (measurands) are usually measured to study the mill load behaviour:

- Conductivity

- Displacement
- Force
- Flow Rate
- Load Mass
- Mill Speed
- Torque
- Vibration
- Viscosity

These sensor subsystems usually contain electronic circuitry which converts the measurand to a 4-20mA or 0-10V analogue current signal. The analogue signal is directly proportional to the measurand. The sensor analogue output signal must be compatible and meet the requirements of the ANSI/ISA-50.00.01-1975 (R2012) Compatibility of Analogue Signals for Electronic Industrial Process Instruments (International Society of Automation (ISA), 2012). This means that the 4-20mA transmitter of such devices will be designed to meet this requirement by the manufacturer.

For design and modelling purposes, this sensor can be modelled as a variable current source, with a current span of 16mA. The current span (16mA) represents the entire span of the input stimuli and with zero measurands represented by 4mA and a maximum measurand by 20mA respectively (Fraden, 2010). It should be emphasised that the analogue signal transmits information between the two subsystems, which are a sensor and DAS receiver respectively. However, this information transmission is only one way, from the sensor to the DAS receiver.

One of the important characteristics of a sensor is its bandwidth. This parameter is important as it determines the channel and overall system bandwidth; it is strictly a function of the stimulus or measurand (Coombs Jr, 2000). The characteristics of sensors are obtained from manufacturers' datasheets, and in this research, the interest is more on the subset of sensor characteristics which directly influence DAS performance and specifications. This subset includes resolution, accuracy and bandwidth. With that said, it is obvious that DAS accuracy,

resolution and bandwidth must be better than that of the sensors in order to reliably capture signals.

Table 4.1 lists some of the high-bandwidth sensors that will influence the performance of DAS.

Table 4.1 Typical mill sensor bandwidth

| Sensor Type | Parameter | Maximum Value | Units | Comments |
|--------------|-----------|---------------|-------|--|
| Vibration | Bandwidth | 50 000 | Hz | Depends on the accelerometer type (Wilson, 2005) |
| Displacement | Bandwidth | 40 000 | Hz | Depends on the sensing method (Fraden, 2010) |

It is evident that in the context of mill measurements, vibrations measurements will require high-bandwidth channels, compared to other parameters. This will also demand high sampling rates that are greater than 120 kilo Samples per Second (kSPS) for such channels. Force, torque and mass are closely related quantities which are governed by Newton's laws of motion. This means one or more of these parameters can be measured and the remaining parameter can be computed, applying Newton's laws of motion.

4.3.2 Sensor 4-20mA Transmitter Model

A 4-20mA transmitter is basically a current source device. An ideal current source is characterised by infinite output impedances, thus not influenced by load variations. However, in practice, a current source will have very high finite output impedances, which will slightly be influenced by load variations, thus affecting output current. This influence can be seen as noise generated by the current source. The ANSI/ISA-50.00.01-1975 (R2012) standard sets the limits of the peak-to-peak ripple and total noise level of the current source to a maximum of 0.25% at maximum signal (20mA). Thus, the noise level should not exceed 50 μ A.

However, the ANSI/ISA-50.00.01-1975 (R2012) standard does not specify the temperature conditions and minimum signal limits. It is assumed in this project that noise levels are valid across the operating temperature range of the transmitter. The noise levels should be smaller at the minimum signal (4mA) levels, provided the same criterion is applied. Accordingly, noise levels at minimum signal levels should not exceed 10 μ A.

The 4-20mA transmitter is used to transmit a signal which is proportional to the measurand or stimulus. With this in mind, a model of such sensor is developed.

The 4-20mA transmitter can be viewed as a controlled current source, with output current dependent on the input stimulus (see Figure 4.2). Because this dependency is linear, the model will have output characteristics governed by equation (2.5). As stated in the ANSI/ISA-50.00.01-1975 (R2012) standard, the minimum input stimulus is represented by 4mA and the maximum is represented by 20mA. This gives an output span of 16mA. These requirements can be represented by the following equations:

$$y_1 = mx_1 + c_1 \quad (4.1)$$

$$y_2 = mx_2 + c_2 \quad (4.2)$$

where $y_1 = 4 \text{ mA}$ at x_1 and $y_2 = 20 \text{ mA}$ at x_2 . The goal is to solve for parameters m , c_1 and c_2 which will satisfy the two equations. Assuming $c_1 = c_2$ and by subtracting the two equations (4.2) and (4.1), the following slope or gradient is derived:

$$m = \frac{16}{x_2 - x_1} \quad (4.3)$$

With the initial assumption of $c_1 = c_2$ still applying, the two equations (4.2) and (4.1) are added and substituting equation (4.3) to get the intercept c :

$$c = 12 - 8 \left(\frac{x_2 + x_1}{x_2 - x_1} \right) \quad (4.4)$$

Therefore, the output current of the 4-20mA transmitter can be represented by the following

mathematical model:

$$I_o = \frac{16}{x_2 - x_1} S_i + 8 \left[\frac{3}{2} - \left(\frac{x_2 + x_1}{x_2 - x_1} \right) \right] \quad (4.5)$$

where I_o is the output current, S_i is the input stimulus or measurand, x_1 and x_2 are the lower and upper limits of the measurand, thus the measurement range (span) of the sensors. It is apparent that sensor range is an important parameter in any measuring application; this is clearly indicated by the model. It can be seen from the model that the measuring range of the sensor has a major influence on the slope (sensitivity) and offset. The model does not show the mechanics of how the measurand is transformed into current. Also, the model does not include temperature and any other environmental influence on the output current; thus, these are assumed to be negligible.

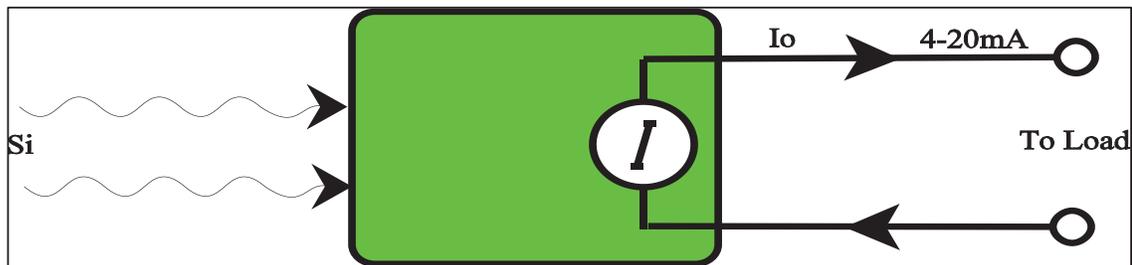


Figure 4.2 Sensor with 4-20mA transmitter model

4.3.3 Simulink 4-20mA Transmitter Model

The Simulink® model developed from equation (4.5) is shown in Figure 4.3; it is used to verify the mathematical model. The sinusoidal signal is used as the input to the model (In1) and the sensor range $x_1 = -100$ and $x_2 = 100$ as other two inputs (In2 and In3). The subsystem model with inputs is presented in Figure 4.4.

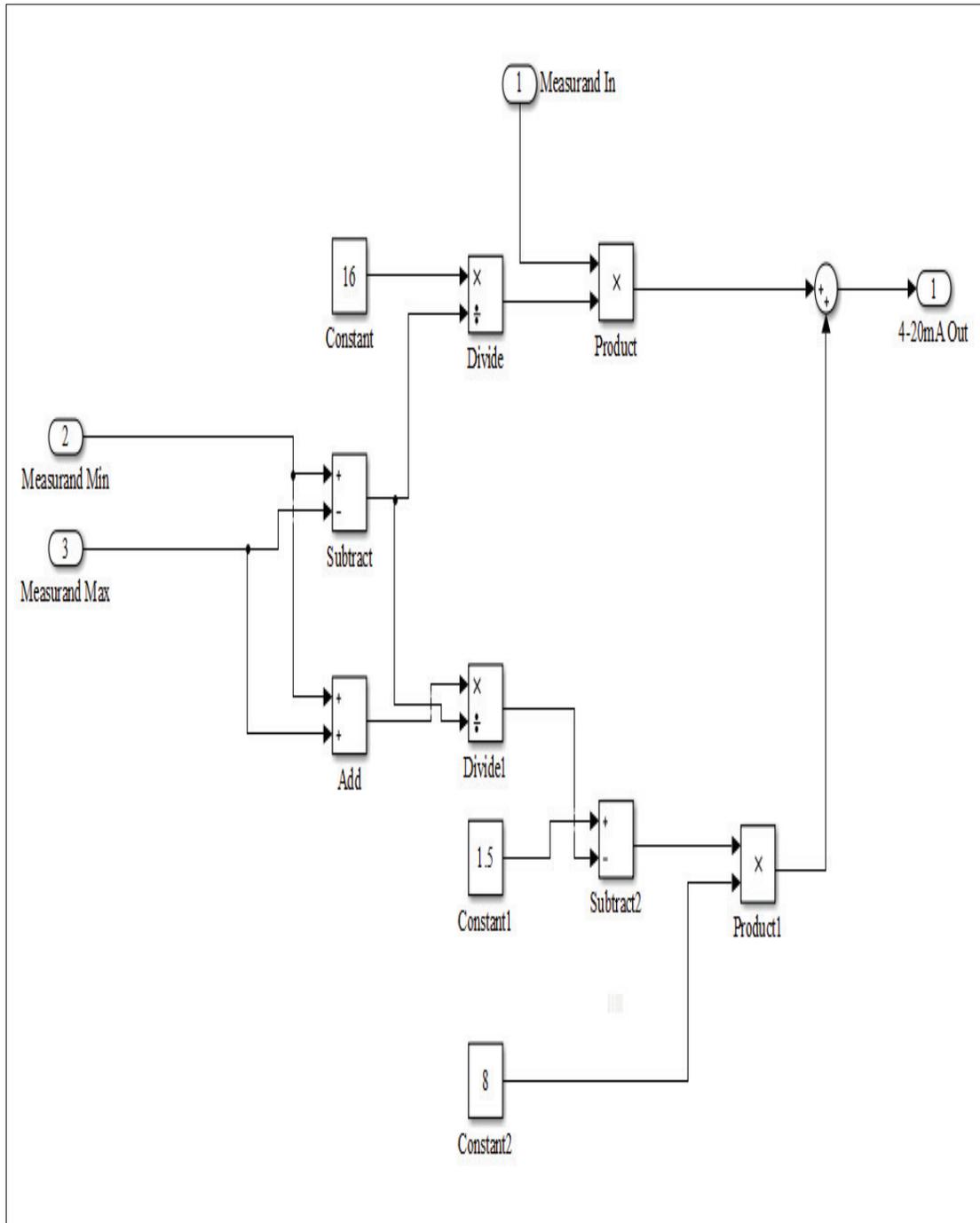


Figure 4.3 Simulink model for sensor 4-20mA transmitter

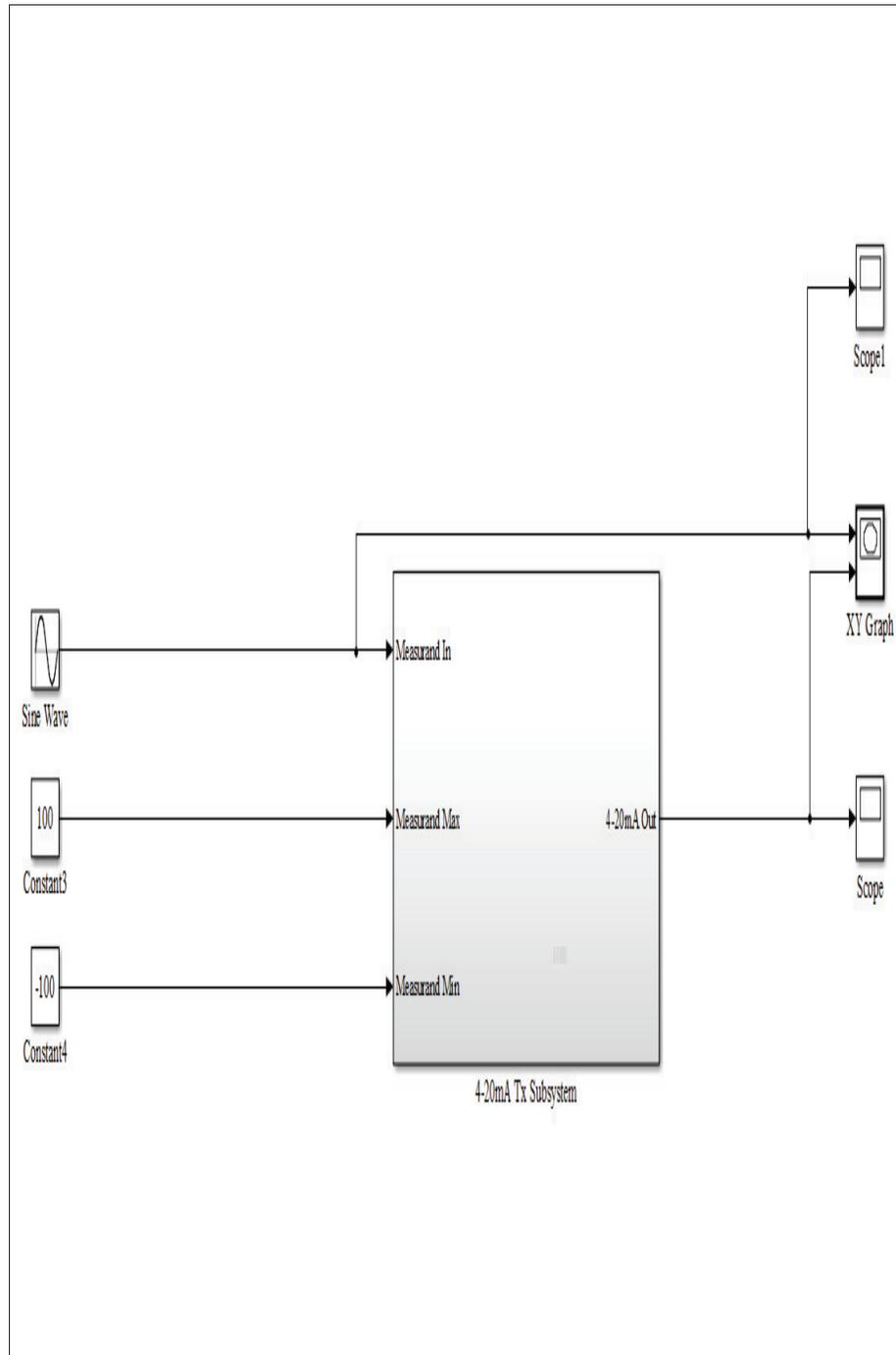


Figure 4.4 Simulation set-up for 4-20mA transmitter model

The results of the simulation are indicated in Figure 4.5 and Figure 4.6, showing that the output current has a linear relationship with the input signal. However, if the sensor input range is outside of the model measurement range (measurand max – measurand min), then the simulation results in this case show an erroneous output current (Figure 4.7). The Simulink model developed from mathematical equation (4.5) will generate an erroneous output current if the input signal is out of range. This is comparable to sensors used in practice, in which accuracy is guaranteed over specified input range. This validates the idea that the mathematical model can be used to model a sensor with a 4-20mA output.

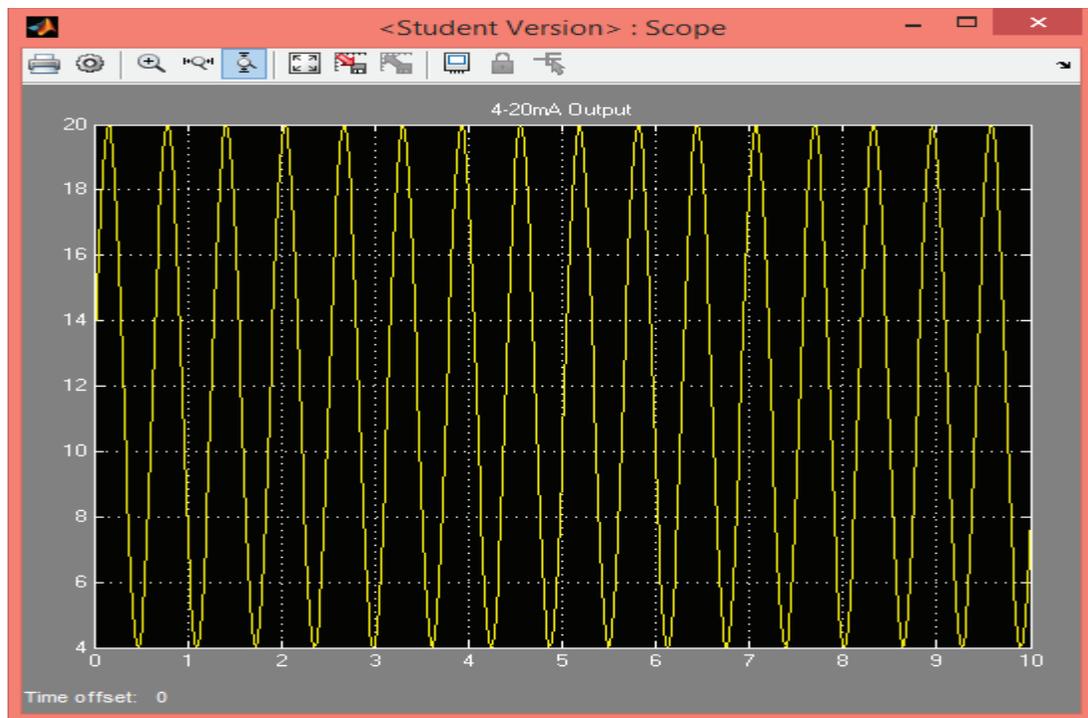


Figure 4.5 4-20mA transmitter output

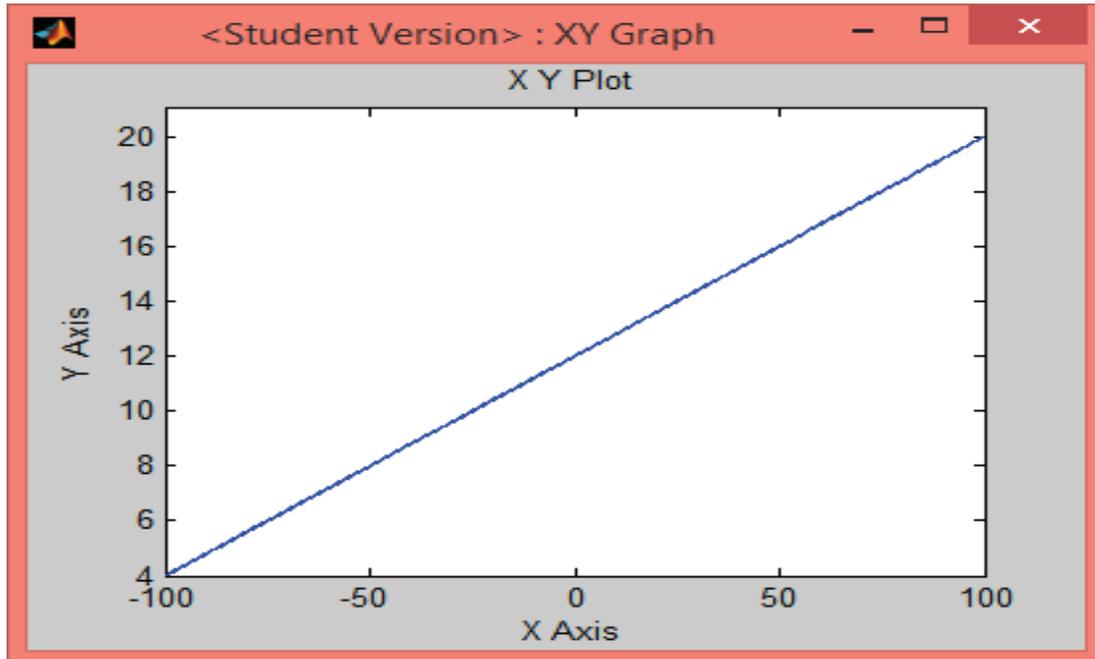


Figure 4.6 Output current (Y) vs. Input signal (X)

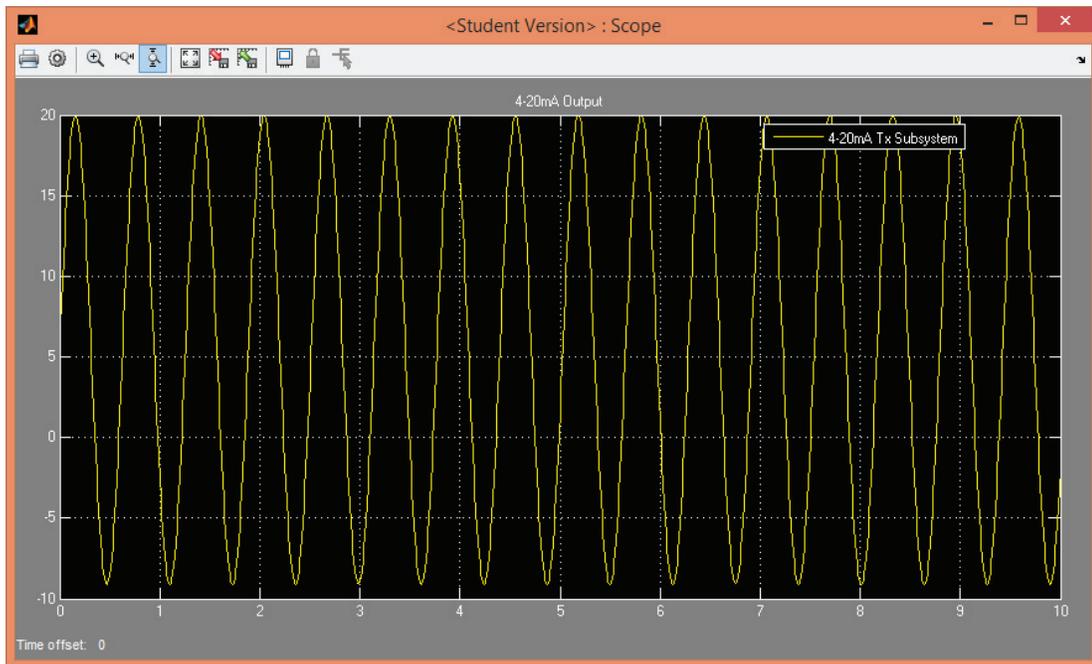


Figure 4.7 Erroneous output current

4.3.4 DAS 4-20mA Receiver Model

The analogue signal produced by the 4-20mA transmitter must be converted into a voltage for further processing by the DAS. This is achieved by using a load resistor and applying Ohm's law, which is shown in Figure 4.8.

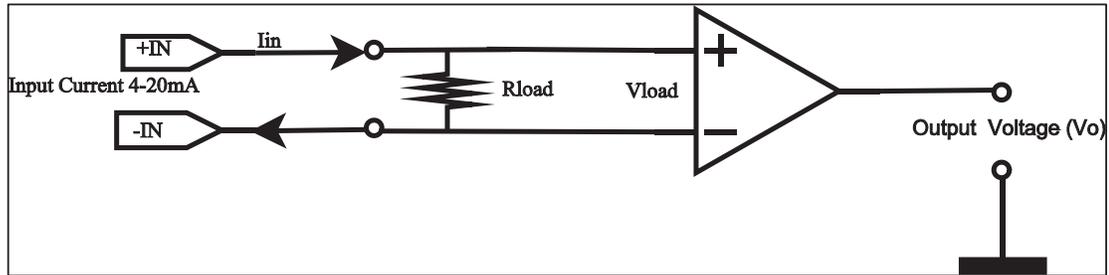


Figure 4.8 Simplified 4-20mA receiver diagram

The input current I_{in} passing through R_{load} generates voltage V_{load} , which is fed to a differential amplifier to produce an output voltage V_o . The output voltage is given by:

$$V_o = A(V_+ - V_-) \quad (4.6)$$

where A is the op-amp open loop gain, V_+ signal to non-inverting input and V_- signal to inverting input. The output voltage is given as:

$$V_{Load} = I_{in} * R_{Load} = V_+ - V_- \quad (4.7)$$

Thus, if V_+ is always greater than V_- , then output signal V_o will always have a positive value and given by:

$$V_o = A(I_{in} * R_{Load}) \quad (4.8)$$

It is a known fact from literature that gain A is infinity for an ideal op-amp and very large for a practical op-amp. This infinity gain will result in an infinity output signal (i.e. saturation), thus operating in the nonlinear region of the op-amp. Furthermore, the input impedance into the inverting and non-inverting inputs is infinity for an ideal op-amp. This means there will be no current flow into the op-amp input terminals. Moreover, the input offset voltage of an ideal

op-amp is zero. However, this is not the case with practical op-amps (Ball, 2002).

4.3.5 Simulink 4-20mA Receiver Model

In developing a Simulink® model using equation (4.8), it is assumed that gain A is unity and that the output voltage, which is function of the input current I_{in} and R_{Load} , is a constant. To comply with the ANSI/ISA-50.00.01-1975 (R2012) standard – the receiver requirements in converting the standard input current signal (4-20mA) to a standard input voltage signal (V_o) – the load resistor R_{Load} is required to be equal to $250\Omega \pm 0.25\Omega$. These will result in a standard input voltage range of 1-5V.

The Simulink model of equation (4.8) is shown in Figure 4.9. It simply consists of two product blocks and constants. The model is verified by injecting an output signal generated by the 4-20mA transmitter model (Figure 4.10). It should be noted that the output of the 4-20mA transmitter model is not scaled to milliamps (mA). Thus, it is necessary to insert a gain block between the 4-20mA transmitter model and the 4-20mA receiver model. The gain is set at 0.001 as shown in Figure 4.10. The signal will thus be converted to milliamps (4mA to 20mA).

It can be seen from Figure 4.11 that the 4-20mA output current is converted to a voltage signal. The sinusoidal input signal is converted to a 4-20mA current signal, and this current signal is converted to a 1-5V voltage signal.

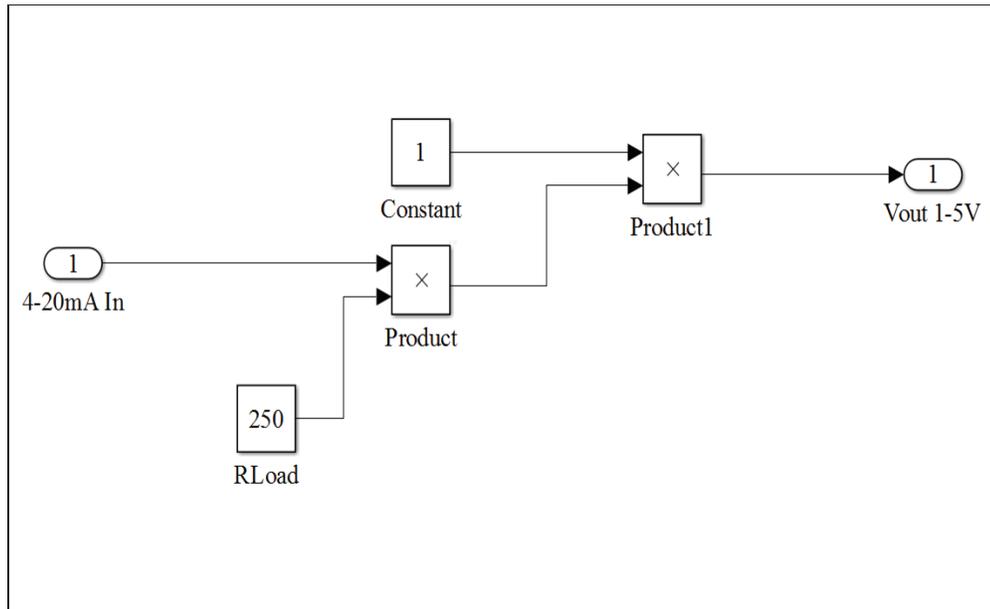


Figure 4.9 Simulink model for 4-20mA receiver

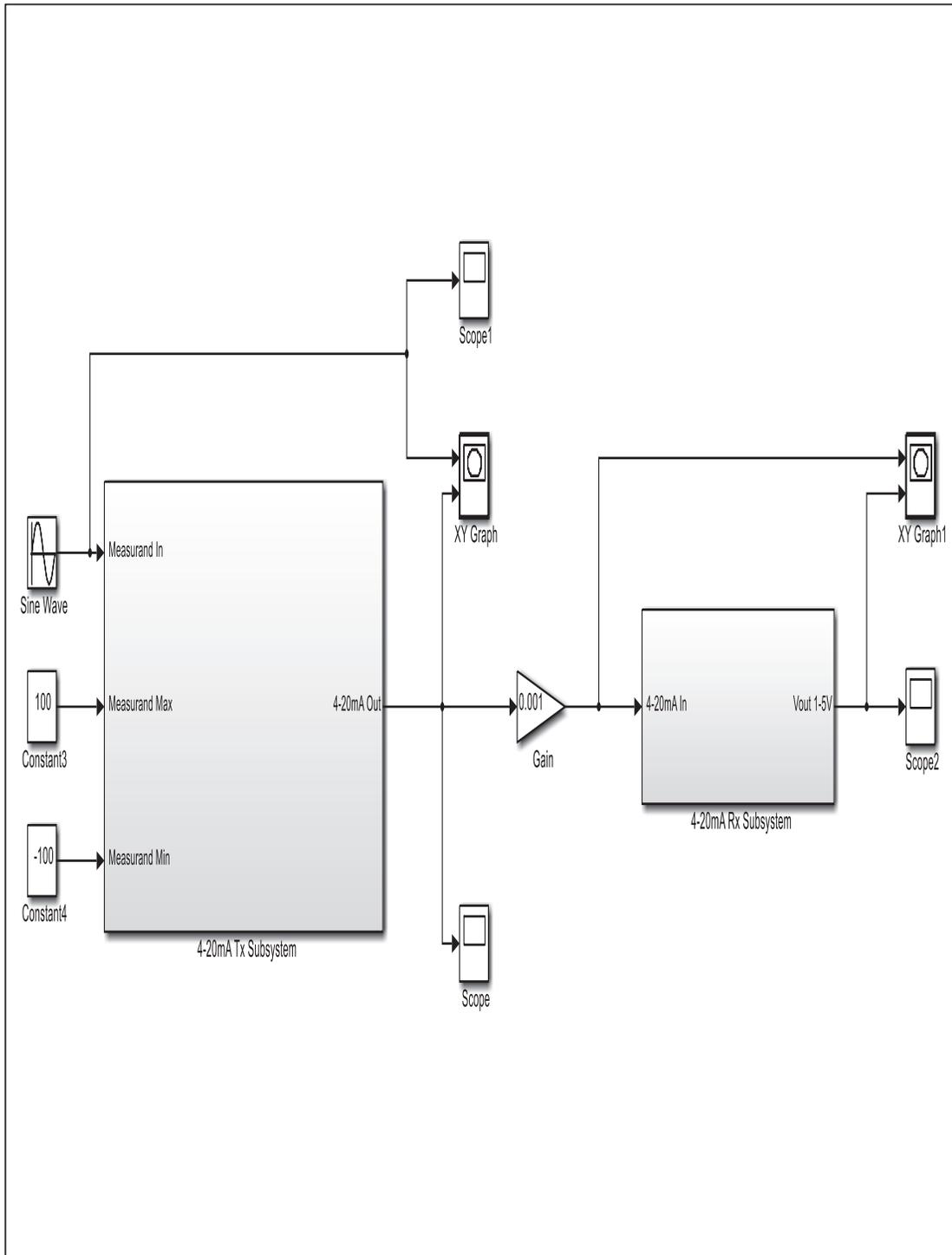


Figure 4.10 Simulation set-up for 4-20mA receiver model

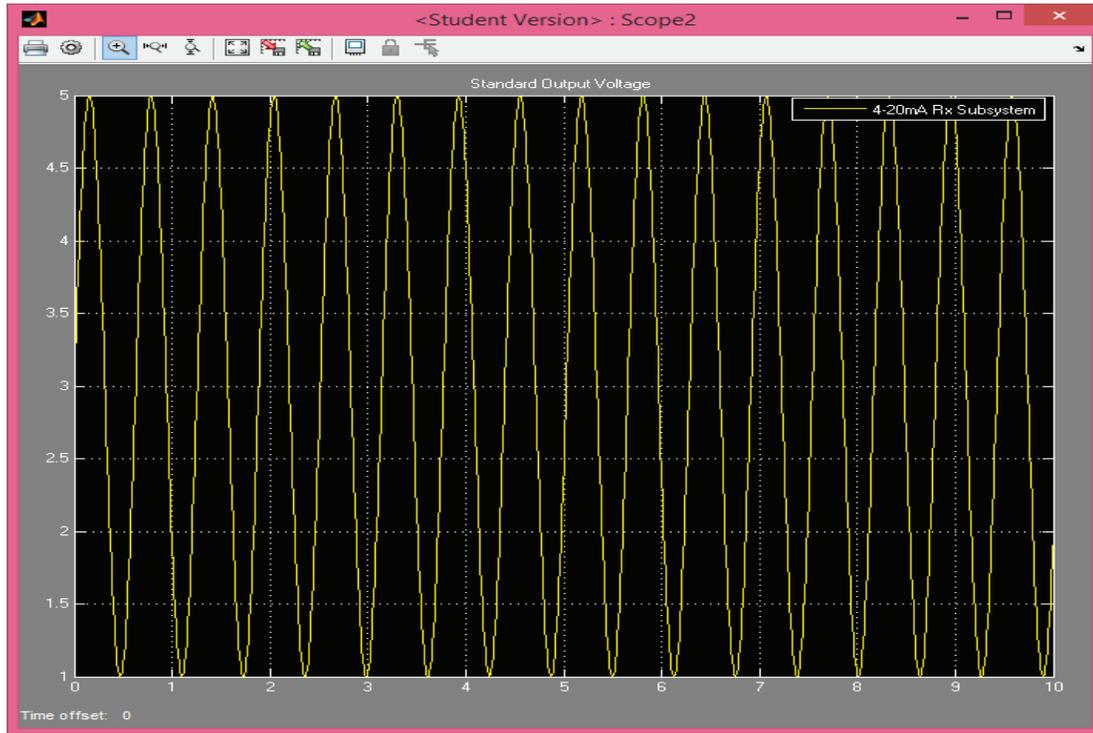


Figure 4.11 4-20mA receiver output signal

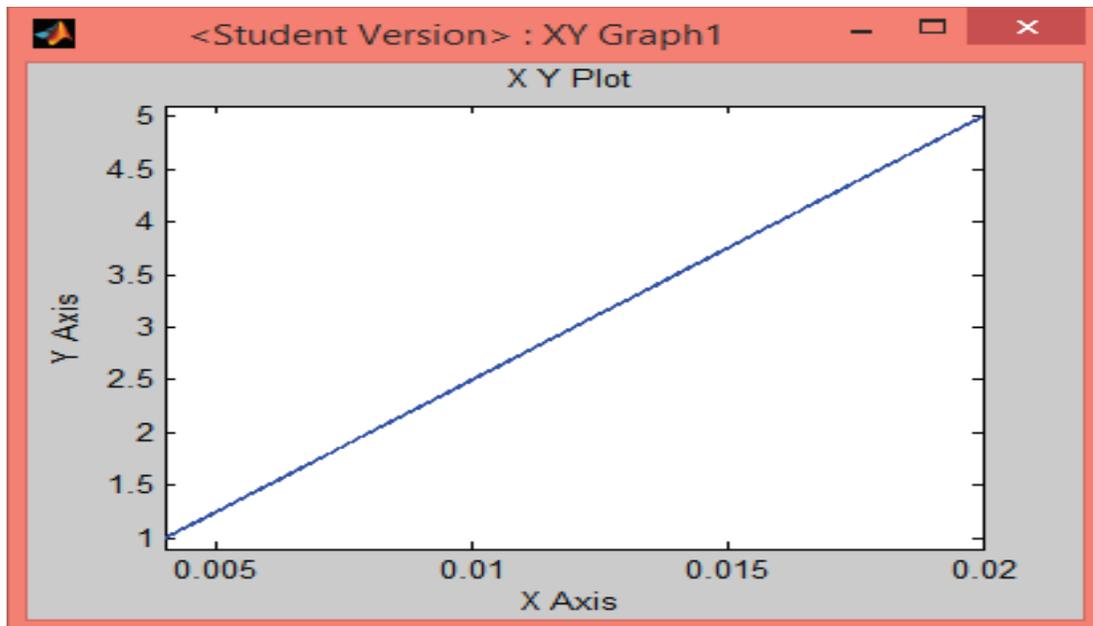


Figure 4.12 Input current (X Axis) vs. Output voltage (Y Axis)

As highlighted in Figure 4.11 and Figure 4.12, the 4-20mA transmitter output has a linear relationship with the 4-20mA receiver output. Basically, the input stimulus energy is captured by the sensor and transformed into a 4-20mA current signal, then transmitted and converted to an output voltage signal by the receiver. It is also evident that the sensor inputs span ($x_2 - x_1$) is always limited to 16mA span for the 4-20mA transmitter and 4V span for the 4-20mA receiver, i.e. if a standard input voltage is required. In essence, the sensor input span is compressed into 4V signal span by the standard 4-20mA receiver.

4.3.6 Analogue Filter

The DAS developed in this project must pass the signal from DC to 50kHz in some channels. The filter is required to limit the frequency bandwidth of the channels. This can be achieved by using a simple lowpass filter.

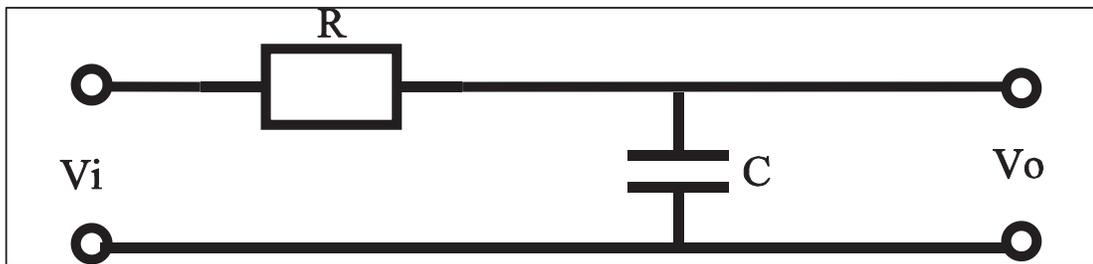


Figure 4.13 Single-pole lowpass filter

Analogue filters are simply implemented with a resistor and capacitor (RC) circuit (Figure 4.13). The transfer function of the filter in the frequency domain is given by:

$$\frac{V_o}{V_i} = \frac{1}{1 + sRC} \quad (4.9)$$

and the output signal V_o is thus represented by:

$$V_o = \frac{1}{1 + sRC} V_i \quad (4.10)$$

The Butterworth filter, the cut-off frequency f_c in time domain, is given by:

$$f_c = \frac{1}{2\pi RC} \quad (4.11)$$

where R is the resistor and C is the capacitor. The cut-off frequency is the maximum signal frequency allowed to pass through the channel. However, in the frequency domain, the pole of transfer function is $s = -1/RC$ and represents a signal decay of -20dB/decade (Carter and Mancini, 2009; Jung, 2005; Thompson, 2006).

The Simulink model representing equation (4.10) is simply implemented using a transfer function block, and the denominator coefficient is set by RC ; this is illustrated in Figure 4.14. The product RC (pole) is obtained from equation (4.11), which sets the cut-off frequency.

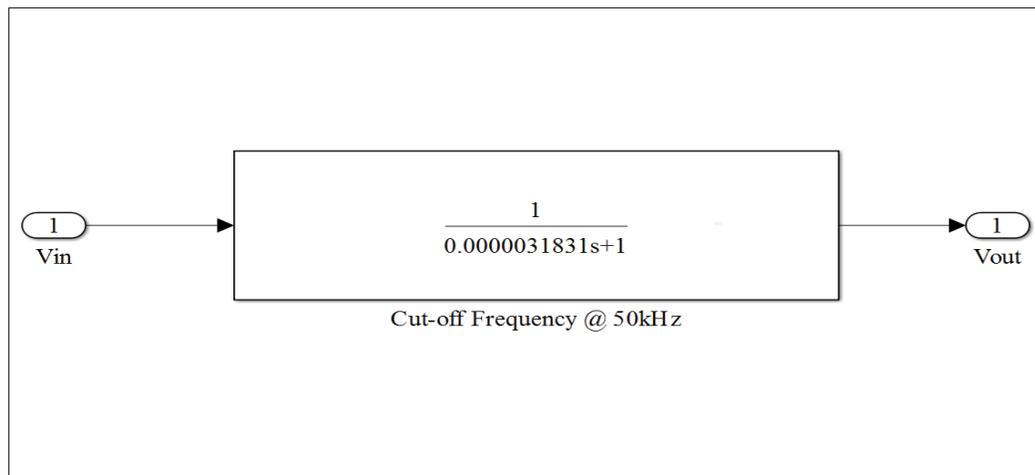


Figure 4.14 Lowpass filter transfer function

It can be seen from the simulation that at cut-off frequency, the output signal level is approximately 3.535V and out of phase (-45°) with the input signal. This is known as the -3dB point in literature; it is shown in Figure 4.16. However, this signal is still very high (i.e. $0.707 \cdot 5 = 3.535V$) and can be easily digitised by the ADC stage, thus processed as a useful signal. In other words, the channel bandwidth will be much wider than required. This will not achieve the anti-aliasing function of the filter. The simulation set-up is shown in Figure 4.15.

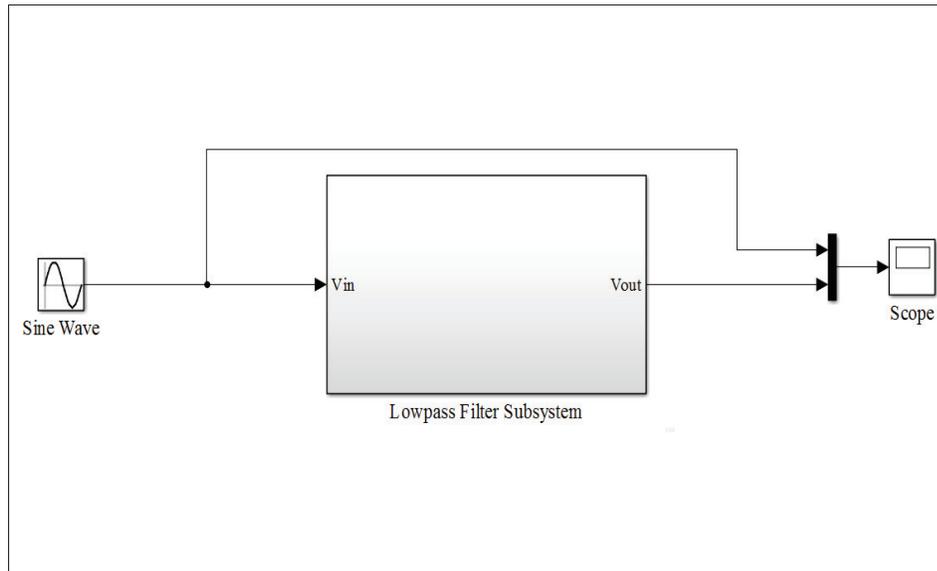


Figure 4.15 Single-pole lowpass filter simulation

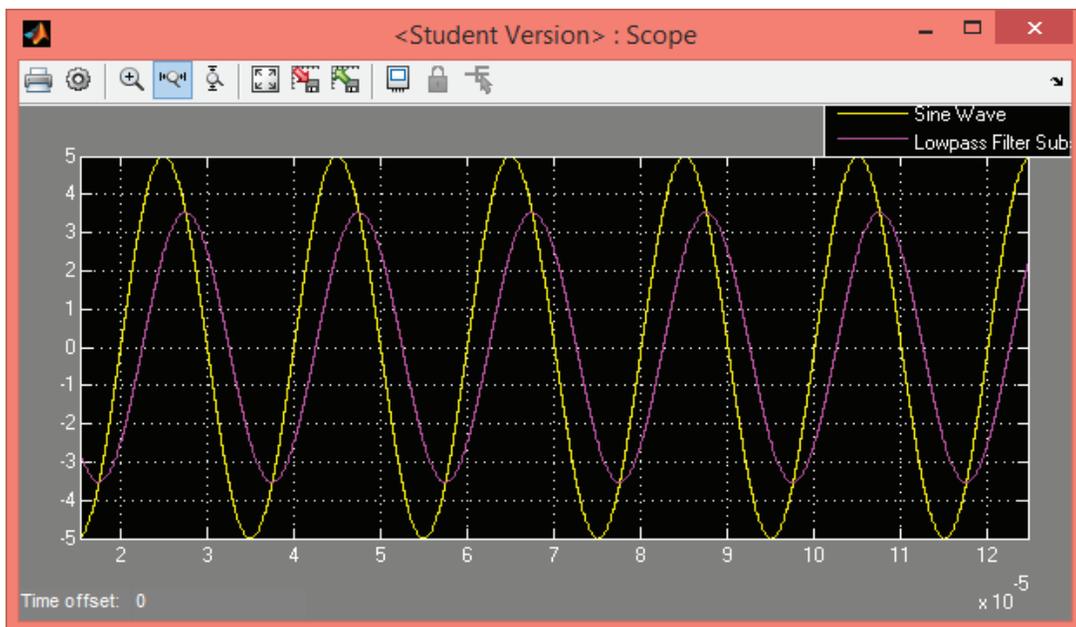


Figure 4.16 Single-pole filter output at 50kHz (cut-off frequency)

It is evident from simulation that a single-pole lowpass filter will not provide the required rejection of a signal above 50kHz (Figure 4.17), where an input signal at 400kHz is passed through the filter and the output signal level is still very sizeable. If a 10-bit ADC is used to

digitise the signal and assuming a reference voltage of 5V, then the signal at the output of the filter should be less than $5V/2^{10}$ ($< 4.882\text{mV}$).

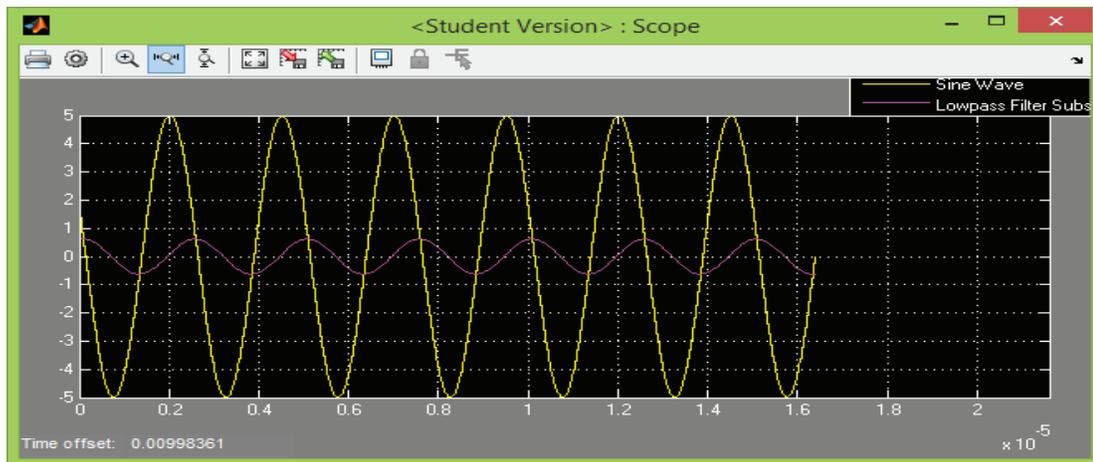


Figure 4.17 Single-pole filter output at 400kHz

To achieve the DAS bandwidth requirements, a high-order (multi-pole) filter will be required. This is usually achieved by cascading the single-pole filters, illustrated in Figure 4.18. The simulation indicates an improved and better signal rejection above 50kHz, when a fourth-order (4 poles) lowpass filter is used (Figure 4.19 and Figure 4.20). This will give a signal decay rate of 80dB/decade.

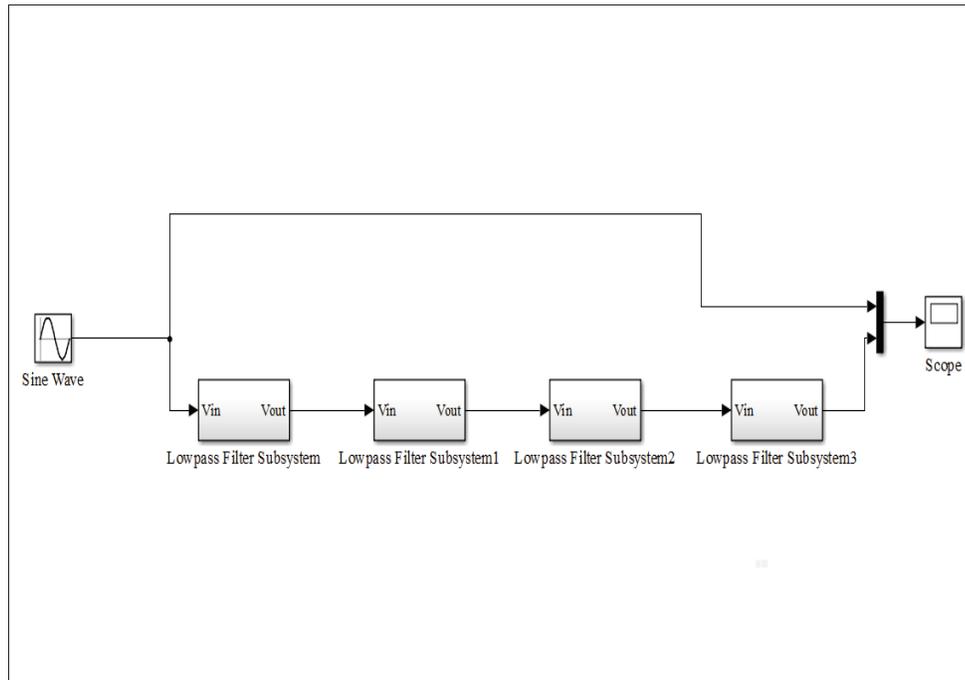


Figure 4.18 Fourth-order lowpass filter simulation



Figure 4.19 Fourth-pole filter output at 50kHz (cut-off frequency)

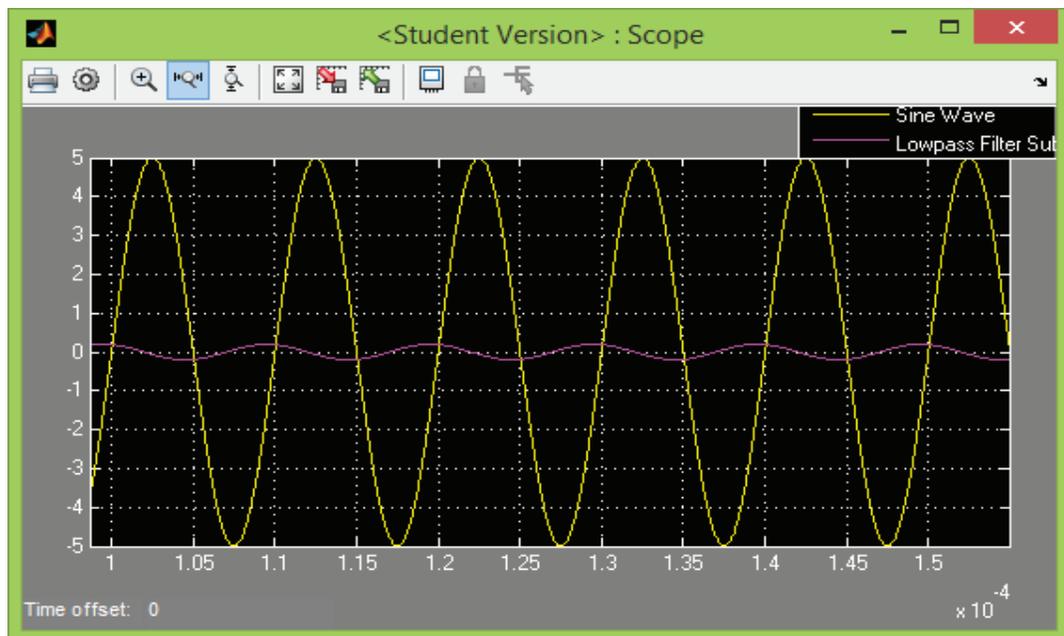


Figure 4.20 Fourth-pole filter output at 100kHz

4.4 Analogue-to-Digital Converter (ADC)

The major function of the ADC stage is to sample the analogue signal and then digitise the signal. The digitised signal will then be in the digital domain, thus allowing for further processing by the controller stage. The ADC is basically made of three core elements: sampler and hold, quantiser, and digitiser or binary encoder. The sampler and hold unit's task is to sample the incoming analogue signal and hold it for a specific time in order to allow the quantisation of the signal. Thereafter, once quantised, the signal is digitised by the binary encoder.

The goal is not to develop a new model for an ADC from scratch (which is beyond the scope of this project) but to use an existing idealised ADC Quantiser and holder model in MATLAB® Simulink® to model the proposed solution. This will provide a means to study the effects and limits of ADC resolution (number of bits) and reference voltage influences to the digitised signal.

4.4.1 Sampler and Hold Modelling

An ideal sampler is governed by equation (2.16), which requires samples to be taken at uniform periodic intervals T_s . The output of the sampler is still an analogue signal $X(t)$, but it consists of the instant of the sampled signal at that specific sampling time X_s . This signal must be held at that instant of time by the holder stage. The holder basically converts sampled continuous signal to sampled discrete signal X_h . The sampler and holder stage is usually modelled with a switch and ideal capacitor (Figure 4.21).

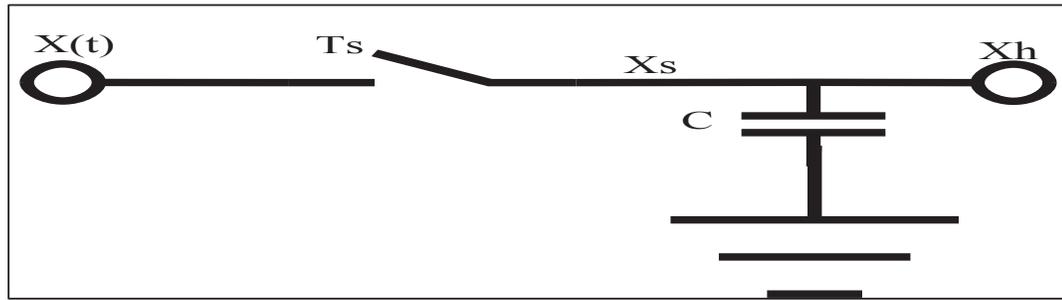


Figure 4.21 Ideal sampler and hold model

This model is represented by a Zero-Order hold block in Simulink, shown in Figure 4.22. The zero-order means that essentially no delay or filtering is performed in Figure 4.21. However, for the model depicted in Figure 4.21 to function as a filter, it should be the same as what is presented in Figure 4.13. Thus, from equation (4.11) where $R = 0$ for Figure 4.21, the cut-off frequency f_c is infinity. Therefore, an ideal sampler and hold has no delay or filtering capability, thus having an infinity bandwidth (zero-order hold).

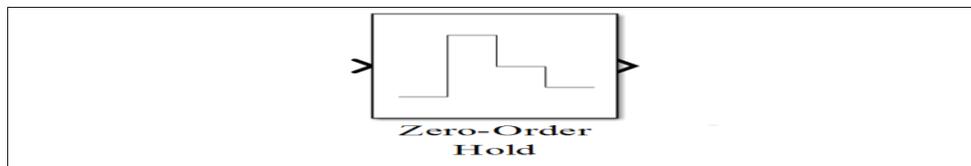


Figure 4.22 Simulink zero-order hold model

The simulation model is shown in Figure 4.23. The Simulink Zero-Order model only required the sampling rate as the input parameter. The signal frequency is set at 1Hz; as such, sampling frequency must be greater than $2.2 \times 1\text{Hz}$. The simulation results of a sampled analogue sinusoidal signal sampled at 10Hz and 100Hz sampling rates are shown in Figure 4.24. It is evident from the simulations that the Zero-Order hold output signal is smoother at higher sampling frequencies, thus having less sampling errors.

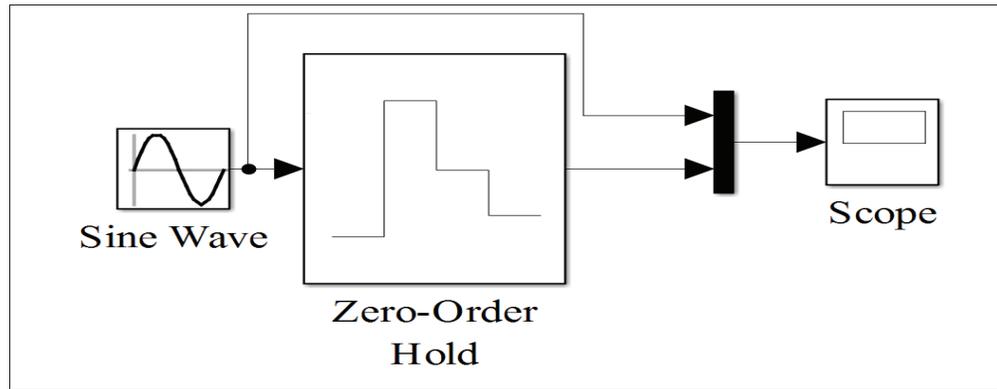


Figure 4.23 Zero-order hold simulation set-up

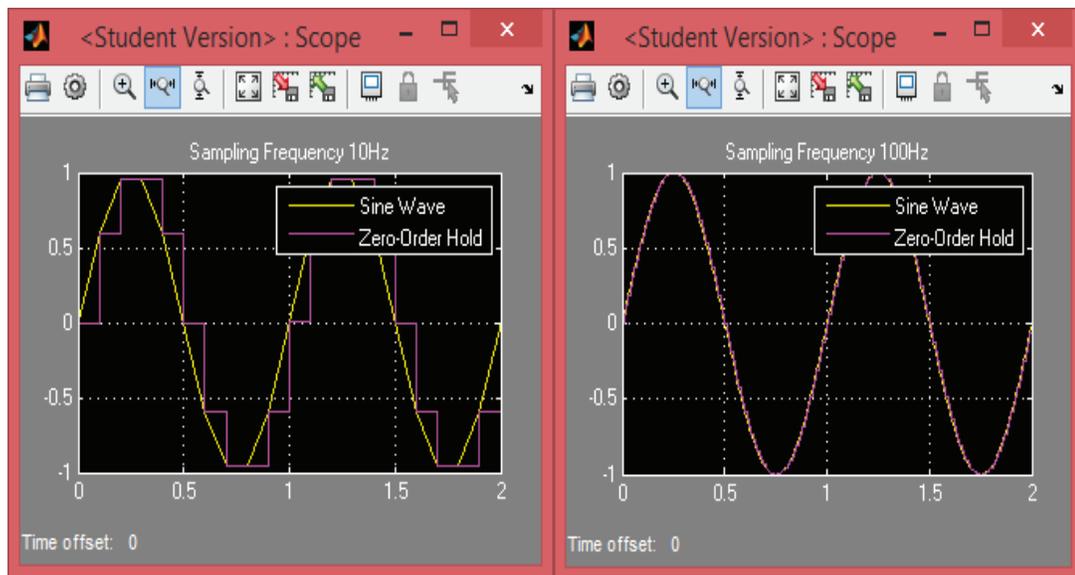


Figure 4.24 Zero-order hold output signal at 10Hz and 100Hz sampling frequency

4.4.2 Simulink Idealiser ADC Quantiser

The next stage to follow the Sampler and Hold unit is the quantiser, which quantises the sampled signal into multiples of a unit step size (Al-Eryani et al., 2011). This function is performed by the Simulink Idealised ADC Quantiser. The Simulink Idealised ADC Quantiser basically maps input points (which continuously hovers around input points) into one point at the output (MathWorks, 2014a).

The quantiser output is expressed as:

$$y = q * \text{round}\left(\frac{u}{q}\right) \quad (4.12)$$

where y is the output, q is the quantisation interval parameter, and u is the input signal. The simulation set-up is shown in Figure 4.25.

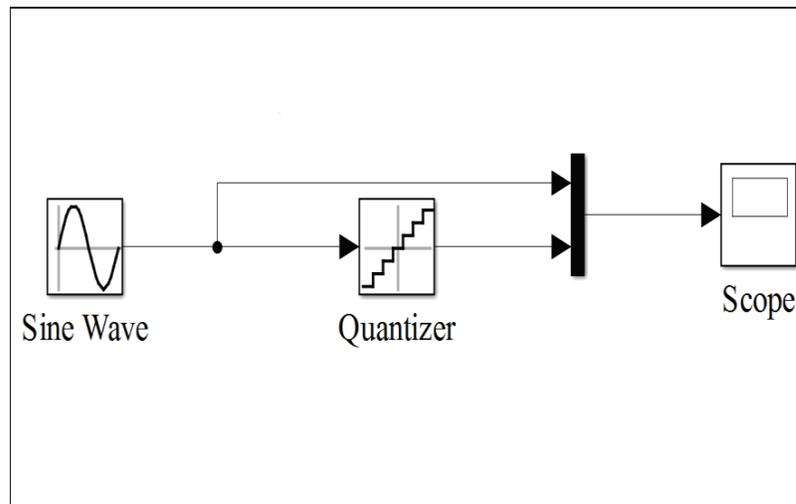


Figure 4.25 Idealised ADC quantiser simulation set-up

The simulation is performed using quantisation intervals set at 0.1, 0.01 and 0.001. It is evident that the shorter the quantisation interval, the shorter the stair steps, and the closest quantiser output signal represents the input signal. It is clear from equation (4.12) that the shorter the quantisation interval, the more data points are generated for a given input value.

The results of the simulation are highlighted in Figure 4.26 and Figure 4.27.

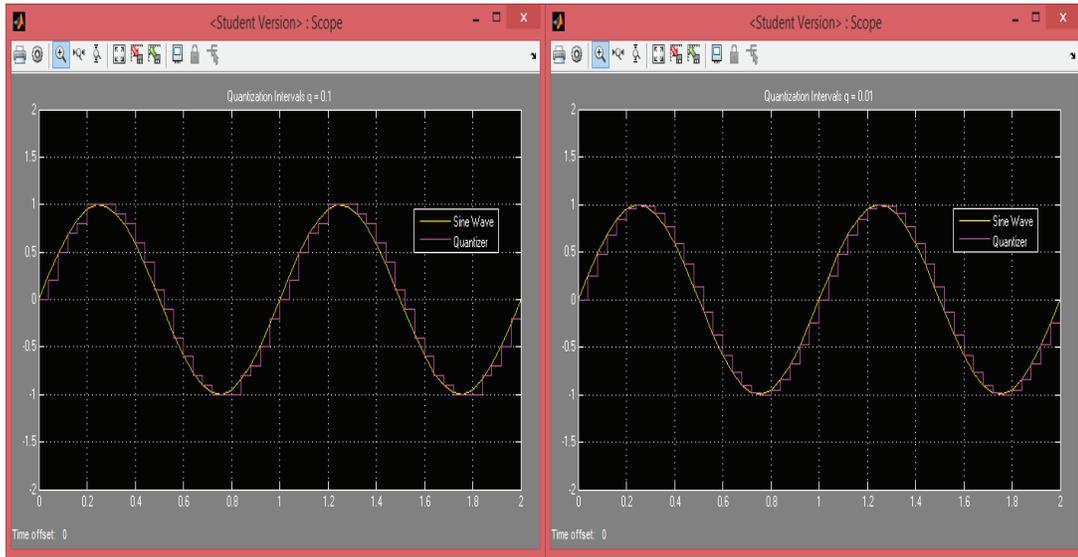


Figure 4.26 Simulation results for quantisation interval 0.1 and 0.01

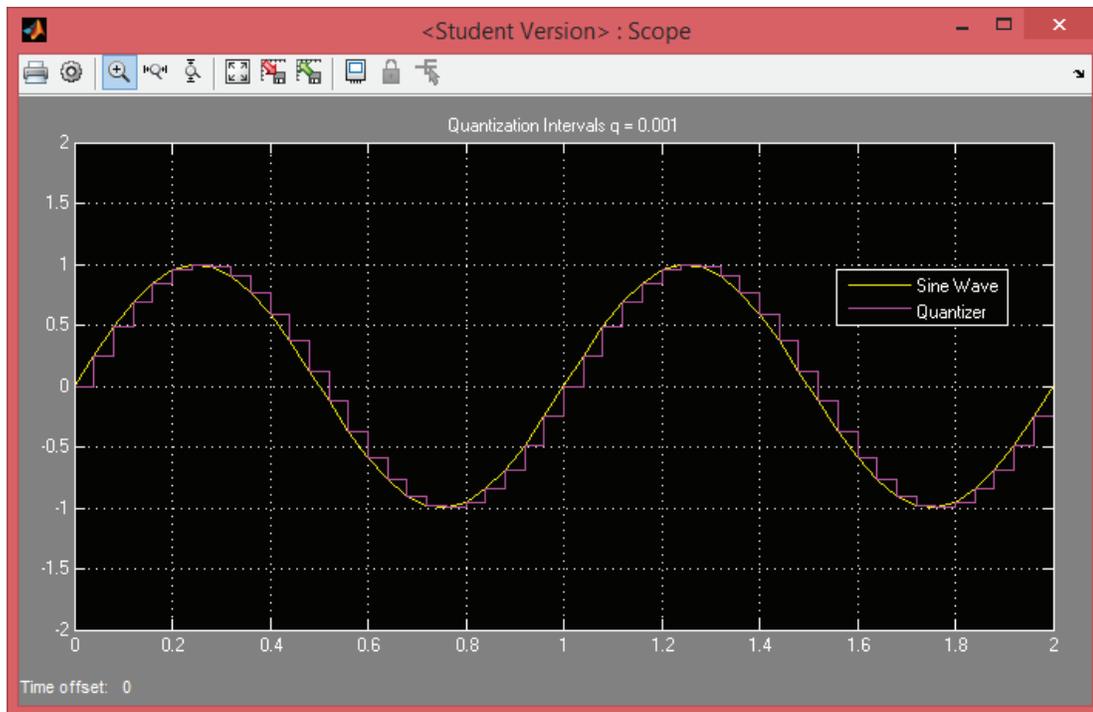


Figure 4.27 Simulation results for quantisation interval 0.001

The simulation set-up with Zero-Order Hold and Idealised ADC Quantiser in series is

reflected in Figure 4.28.

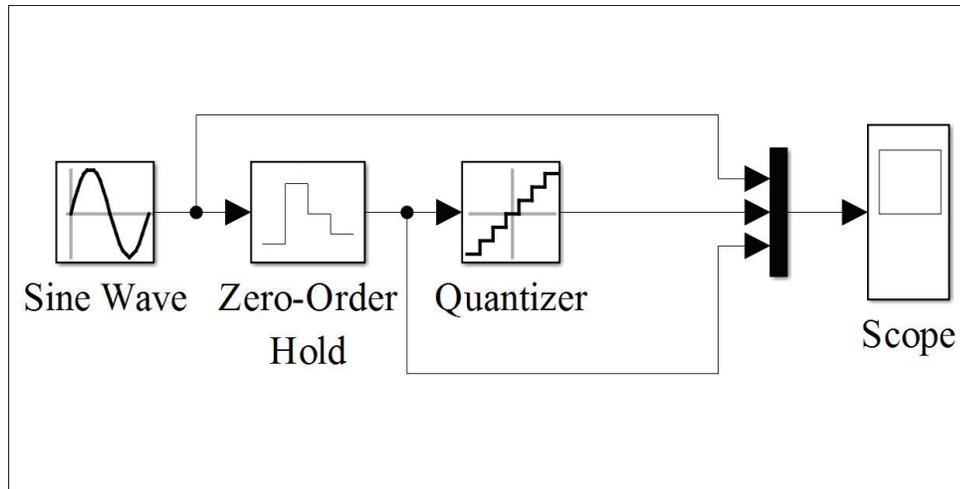


Figure 4.28 Zero-order hold and idealised ADC quantiser in series simulation set-up

4.4.3 ADC System-level Model

The model in Figure 4.28 does not truly model a practical ADC and has several limitations. The output of the quantiser stage is still not in digital form, just discrete stair step values at a specific time interval (quantisation interval) representing sampled signal value. Also, the Simulink Idealised ADC Quantiser output expressed by equation (4.12) does not depend on the full-scale range V_{fs} (input operating range) and number of binary bits N as expressed by equation (2.22) for an ideal uniform quantiser. It is also not clear from equation (4.12) what the minimum step size is. Thus, with this model, the quantisation errors cannot be determined. This will make it difficult to model a quantisation process that is similar to a quantisation process in practical ADC, consequently, limiting the ability to model and seeing the effects of the choice of ADC bits size (N) and input operating range (V_{fs}).

The Uniform Encoder block from MATLAB® DSP System Toolbox is used to model an ADC solution similar to a practical ADC. The Uniform Encoder block quantise input values using the same precision set by equation (2.22) and encode the mapped value to $2^N - 1$ integer value, which is the binary digital code. The output of quantiser stage can be expressed

as:

$$q_n = n \times Q_{LSB} \quad (4.13)$$

where q_n is the output with constant size, n is the integer value (range 0 to $2^N - 1$), and Q_{LSB} is the minimum quantisation step or resolution of the model. This model uses the rounding method to round down to the nearest quantisation step (MathWorks, 2014b). The quantisation error due to this method is given by equation (2.21).

The improved ADC model that closely represents a practical ADC is shown in Figure 4.29. This model consists of a Zero-Order Hold in the input stage of the Uniform Encoder. The Uniform Encoder will assign a binary value to the output signal of the Zero-Order Hold block based on the number of bits. The results of the simulation indicate a difference in resolution based on the number of bits of the Uniform Encoder. Also, the quantisation noise is noticeable from the results.

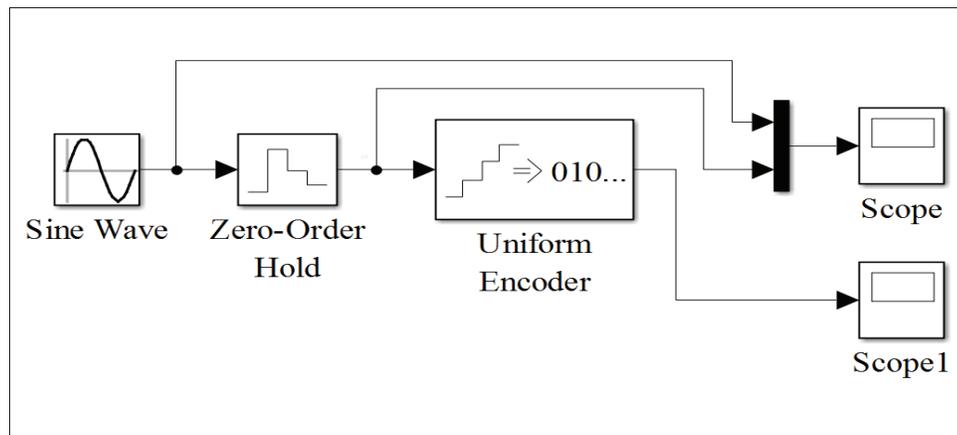


Figure 4.29 System-level ADC model simulation set-up

The quantisation error of 8, 10 and 12 bits ADC is approximately 4.883mV, 1.221mV and 305.176 μ V respectively. This is based on equation (2.21) and represents the maximum quantisation error of an ADC. This is basically the difference between the quantised output (equation (4.13)) and the input signal.

It is evident that the higher the number of bits, the better the ADC resolution, thus less quantisation error. The theoretical RMS noise is 2.819mV, 704.944 μ V and 176.163 μ V for 8, 10 and 12 bits ADC, given by equation (2.23). From equation (2.24), the signal-to-noise ratio for 8, 10 and 12 bits ADC is 49.92dB, 61.96dB and 74dB respectively.

The model is limited, as it excludes other errors discussed in literature. It is only focused on higher-level abstraction behaviour and studying the effects of a number of bits, input voltage range and voltage references on the digitised signal.

4.5 Data Acquisition Controller

The output from the ADC stage must be collected, stored, communicated and displayed to be of any use to a DAS user. This core function is performed by the controller stage. The data acquisition controller is made of hardware and software. The software modules' operations are presented in this section, and they are referred to as the DAS controller.

The controller can be subdivided into three subsystems consisting of the following modules:

- Data Collector
- Data Processing
- Data Displaying

The data collector module is modelled by the controller on the DAS side as shown in Figure 4.1. The data processing and displaying modules are modelled by the controller on the PC/laptop side (Host system).

4.5.1 Data Collector

The data collector module's main tasks are to control, acquire and format data from the ADC. Therefore, the data collector can be further subdivided into three modules:

- Controlling Module
- Data Formatting Module

- Data Transmitting and Receiving Module

The data collector must be able to acquire all the analogue signals from the 16 channels. The diagram and module interaction is shown in Figure 4.30.

Controlling Module

This sub-module's task is to control data from the ADC. This is done by sending a control signal to initiate, start or stop the ADC conversion process and querying if data is ready. The data obtained from the ADC is in a numerical form (i.e. integer) and transmitted to the control module at an ADC sampling rate. The precision of the numerical data depends on the number of bits of the ADC. The module must be able to process and execute commands obtained from the host system via the data transmitting and receiving module. The data received from the host system is expected to be commands and parameters to set up and control the data collector. The model is presented in Figure 4.30.

Data Formatting Module

This sub-module's main task is to arrange data, pack and prepare data for transmission. The module must be able to take any code width (resolution) data from the ADC. This module might also perform data buffering before data is sent to the next module. The number of bits transmitted from the ADC is equal to the resolution of the ADC.

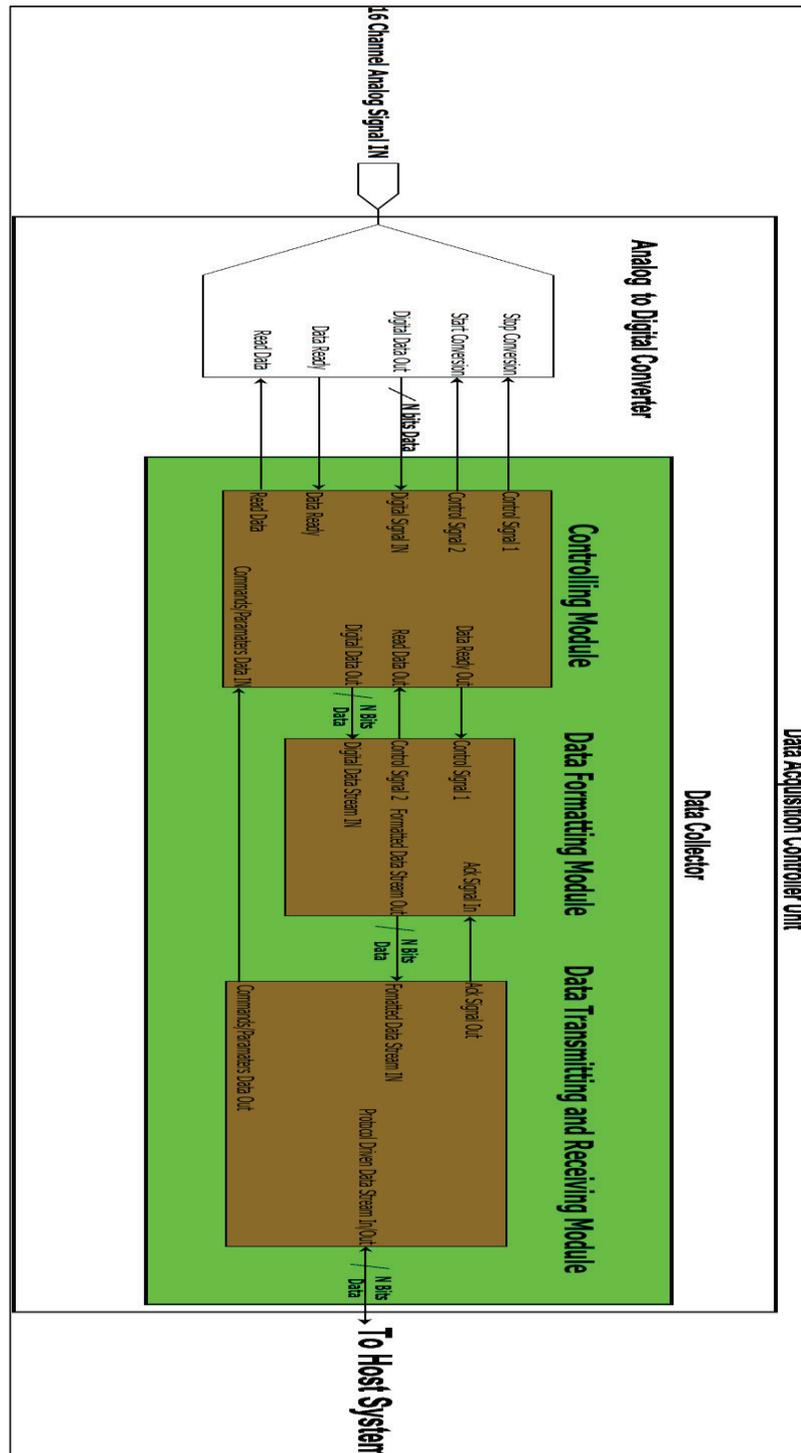


Figure 4.30 Data acquisition controller model

Data Transmitting and Receiving Module

The core function of this module is to transmit, receive and verify data transmitted to and from the host system. The output of the data formatting module is sent to this module for transmission and acknowledgement. The module will use an appropriate communication interface protocol to send and receive data from the host system. This is shown in Figure 4.30.

4.5.2 Data Processing Controller

This module is responsible for receiving data, buffering and acknowledging data on the host system. It should receive data correctly and buffer the data for further processing (Figure 4.31). The controller can be subdivided into two modules:

- Data Transmitting and Receiving Module
- Data Buffering Module

Data Transmitting and Receiving Module

Data receiving sub-module should ensure error-free data reception and acknowledgement of data received. The module will use an appropriate communication interface protocol to receive data from the data transmitting and receiving module.

Data Buffering Module

The output from the data receiver module feeds data into this data buffering module. The data buffer's main task is to collect and organise data into memory before further processing (Heath, 2002).

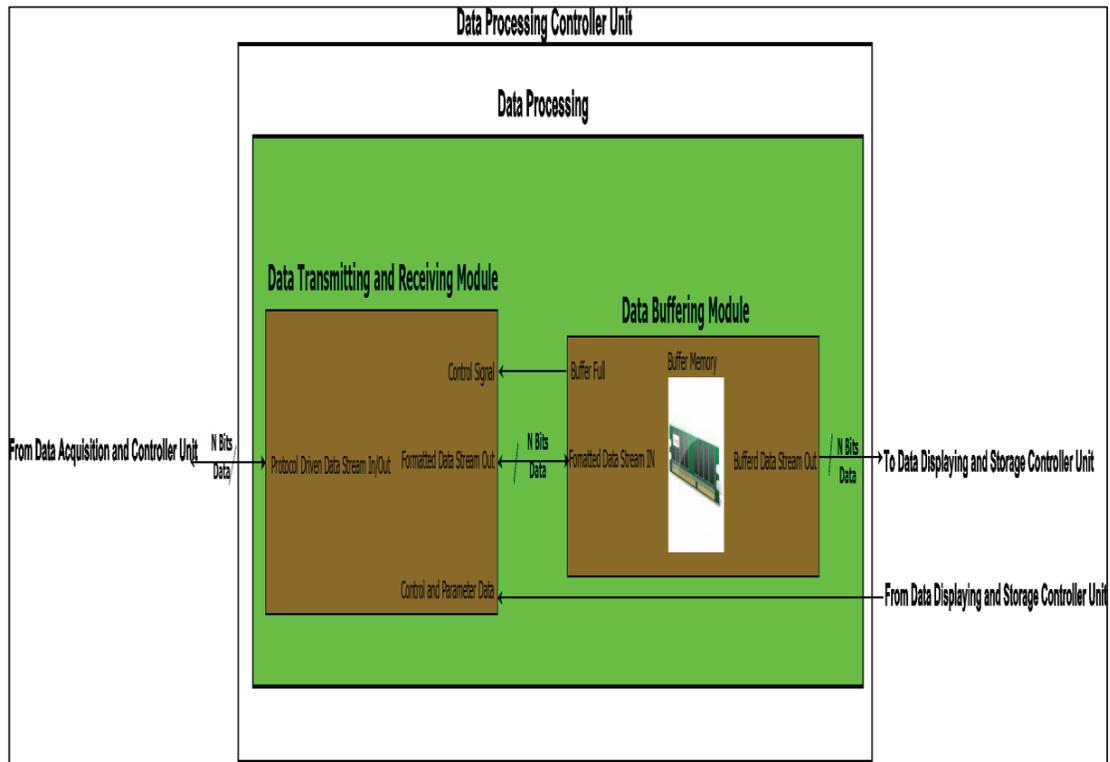


Figure 4.31 Data processing controller

4.5.3 Data Displaying and Storage Controller

The core function of this controller is to display acquired data in a meaningful way. This will usually involve the use of displays for human-computer interaction. The controller will reside on the host system side. Furthermore, this controller can also be divided into three sub-modules:

- Data Converter
- Displaying Module
- Data Storage

The controller is presented in Figure 4.32.

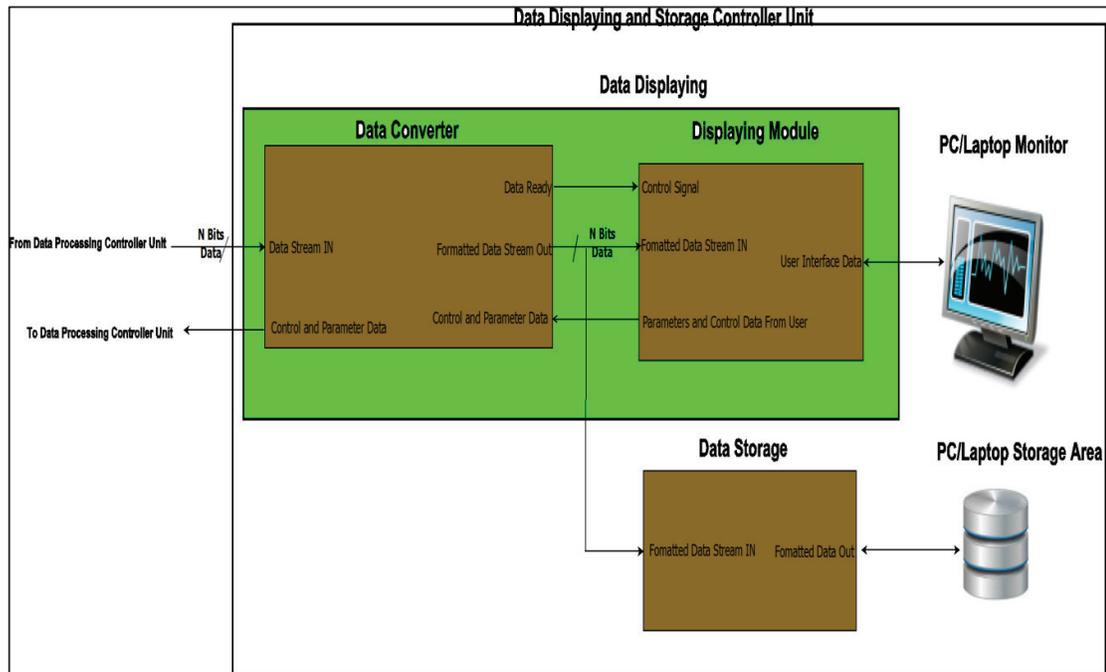


Figure 4.32 Data displaying and storage controller

Data Converting

The data converting module function is to convert the acquired digital data into a meaningful signal. However, this requires the sub-module to allocate the correct units to the streamed data received from the DAS controller (e.g. °C, metres, Nm, Newton). The channel allocation information will be provided by the Displaying module. The model is depicted in Figure 4.32.

Displaying Module

The module will provide the required information interface to the user. This will be done through a laptop or PC display. The module will be required to obtain parameters from the user. The parameters required shall be a number of channels sampled (maximum 16) and measurement units allocated to a specific channel. Furthermore, the module must graphically plot the data from each channel. The module will also provide the user the ability to start or stop the data acquisition process (Figure 4.32).

Data Storage

The module's function is to store data to a disk. This data may be stored in binary or ASCII format. The binary format has the advantage of saving disk space but requires pre-processing (i.e. export to ASCII or text or spreadsheet format) to be meaningful to the user. However, the ASCII file occupies a larger disk space compared to a binary file containing the same information (Austerlitz, 2003). The advantage of ASCII format data is that it is already in human readable form and can easily be imported to a spreadsheet application for further analysis and displaying. Furthermore, for easy import to spreadsheet programs (i.e. Microsoft Excel) the data can be stored in special ASCII file format called comma separated variables (CSV). This essential is an ASCII text file, which has data parameters separated by a special character comma (,).

Chapter 5 DAS Hardware, Firmware and Software

5.1 Introduction

The previous chapter discussed the system description and model of this study. The DAS system design is the main focus of this chapter. The signal conditioning stage is critical in the signal chain path. The design and simulations are presented, and circuit simulation results are discussed. The ADC is one of the important subsystems within the whole DAS system; it must be “specified” correctly to achieve the accuracy required for the system. The limits of the ADC subsystem are determined and discussed. The DAS controller and Host hardware, firmware and software requirements are detailed.

5.2 Signal Conditioning Card Design

The analogue signal conditioning stage is required to “condition” the analogue signal. Signal conditioning must improve the performance of DAS and not degrade the signal. It usually involves filtering, scaling or amplification, and impedance matching, and should not introduce any noise to the signal. In this case, the signal conditioning stage will perform the following main functions:

- Filtering (lowpass filter)
- Impedance matching
- Scaling

In practice, noise removal is hard to achieve, but there are a number of techniques one must adhere to in order to minimise noise effects. Noise cannot be removed from electronic circuits, due to the fundamental, inherent nature of current flow. However, it can be minimised and ignored if it is small enough and not does not influence measurements.

5.2.1 Input Stage

The input stage must be able to convert the analogue current signal (4-20mA) to voltage and band limit the signal. Furthermore, it must also process voltage signal from 0 to 10V and also band limit the signal. The input stage is basically a standard 4-20mA receiver and must meet the ANSI/ISA-50.00.01-1975 (R2012) requirements. The standard allows input resistance between 0 and 600 ohms. The zero input resistance is of no practical use, as it will always give a zero voltage drop, thus not useful. The ANSI/ISA-50.00.01-1975 (R2012) requirements also set a standard voltage output signal for the receiver to a range of 1-5V, which corresponds to an input resistance of 250 ohms over the 4-20mA range. Moreover, it limits the source resistance of the receiver output stage to a maximum of 250 ohms. The full standard receiver requirements as per ANSI/ISA-50.00.01-1975 (R2012) standard are listed in Table 3.2.

The aforementioned requirements are mostly applicable to the 4-20mA receiver. Additionally, the 0-10V signals require much high input resistance. This is due to fact that voltage sources usually have low output impedance, resulting in a loaded output signal if input impedance is too low. Another requirement of the input stage is to limit the input signal bandwidth to 5kHz and 50kHz. This means that a lowpass filter will be required with a cut-off frequency of 5kHz and 50kHz respectively.

5.2.2 Buffer Amplifier

To meet the requirements of the 4-20mA receiver, the voltage developed across the input resistance (250Ω) must not deviate from $\pm 4\text{mV}$ at the amplifier inputs. This means that current drawn by input impedance seen in the input stage (Figure 5.1) must be less than $\pm 16\mu\text{A}$ ($= \frac{\pm 4\text{mV}}{250\Omega}$). To meet this requirement, the input stage must be followed by a buffer that provides the required high impedance. Because of very high-input impedance of the op-amp, the current in the positive terminal should be very small, thus negligible (Figure 5.2) and will be assumed to be zero. This will result in the output signal being the function of input signal given by:

$$V_{o1} = V_{in} \quad (5.1)$$

$$= I_{4-20mA} * 250\Omega$$

where I_{4-20mA} is the 4-20mA input signal. Therefore, the output signal (V_{o1}) will be dependent on the input current signal. The circuit shown in Figure 5.2 performs impedance transformation, by providing high-input impedance to the input voltage V_{in} and very low source impedance to the next stage. The buffer only provides unit gain to the signal. However, practical op-amps have finite current flow into their input terminals, which is known as input bias current (Carter and Mancini, 2009; Clayton and Winder, 2003; Huijsing, 2011; Jung, 2005; Terrell, 1996; Thompson, 2006). This bias current must flow out of the non-inverting input terminal pin for proper operation of the buffer. If this bias current is more than $\pm 16\mu A$, it will alter the voltage drop developed across the input resistor (250Ω) by more than $\pm 4mV$, thus violating the ANSI/ISA-50.00.01-1975 (R2012) requirement. The op-amp chosen in the implementation stages must have a very low input bias current, preferably in the range of nA to pA (10^{-9} to 10^{-12}).

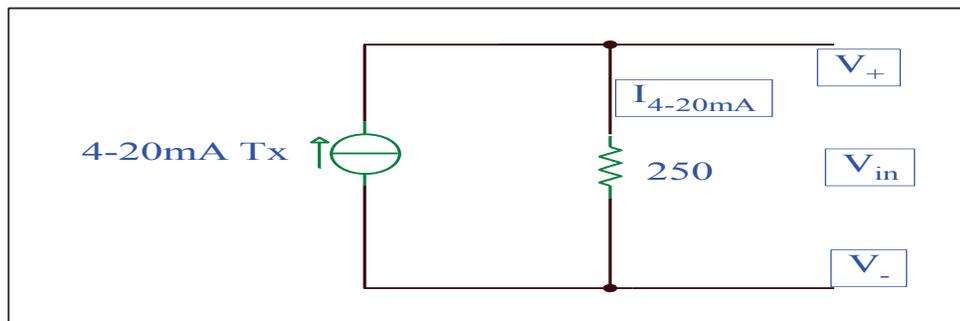


Figure 5.1 4-20mA signal receiver

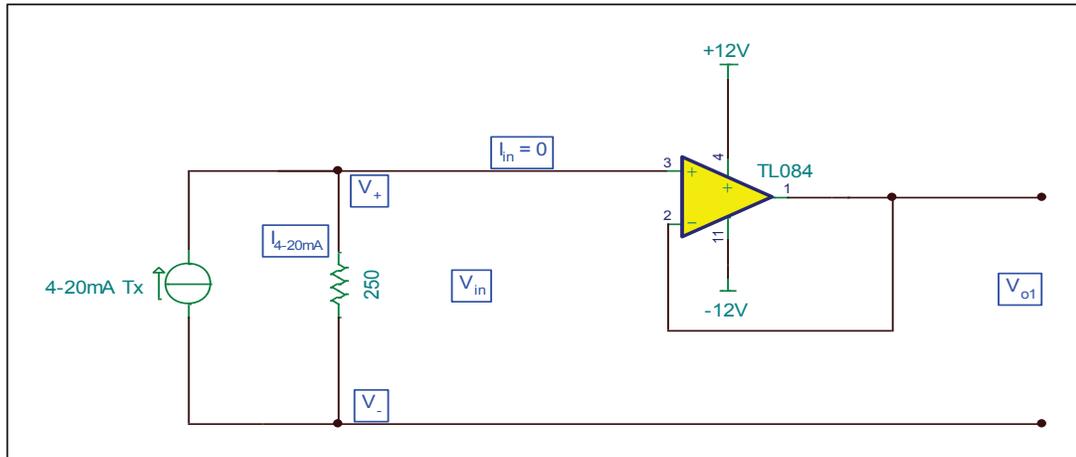


Figure 5.2 Input buffer stage

5.2.3 Differential Amplifier

The next stage following the buffer or voltage follower is the differential amplifier stage. Its main task is to amplify the difference in voltage signals between the non-inverting terminal and inverting terminal. The major advantage of the differential amplifier is that it provides good common mode signal rejection (Zumbahlen, 2008). This means equal input signals seen at the amplifier's inputs will be rejected, producing no output signal. It is important that the differential amplifier have fairly high-input impedance to reduce the signal loading, thus minimising errors. The output impedance of the buffer stage shown in Figure 5.2 is given as (Terrell, 1996):

$$Z_{oB} = \frac{R_o}{A_{OL}} \quad (5.2)$$

where R_o is the open loop resistance given in the op-amp datasheets, usually shown on the op-amp schematic. A_{OL} is the open loop gain of the op-amp, and it is frequency dependent and obtained from an op-amp datasheet in graph form. A_{OL} can be estimated as the ratio of the unity gain frequency of the op-amp and the maximum input signal frequency (Fischer-Cripps, 2002), given by:

$$A_{OL} = \frac{F_{UG}}{F_{SB}} \quad (5.3)$$

The unity gain frequency or gain-bandwidth product (F_{UG}) is one of the op-amp's critical performance parameters, which gives the frequency at which the open loop gain falls to unity (0dB) and is obtained from the op-amp datasheet. This parameter is important as it determines the frequency's response to the op-amp. For equations (5.1), (5.2), (5.4) and (5.5) to hold, A_{OL} must be large in the design of the buffer, filter and differential amplifier circuits, as their development are based on ideal op-amps characteristics. Furthermore, it is obvious from equation (5.2) that for very small source impedance from the buffer output, A_{OL} must be very large. The buffer and differential amplifier circuits must be able to pass through and process signal bandwidth of 5kHz and 50kHz.

The output voltage of the differential amplifier stage shown in Figure 5.3 is given by:

$$V_{o2} = V_{o1} \frac{R_2}{R_1 + R_2} \left(\frac{R_3 + R_4}{R_3} \right) - V_- \frac{R_4}{R_3} \quad (5.4)$$

If $R_1 = R_3$ and $R_2 = R_4$, then the gain of the differential amplifier stage is set by the ratio of the resistors R_2 and R_1 . This may result in a reduced common mode rejection parameter, which is one of the disadvantages of this circuit configuration (Fischer-Cripps, 2002; Jung, 2005). This is because the common mode rejection parameter is given as the ratio of the op-amp open loop gain and the common mode gain. The common mode gain is influenced and dependent on the ratio of the resistors R_2 and R_1 , thus the differential amplifier gain.

In this design, voltage gain is not required, and a special case is used where $R_1 = R_2 = R_3 = R_4$. This will reduce equation (5.4) to:

$$V_{o2} = (V_{o1} - V_-) \quad (5.5)$$

Therefore, the output signal is equal only to the difference in buffer stage output signal (V_{o1}) and the signal return (V_-) and with unity amplification. This will aid in improving common

mode rejection parameter (Figure 5.3). The output of this stage will feed into the next stage. This requires that the output source impedance be very low as compared to the input impedance. The output impedance can be approximated (Fischer-Cripps, 2002) by:

$$Z_{oDS} = \frac{(R_1 + R_2)R_o}{A_{OL}R_1} \quad (5.6)$$

with $R_1 = R_2$, equation (5.6) will reduce to:

$$Z_{oDS} = \frac{2R_o}{A_{OL}} \quad (5.7)$$

where R_o is the open loop resistance and A_{OL} is the open loop gain.

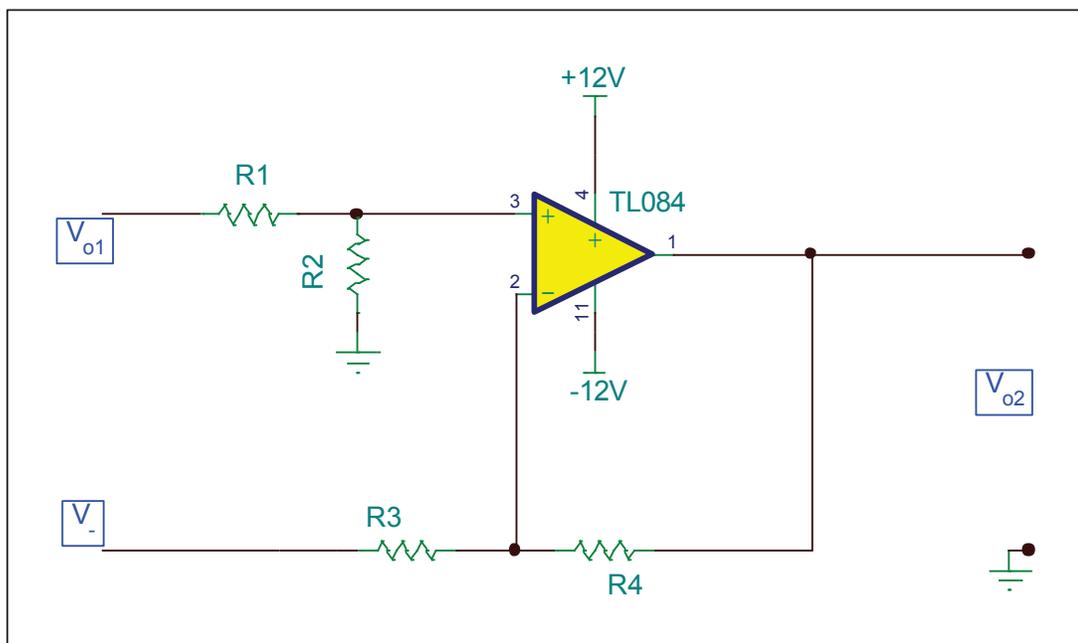


Figure 5.3 Differential amplifier stage

The inherent source of error for this circuit (Figure 5.3) is the op-amp input offset voltage (V_{i0}). This voltage will result in an output voltage ($V_{o2} = V_{i0}$) when input voltage is zero ($V_{o1} =$

$V_- = 0$). The op-amp chosen for the design must have very low input offset voltage, preferably below $\pm 4\text{mV}$. It should be noted that this stage could have been designed using an instrumentation amplifier, but with limited configuration or programmable options (Jung, 2005; Kester, 2005b).

5.2.4 Signal Scaling

This stage consists of a voltage divider network and buffer amplifier. The main task of this stage is to provide an output signal scaled to a particular level. This means the output signal is dependent on the ratio of the two input resistors R_1 and R_2 , which is called scale factor (Figure 5.4). The voltage V_{sc} is given by:

$$V_{sc} = \frac{R_2}{R_1 + R_2} V_{o2} \quad (5.8)$$

For equation (5.8) to hold, the input stage should draw negligible current from the voltage divider network. This will require R_2 to be at least 100 times (Glisson, 2011) smaller than the buffer input impedance. Because the output stage of the circuit consists of the buffer amplifier, the output signal is given as $V_{o3} = V_{sc}$, which is the same as given in equation (5.1). The input impedance of this stage is given by:

$$Z_{sc} = R_1 + R_2 \quad (5.9)$$

The reason for the above is because the signal from the differential stage (V_{o2}) sees only the path through R_1 and R_2 to ground, due to the buffer's high-input impedance. This means only the bias current will flow into the input terminal of the buffer. This requires a choice of an op-amp with very low bias currents. The choice of R_1 and R_2 is critical, as it should not load the output signal (V_{o2}) and R_2 should not be too high either. If R_2 is too high, the bias current might result in a large voltage drop developing across R_2 (i.e. $1\mu\text{A} \times 1\text{M}\Omega = 1\text{V}$), thus introducing errors in the signal. The signals will be scaled to be within the range of the ADC input signal levels and reference voltages (Appendix V ADC Resolution).

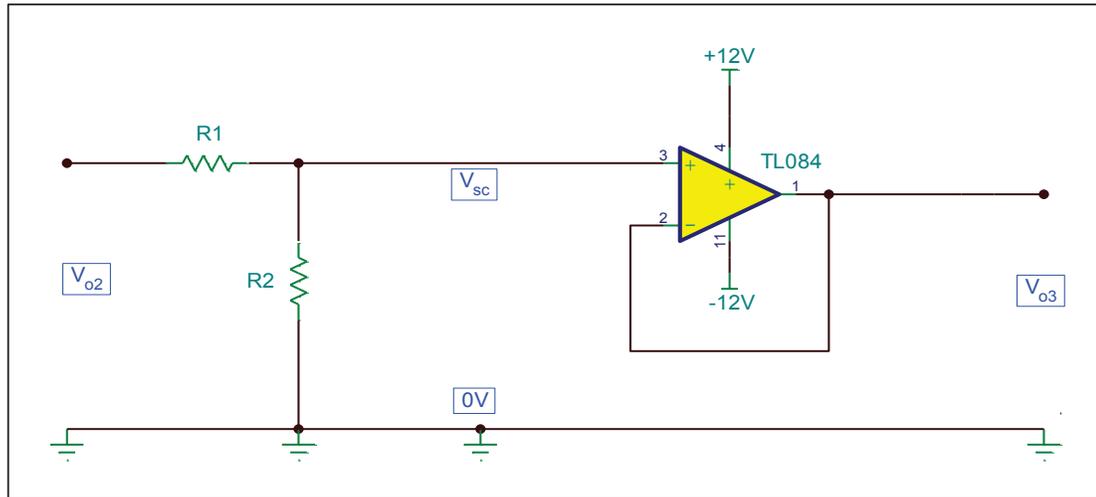


Figure 5.4 Signal-scaling stage

5.2.5 Filter Design

A filter is required to band limit signal passing through the channel. This means it will only allow signals below specific frequency (in this case, 5kHz and 50kHz) to pass through the channels. As mentioned in literature, these filters are actually anti-aliasing filters, which filter out images or alias signals generated during the sampling process. These image signals lie outside the Nyquist bandwidth, which is from DC to $\frac{F_s}{2}$ (half the sampling frequency) (Zumbahlen, 2008). The filter stop-band is expressed as (ibid):

$$F_{SB} = F_s - F_c \quad (5.10)$$

where F_s is the sampling frequency and F_c is the filter's cut-off frequency. The minimum sampling frequency is given by equation (2.17), where $f_{max} = F_c$; this makes F_c the maximum allowed signal frequency. This requires that the input signal's highest frequency component should be below Nyquist bandwidth.

The 5kHz filter consists of a resistor and a capacitor forming a voltage divider network similar to the one discussed above and the buffer amplifier (Figure 5.4). With reference to Figure 5.5, the output voltage of the filter is given by equation (4.10), where V_o is replaced by V_{Fi} and V_i

is replaced by V_{o3} , thus the filter signal is $V_{o4} = V_{o3}$. The output signal V_{o4} has a frequency dependent response, which is set by equation (4.11). This equation gives the cut-off frequency of the channel, which is dependent on the values of R_1 and C_1 . The product of R_1 and C_1 gives the frequency response of the circuit (Figure 5.5). In determining and choosing R_1 , it is important that its value should not be too low such that it will load and attenuate signals below the cut-off frequency ($F_c = 5\text{kHz}$). However, the combination of R_1 and C_1 should form a low impedance path and attenuate signals at a frequency above 5kHz. The circuit in Figure 5.5 will give attenuation levels of -20dB per decade from signal frequencies above cut-off frequency.

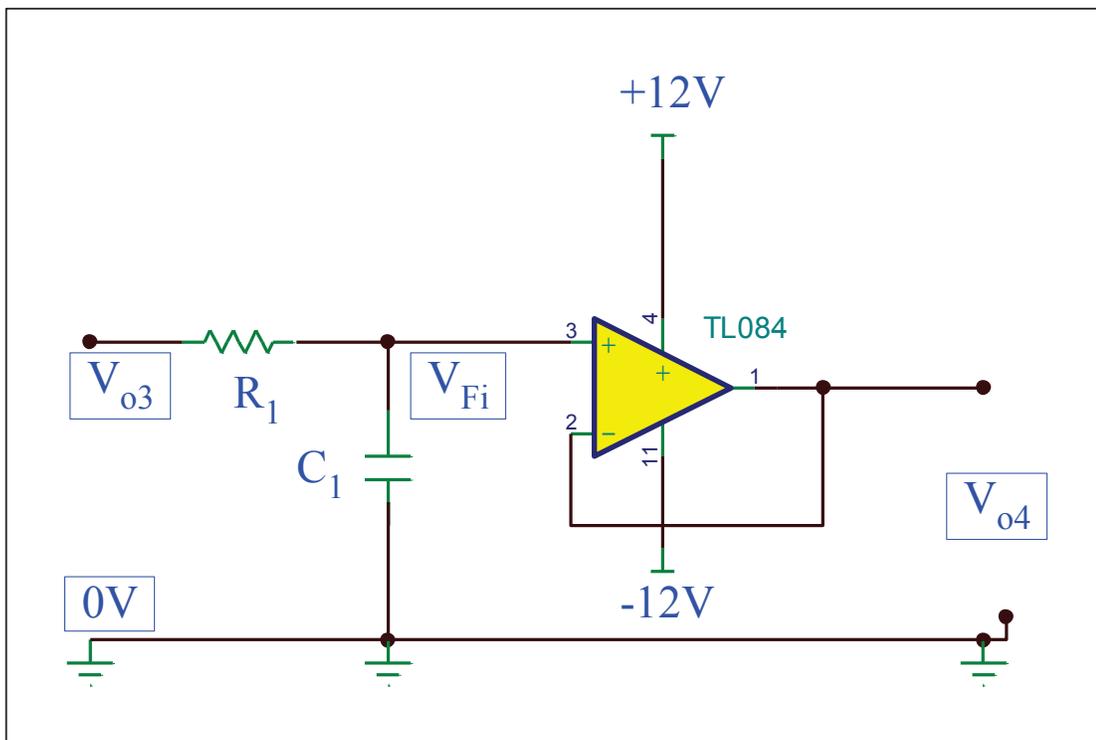


Figure 5.5 Active lowpass filter for 5kHz channels

A filter design tool (FilterPro) from Taxes Instruments® is used to determine the values of R_1 and C_1 . These values are based on the requirements in Table 5.1.

Table 5.1 Lowpass filter (5kHz channel) requirements

| Parameter | Value | Units | Notes | Comments |
|-----------------------|-------------|-------|---|--|
| F_s | 200 | kHz | <i>Sampling frequency</i> | High sampling rates help in improving resolution, so that single-order filter can be used. |
| F_c | 5 | kHz | <i>Filter cut-off frequency</i> | |
| Filter order | 1 | | | |
| Filter Gain | 1 | | | |
| Filter Response | Butterworth | | <i>Gives flat response with minimal signal distortion</i> | |
| Stop-band Attenuation | 20 | dB | <i>Maximum attenuation for a single-pole filter</i> | |

The sampling frequency is set at 40 times (40x) more than the minimum requirement for 5kHz channels. This will help in pushing the image frequencies far above the input signal bandwidth. The Nyquist bandwidth is equal to 100kHz; thus, the signal's highest frequency component is below Nyquist bandwidth ($5\text{kHz} < 100\text{kHz}$), and the stop-band bandwidth is 195kHz. The values obtained from the filter design tool for RC product is $R_1 = 3.2\text{ k}\Omega$ and $C_1 = 10\text{ nF}$. These values can be verified using equation (4.11), which gives a cut-off frequency of $F_c = 4973.59\text{ Hz}$.

The 50kHz filter consists of a resistor and capacitor circuit and with the RC circuit placed on the feedback path of the buffer. This configuration is known as the Sallen-Key filter topology (Glisson, 2011). Some of its main advantages is that it is less dependent on op-amp amplifier performance and has the smallest resistor and capacitor value spread (Jung, 2005). However, one major disadvantage of this topology is that the cut-off frequency (F_c) and the Q factor are highly sensitivity to the amplifier gain (ibid.). The second-order Sallen-Key filter circuit is shown in Figure 5.6. However, in this case, the topology's weakness is not going to be a problem, as no gain is required (gain should be unity). The cut-off frequency (F_c) is still fundamentally governed by equation (4.11). The cut-off frequency (F_c) for a two-order lowpass filter is given by (Floyd, 1999):

$$F_c = \frac{1}{2\pi\sqrt{R_1C_1R_2C_2}} \quad (5.11)$$

The filter is designed using a filter design tool (FilterPro) from Taxes Instruments® and is used to determine the values of the RC networks, based on the requirements highlighted inTable 5.2.

Table 5.2 Lowpass filter (50kHz channel) requirements

| Parameter | Value | Units | Notes | Comments |
|-----------------------|-------------|-------|---|--|
| F_s | 200 | kHz | <i>Sampling frequency</i> | Improves resolution |
| F_c | 50 | kHz | <i>Filter cut-off frequency</i> | |
| Filter order | 7 | | | High-order filter required to increase the sharpness at cut-off. |
| Filter Gain | 1 | | | |
| Filter Response | Butterworth | | <i>Gives flat response with minimal signal distortion</i> | |
| Stop-band Attenuation | 80 | dB | <i>Required attenuation</i> | |

The sampling frequency is set at two times (2x) more than the minimum requirement for 50kHz channels. The Nyquist bandwidth is equal to 100kHz, with the signal's highest frequency component below Nyquist bandwidth and the stop-band bandwidth is 150kHz. The higher-order filter is required to have a sharper cut-off because the signal's highest frequency component (50kHz) is closer to the Nyquist bandwidth (100kHz). As such, a seven-order lowpass filter consisting of a cascade of one-order filter (Figure 5.5) and three two-order filters (Figure 5.6) is required.

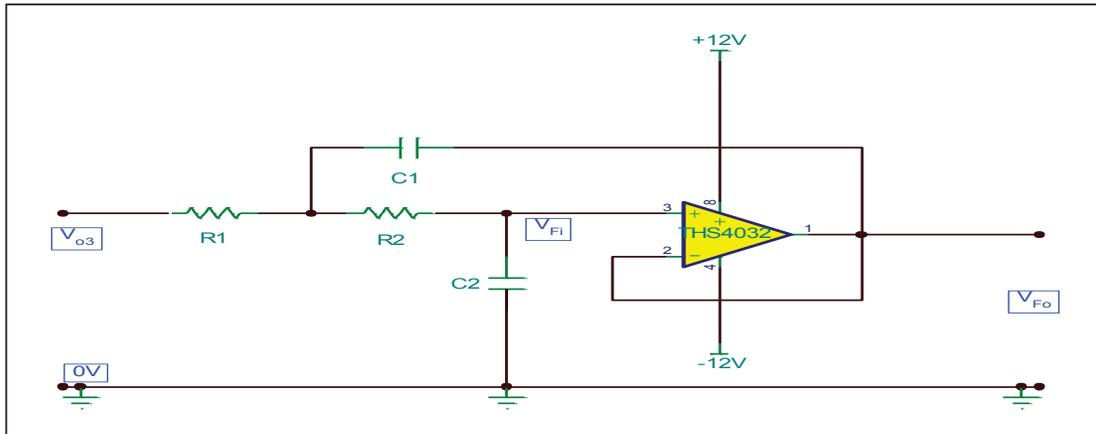


Figure 5.6 Active two-order lowpass filter 50kHz channels

The values obtained from the filter design tool are shown in Table 5.3, and the circuit is shown in Appendix M.

Table 5.3 Resistor and capacitor values for seven-order lowpass filter

| Item | Resistor (Ω) | Item | Capacitor (nF) |
|-------|-----------------------|-------|----------------|
| R_1 | 3.2 k | C_1 | 1 |
| R_2 | 2.64 k | C_2 | 1.24 |
| R_3 | 3.09 k | C_3 | 1 |
| R_4 | 1.74 k | C_4 | 2.61 |
| R_5 | 2.23 k | C_5 | 1 |
| R_6 | 619 | C_6 | 20.5 |
| R_7 | 796 | C_7 | 1 |

The first stage is made up of R_1 and C_1 , setting the cut-off frequency at $F_c = 49735.91 \text{ Hz}$ and providing the 20dB/decade attenuation. Further, the second, third and fourth stages are made of two-order filters providing the additional 120dB/decade attenuation. The choice op-amp is critical, as the stages should not load the RC networks of the subsequent stages at the desired channel frequency.

5.2.6 Component Selection

The passive components should be chosen based on the criteria highlighted in Table 5.4.

Table 5.4 Passive components tolerances

| Passive Components | Tolerance | Temperature Coefficient (TC) | Notes |
|--------------------|------------------------|---|---|
| <i>Resistors</i> | $\pm 0.5 \%$ or better | $25 \text{ ppm}/^\circ\text{C}$ or better | <i>Should be a thin film type (Soloman, 2010)</i> |
| <i>Capacitors</i> | $\pm 10 \%$ or better | $25 \text{ ppm}/^\circ\text{C}$ or better | <i>Must have lowest leakage current</i> |

The parameters list in Table 5.4 is the most critical in minimising error contributed by these components. It is important that the resistor values should not exceed $10 \text{ k}\Omega$, which will help in minimising thermal noise contribution generated by the resistor. The choice of op-amp is challenging, as there are literally thousands of op-amp IC that can be selected from for the design. The op-amp selected should have design parameters approximating a theoretical ideal op-amp and should use a dual-power supply. Both the AC and DC op-amp performance are important and must minimise their contributions to signal error. The parameters list in Table 5.5 (DC Parameters) and Table 5.6 (AC Parameters) are critical in choosing an op-amp for the design of the buffer, differential amplifier and anti-aliasing filter stages.

The op-amp DC parameters may affect the accuracy of the system, and they can contribute errors to the signal. Parameters such as input noise, offset voltages, bias currents and output

impedances should be as low as practically possible. Furthermore, parameters such as input impedances and gain should be as high as practically possible. On the other hand, the op-amp AC parameters may affect the filter performance. They should meet the minimum bandwidth requirement of the channels and filter.

Table 5.5 Op-amp DC parameters

| Parameter | Value | Units | Notes | Selection Criteria |
|------------------------------|-------|------------------|--|---|
| Maximum Input offset voltage | 5 | mV | <i>Inherent input voltage that causes output voltage when the voltage at the input + and – terminals are zero.</i> | Minimum input signal level must be larger than this parameter. |
| Maximum Input Bias current | 1 | μA | <i>Inherent input current that flows out of op-amps + and – terminals.</i> | This current must be extremely low such that it is negligible; it would not add significant error to the signal. |
| Minimum Input Impedance | 1 | $\text{M}\Omega$ | <i>Impedance between op-amps + and – terminals.</i> | Must be very large in order not to load the signal. Signal source impedance should at least be 1 000 times less than the input impedance. |
| Minimum Open Loop Gain | 1 000 | | <i>DC amplifier gain and frequency dependent</i> | Must be very large from DC to maximum signal frequency. |
| Maximum Output | 200 | Ω | <i>Output resistance in series</i> | Source resistance |

| Parameter | Value | Units | Notes | Selection Criteria |
|-----------|-------|-------|------------------------------------|--|
| Impedance | | | <i>with op-amp output terminal</i> | of the output signal. Must be very low compared to the input impedance of the next stage. It is reduced further by voltage feedback configuration. |

Table 5.6 Op-amp AC parameters

| Parameter | Value | Units | Notes | Selection Criteria |
|--------------------------------|-------|------------|--|--|
| Minimum Gain-Bandwidth Product | 3 | MHz | <i>The maximum frequency at which the op-amp open loop gain drops to unity</i> | Important in implementing filters; limits the performance of filters. |
| Minimum Slew Rate | 13 | V/ μ s | <i>Maximum change of voltage at the op-amp output terminals</i> | This parameter is important in implementing filters; it limits the maximum signal frequency used on an op-amp. |

Based on the foregoing selection criteria, the op-amp choice was narrowed down to two options:

- TL084BC
- THS4032

The list is not exhaustive, as there are other op-amps that can be used and which meet the criteria. The TL084BC consists of four op-amps in a 16-pin surface mount package, and the THS4032 has two op-amps in an 8-pin surface mount package. However, the THS4032 has lower input impedance compared to the TL084BC. The parameters lists of the chosen op-amps are shown in Table 5.7 and Table 5.8.

TL084BC AC and DC parameters:

Table 5.7 TL084BC design parameters

| Parameter | Value | Units | Notes |
|--------------------------------|-----------|-------|---------------------------|
| Maximum Input offset voltage | 5 | mV | <i>Meets the criteria</i> |
| Maximum Input Bias current | 7 | nA | <i>Meets the criteria</i> |
| Minimum Input Impedance | 1 000 | GΩ | <i>Meets the criteria</i> |
| Minimum Open Loop Gain | 2 000 000 | | <i>Meets the criteria</i> |
| Maximum Output Impedance | 128 | Ω | <i>Meets the criteria</i> |
| Minimum Gain-Bandwidth Product | 3 | MHz | <i>Meets the criteria</i> |
| Slew Rate typical | 13 | V/μs | <i>Meets the criteria</i> |

THS4032 AC and DC parameters:

Table 5.8 THS4032 design parameters

| Parameter | Value | Units | Notes |
|--------------------------------|--------|------------------------|---------------------------|
| Maximum Input offset voltage | 3 | mV | <i>Meets the criteria</i> |
| Maximum Input Bias current | 8 | μA | <i>Meets the criteria</i> |
| Minimum Input Impedance | 2 | $\text{M}\Omega$ | <i>Meets the criteria</i> |
| Minimum Open Loop Gain | 44 000 | | <i>Meets the criteria</i> |
| Maximum Output Impedance | 13 | Ω | <i>Meets the criteria</i> |
| Minimum Gain-Bandwidth Product | 100 | MHz | <i>Meets the criteria</i> |
| Minimum Slew Rate | 100 | $\text{V}/\mu\text{s}$ | <i>Meets the criteria</i> |

As mentioned, the DAS must have 16 input channels which must be able to measure 4-20mA and 0-10V signals. Over and above that, it must be able to handle 5kHz and 50kHz signal bandwidths. To achieve these requirements, it is proposed to divide the 16 channels into four categories:

- 7 channels for 4-20mA signals with 5kHz bandwidth (DC-5kHz)
- 7 channels for 0-10V signals with 5kHz bandwidth (DC-5kHz)
- 1 channel for 4-20mA signals with 50kHz bandwidth (DC-50kHz)
- 1 channel for 0-10V signals with 50kHz bandwidth (DC-50kHz)

Regarding op-amp choice for high bandwidth signals, it is recommended in applications that gain-bandwidth product (F_{UG}) parameter should at least be 100 times more than the highest signal frequency component (Pease, 2008; TI, 2011) given as:

$$GBP = 100 * G * F_C \quad (5.12)$$

where G is the amplifier close loop gain and F_C is the highest signal frequency, in this case, filter cut-off frequency. Therefore, the 5kHz bandwidth channels will require an op-amp with a minimum gain-bandwidth product (GBP) of 500kHz and the 50kHz bandwidth channels will require a a minimum GBP of 5MHz, provided the gain is unity ($G = 1$). The 5kHz bandwidth channels should use the TL084BC op-amp, and the 50kHz bandwidth channels should use the THS4032 op-amp. Both the 4-20mA and 0-10V channels will use the input interface shown in Figure 5.1 and Figure 5.2, but with input shunt resistor (250 Ω) removed for the 0-10V channels. Furthermore, the 4-20mA and 0-10V channels will use the same differential amplifier and signal-scaling stages (Figure 5.3 and Figure 5.4).

5.2.7 Circuit Simulation

The circuit in Figure 5.1, Figure 5.2, Figure 5.3, Figure 5.4, Figure 5.5 and in Appendix M will be simulated using TINA-TI V9²⁰ circuit simulation tool. This is done to verify the design and evaluate its performance. The component values obtained will be used in the simulation (Appendix U). The circuit shown in Figure 5.1, Figure 5.2, Figure 5.3, Figure 5.4 and Figure 5.5, once cascaded together, will form a single 5kHz bandwidth channel for 4-20mA and 0-10V signals. In addition, Figure 5.1, Figure 5.2, Figure 5.3, Figure 5.4 and the circuit shown in Appendix M, cascaded together, will form a single 50kHz bandwidth channel for 4-20mA and 0-10V signals. The op-amp will be simulated and supplied with a +12V and -12V (split power supplier).

²⁰ http://e2e.ti.com/support/development_tools/webench_design_center

5.2.7.1 AC and DC Performance 5kHz Channel

The circuit in Figure 5.1 converts current to voltage; all the current passes through the resistor. It will only be used with 4-20mA signals. The circuit shown in Figure 5.1 and Figure 5.2 is cascaded together, and their DC performance is simulated. After that, circuits shown in Figure 5.1, Figure 5.2, Figure 5.3, Figure 5.4 and Figure 5.5 are all cascaded together to simulate the overall performance of the 5kHz channel. The output of the DC simulation in Figure 5.7 was not expected, as it gives an output voltage of 1.62V with an input voltage of 1V ($4\text{mA} \times 250\Omega$) and output voltage of 5.62V with an input voltage of 5V ($20\text{mA} \times 250\Omega$). There is an error of 0.62V, which is not acceptable, as it is too high. This is due to the fact that the 4-20mA signal is a differential signal and the input stage is not referenced to op-amp ground, thus floating.

To improve DC performance and reduce the DC error in line with design expectation, the input signal must be referenced to the op-amp ground (Figure 5.8). The output voltage is 999.98mV with an input signal of 1V. This gives an error of $20\mu\text{V}$ on the low side (4mA input). With the input voltage of 5V ($20\text{mA} \times 250\Omega$), the output signal is 5V. The current source used at the input is assumed to have a very high internal resistance (i.e. $>100\text{M}\Omega$). The DC transfer characteristics graph is shown in Figure 5.9. The input signal and output signal have a linear relationship (Figure 5.9).

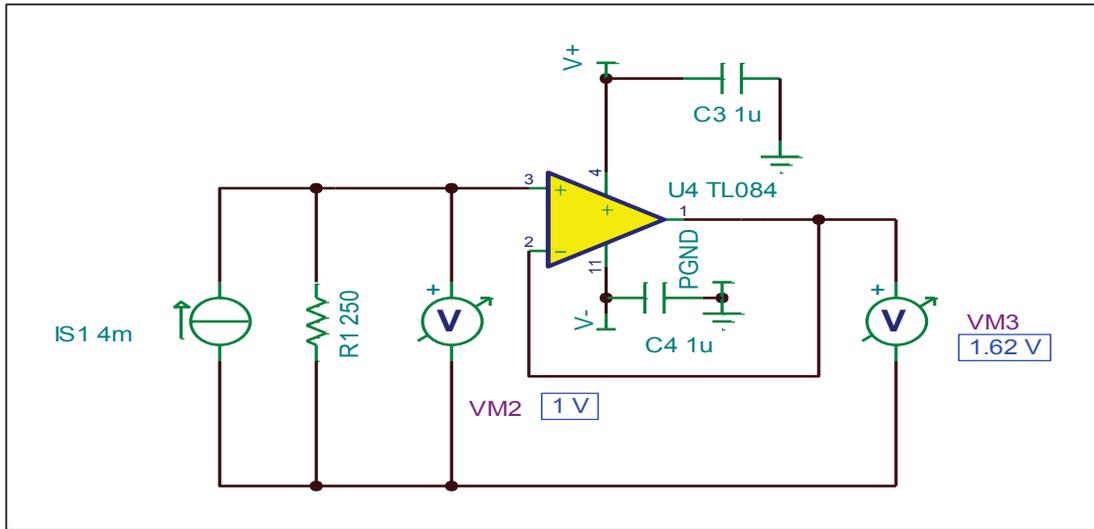


Figure 5.7 4-20mA receiver and buffer stage with poor DC performance

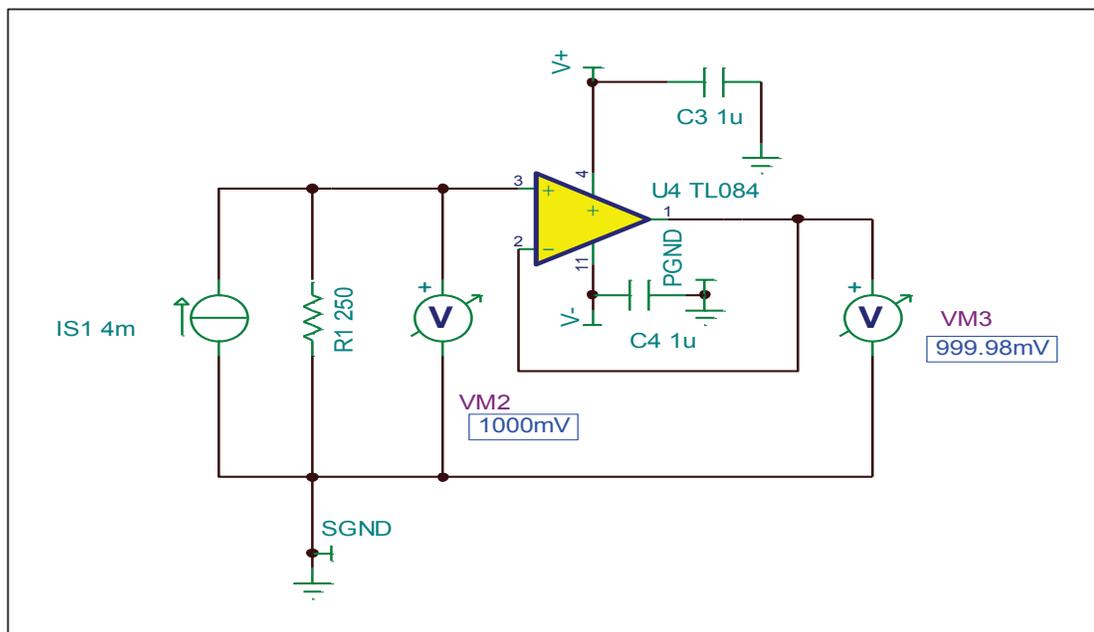


Figure 5.8 4-20mA receiver and buffer stage with good DC performance

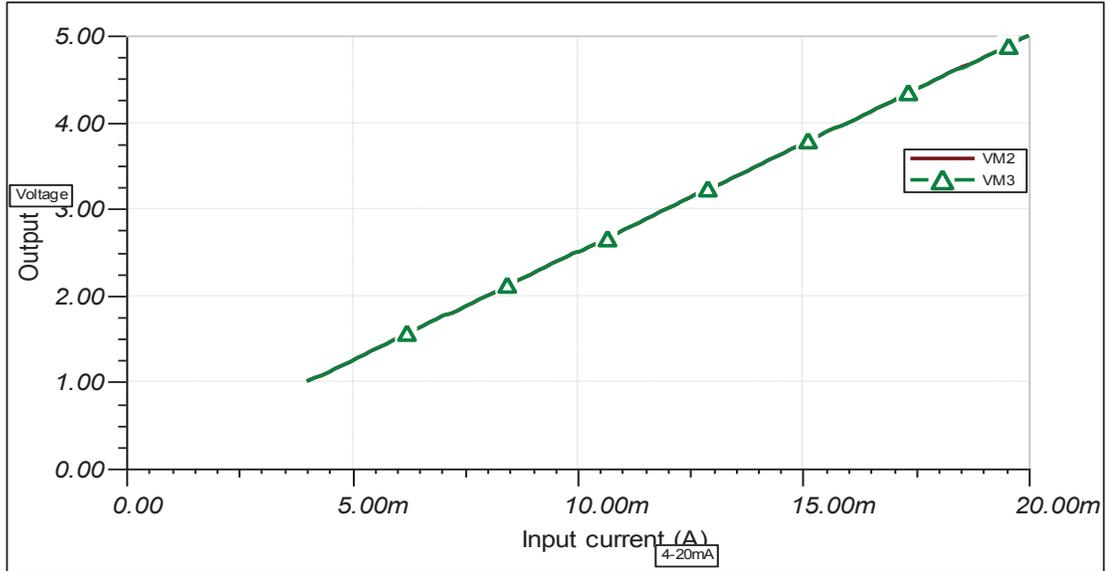


Figure 5.9 Input and output DC transfer characteristics for 4-20mA signal

For 0-10V input signal simulation, the shunt resistor must be removed (Figure 5.10). The output signal follows the input signal with the same error performance as the 4-20mA inputs, and the DC transfer characteristic is also linear (Figure 5.11). However, when the input voltage is set to 0V, the output voltage is $10.97\mu\text{V}$. This voltage can be taken as the TL084BC op-amp input offset voltage and is much lower than specified in the datasheet. The voltage source VG1 used at the input is assumed to have a very low internal resistance (i.e. $\approx 0\Omega$).

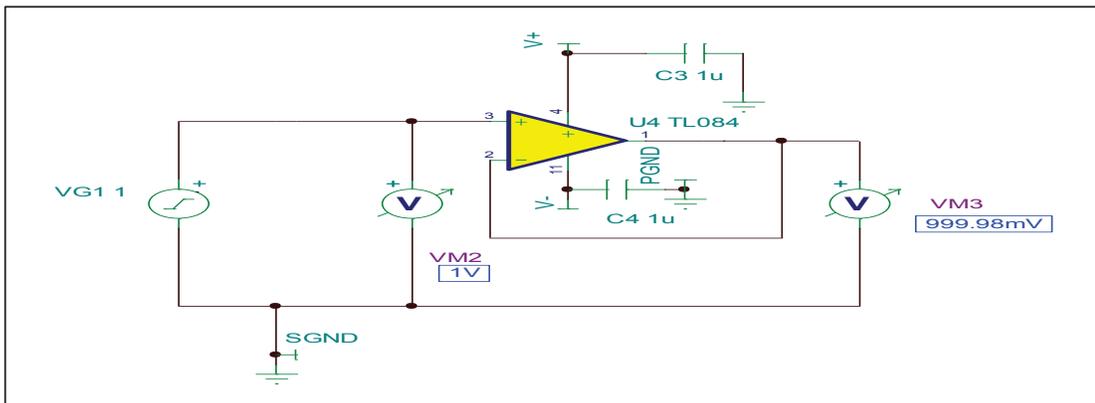


Figure 5.10 0-10V Receiver and buffer stage DC performance (5kHz channel)

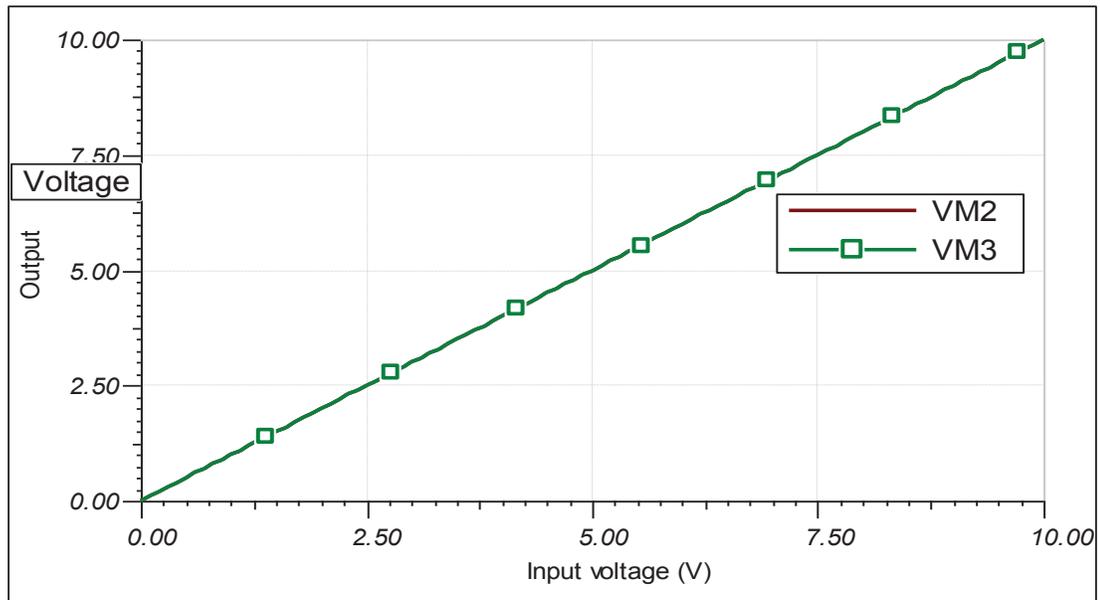


Figure 5.11 Input and output DC transfer characteristics for 0-10V signal (5kHz channel)

Both circuits shown in Figure 5.8 and Figure 5.10 show a good AC performance when the input signal frequency is swept from 1Hz to 5kHz. The AC transfer characteristic is shown in Figure 5.12. The gain is approximately equal to unity ($GAIN \approx 0.99997$). The AC performance is limited by the TL084BC op-amp frequency response.

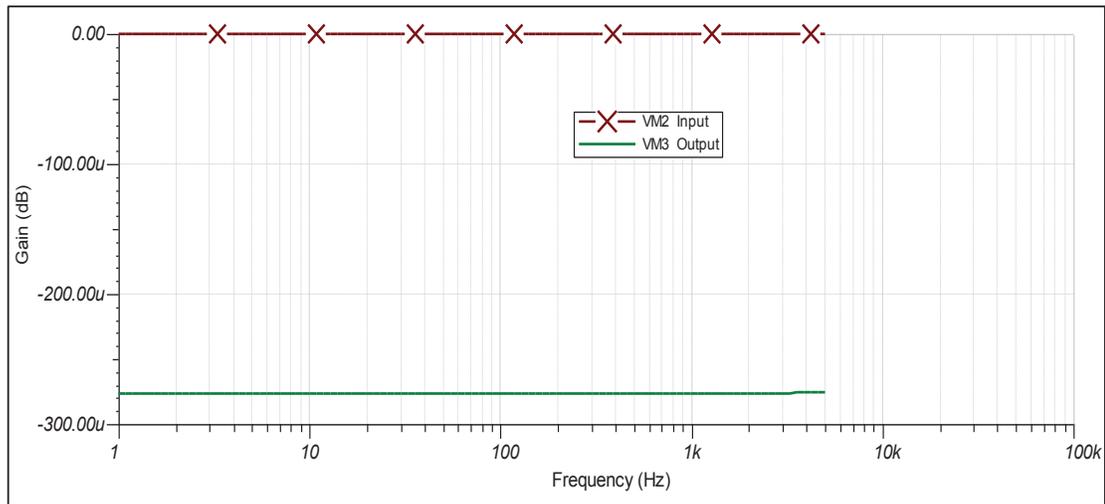


Figure 5.12 Signal receiver and buffer stages AC response (5kHz channel)

The DC simulation is performed with the input buffer amplifier, differential amplifier and the scaling buffer stage cascaded together. The output signal of the buffer is fed into the differential amplifier's positive input and the negative input pulled to op-amp ground via a 10k Ω resistance. This is a special case for differential amplifier inputs. The output of the differential amplifier is fed through a voltage divider network into a buffer amplifier. The buffer amplifier output voltage is 499.98mV with an input signal of 499.98mV, thus resulting in zero errors. The actual input signal is 1V (4mA x 250 Ω) scaled by 0.5. It can be seen from the simulation results that there is an error of 20 μ V propagated through the channel (Appendix N).

To have a true differential signal for the differential amplifier input stage, the reference ground is removed from the negative input at the input stage. This resulted in an increased output error. The values obtained from simulation results is approximately 50 μ V (Appendix O). The increase in error could be due to impedance mismatch between the differential inputs and reference ground mismatch between the current source and the op-amp ground reference. The AC performance is the same as shown in Figure 5.12. Both the 4-20mA and 0-10V signals have output signals which have a linear relationship with the input. However, the output signal of 4-20mA signal is scaled by 0.5, and for the 0-10V signal, it is scaled by 0.25.

The DC transfer characteristic for 4-20mA signal is illustrated in Figure 5.13, and for the 0-10V signal, it is shown in Figure 5.14.

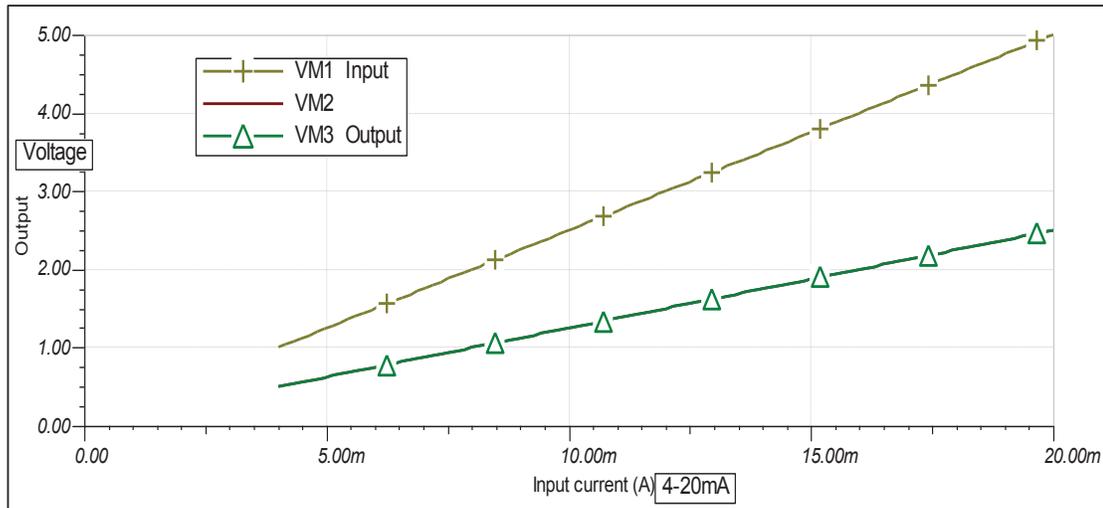


Figure 5.13 Input and output DC transfer characteristics for 4-20mA signal scaled by 0.5 (5kHz channel)

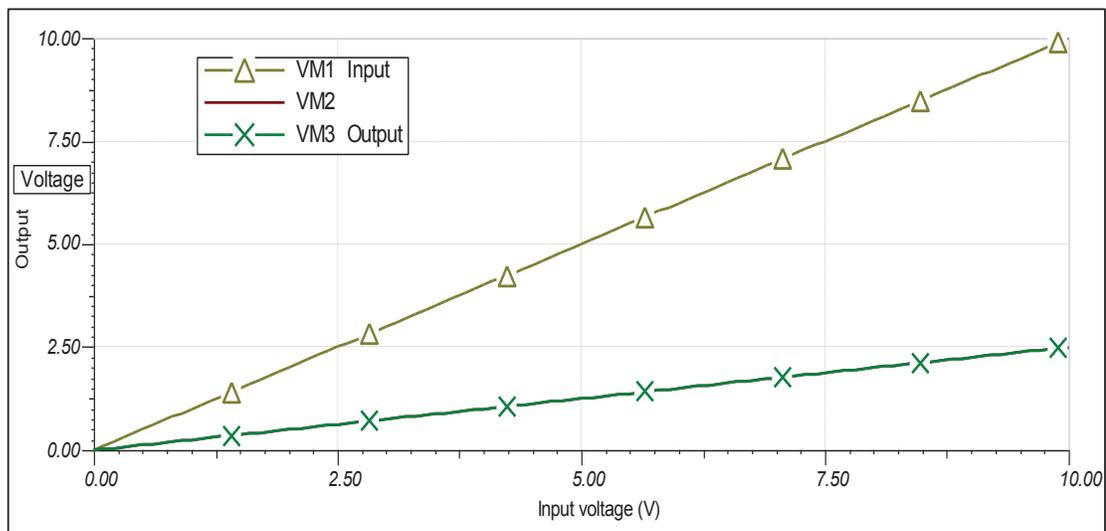


Figure 5.14 Input and output DC transfer characteristics for 0-10V signal scaled by 0.25 (5kHz channel)

The DC simulation is performed with the input buffer amplifier, differential amplifier and the scaling buffer, as well as the 5kHz lowpass filter stage cascaded together (Appendix P). The filter output voltage is 499.97mV with an input signal of 499.98mV, thus producing a 10 μ V error. The actual input signal is 1V (4mA x 250 Ω) scaled by 0.5, giving a 500mV input signal.

The overall DC performance of the 5kHz channel has an error of 30 μ V, with a 4mA signal input. However, on the high-side input 5V (20mA x 250 Ω), simulation results show zero errors at the output signal. The DC transfer characteristic of the 5kHz channel for 4-20mA signal is depicted in Figure 5.15. Also, the DC transfer characteristic of the 5kHz channel for the 0-10V signal is shown in Figure 5.16, where the input signal is scaled by 0.25.

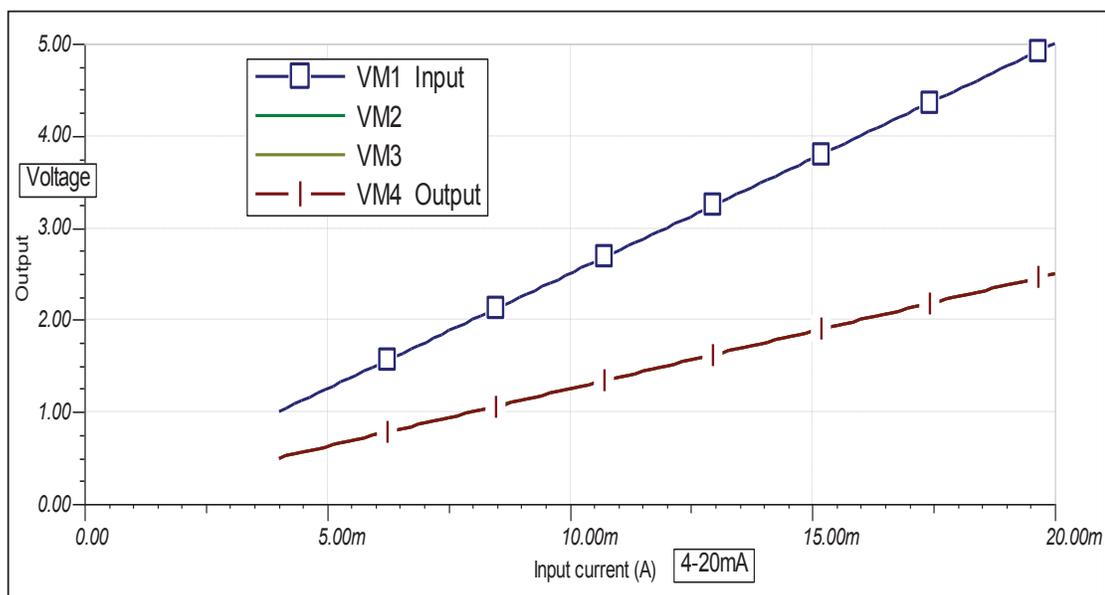


Figure 5.15 DC transfer characteristics for 4-20mA signal scaled by 0.5 (5kHz channel) with lowpass filter stage

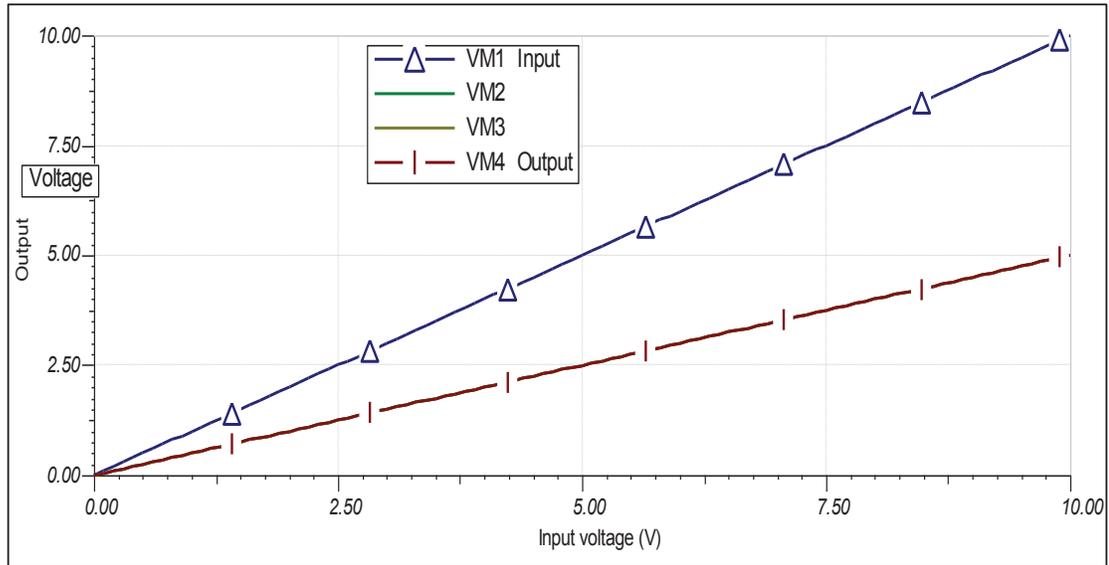


Figure 5.16 DC transfer characteristics for 0-10V signal scaled by 0.25 (5kHz channel) with lowpass filter stage

The channel bandwidth is limited by the lowpass filter stage with a cut-off frequency of 5kHz, hence limiting the channel bandwidth to 5kHz. The AC transfer characteristic of the filter stage is presented in Figure 5.17, and the -3dB point is indicated in Figure 5.18 with a cut-off frequency of 4.98kHz. This value is comparable to the calculated value of $F_c = 4973.59 \text{ Hz}$ (4.97kHz).

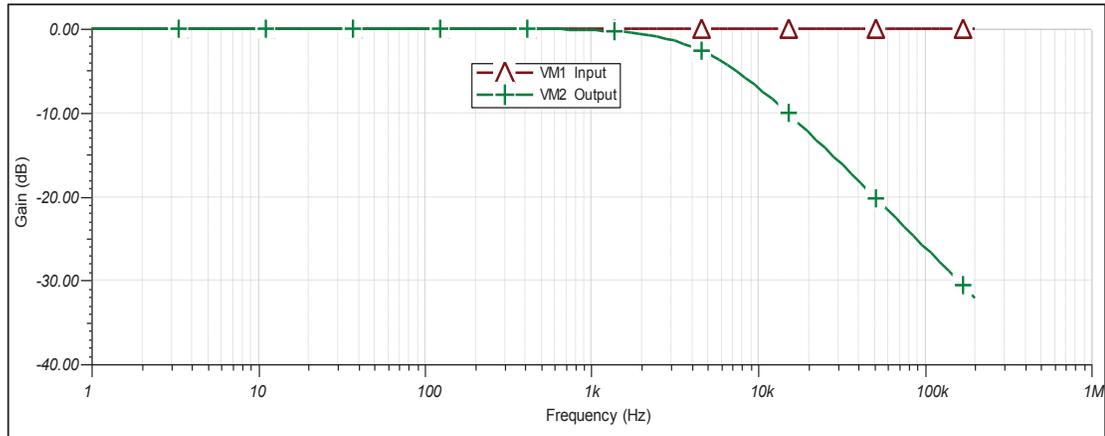


Figure 5.17 AC transfer characteristics lowpass single-order filter (5kHz channel)

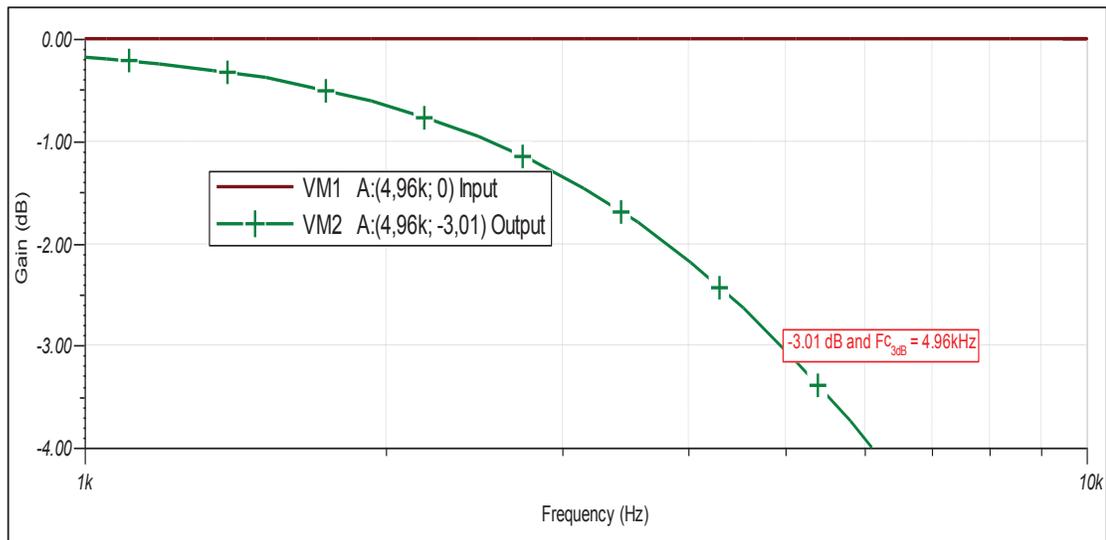


Figure 5.18 AC transfer characteristics lowpass single-order filter -3dB point (5kHz channel)

The overall AC transfer characteristics of the 5kHz channel are highlighted in Figure 5.19. The input signal is swept from 1Hz to 200kHz. The signal at the Nyquist bandwidth will be attenuated by -40dB (0.01), which means a 2.5V signal will be attenuated to 25mV above Nyquist bandwidth frequencies ($> 100\text{kHz}$) (Figure 5.19).

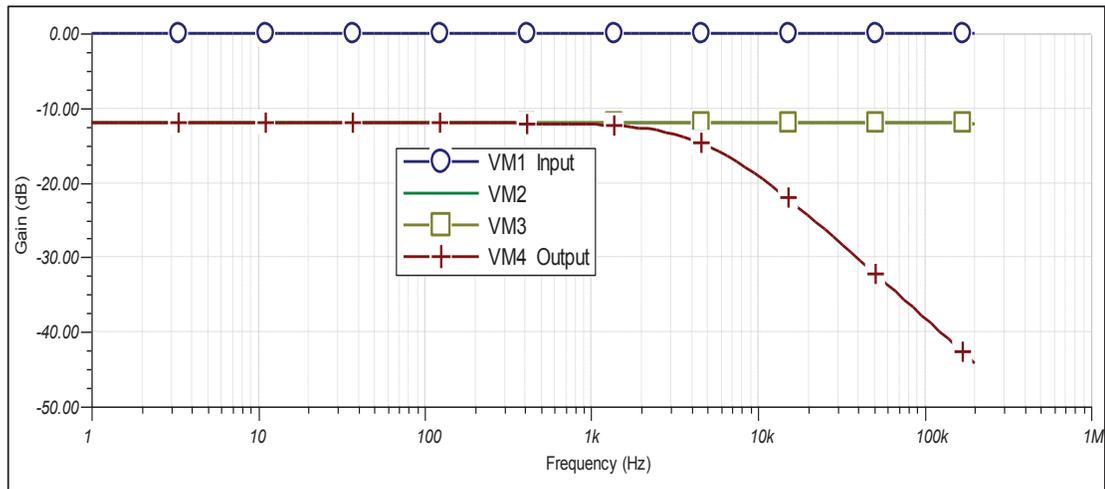


Figure 5.19 Overall AC transfer characteristics (5kHz channel)

5.2.7.2 AC and DC Performance 50kHz Channel

The 5kHz and 50kHz channels use the same input circuit configurations discussed above, and with major differences being the choice of op-amp used in realising the design and filter stage. The TL084BC op-amp is chosen in the design of 5kHz channel, and the THS4032 op-amp was chosen for 50kHz channel design. The 50kHz channel output stage consists of a seven-order lowpass filter (Appendix M). A similar approach employed above will be followed to evaluate the 50kHz channel performance. The circuit shown in Figure 5.1 and Figure 5.2 are cascaded together and simulate their DC performance. Moreover, all stages are cascaded together to simulate the overall performance of the 50kHz channel.

The circuit and simulation results are shown in Figure 5.20. The output voltage is 999.17mV with an input signal of 999.24mV and must be 1V. This suggests that there is a voltage drop of 720 μ V between the shunt resistor and the positive input pin of the buffer. However, this is not the case, as this voltage error must be generated internally within the op-amp. This will result in an error of 830 μ V (1V- 999.17mV) at the output with a signal input of 4mA.

With the input voltage of 5V (20mA x 250 Ω), the output signal is 5V. For the 0-10V signal, the results are shown in Figure 5.21 and Figure 5.22. In Figure 5.21, the input signal is set at

1V and the output signal is 999.93mV, resulting in an error of $70\mu\text{V}$. Furthermore, in Figure 5.22, the input signal is 10mV, and the output signal is 9.95mV, thus giving an error of $50\mu\text{V}$. The buffer generates a $-51.58\mu\text{V}$ output signal with zero input voltage, suggesting a negative input offset voltage (lower than specified in the datasheet). The DC transfer characteristics graphs for 4-20mA and 0-10V signals are shown in Figure 5.23 and Figure 5.24. The input and output signals have a linear relationship.

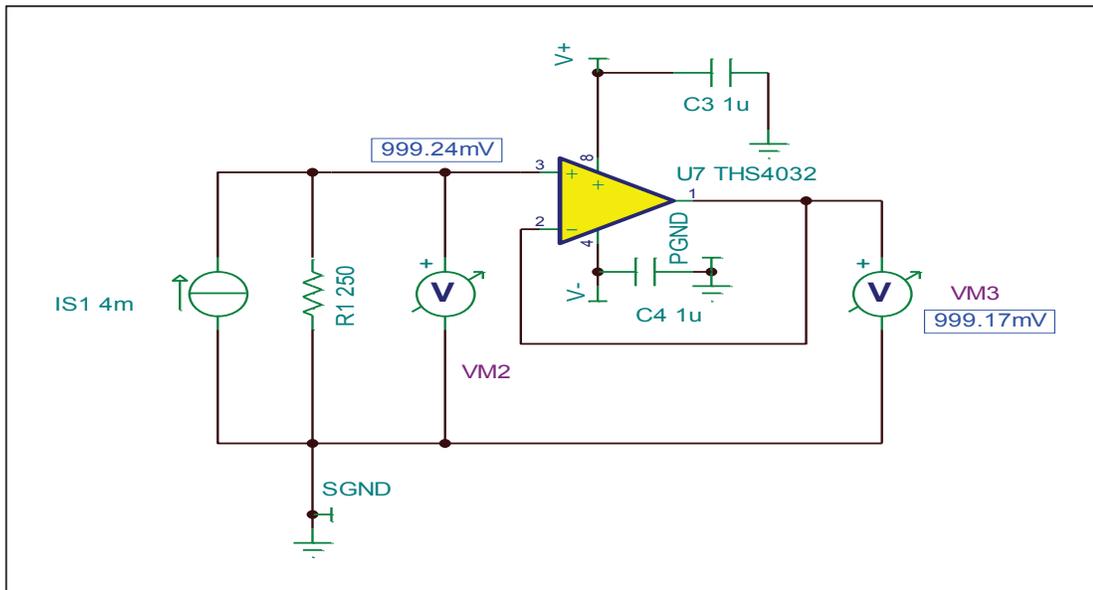


Figure 5.20 4-20mA receiver and buffer 50kHz channel (with 4mA input signal)

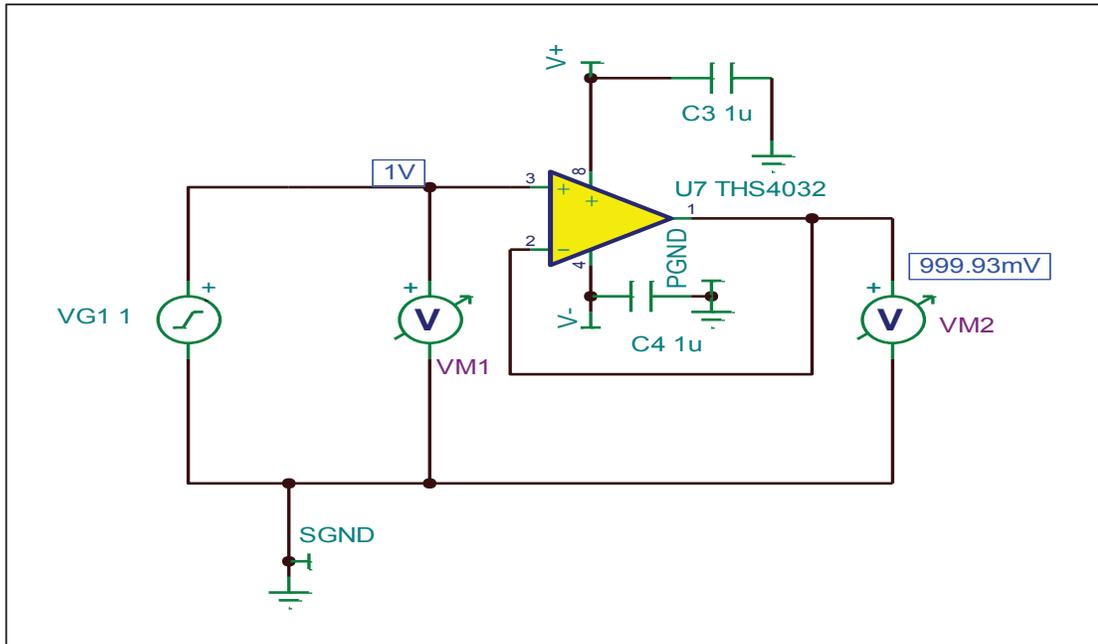


Figure 5.21 0-10V receiver and buffer 50kHz channel (with 1V input signal)

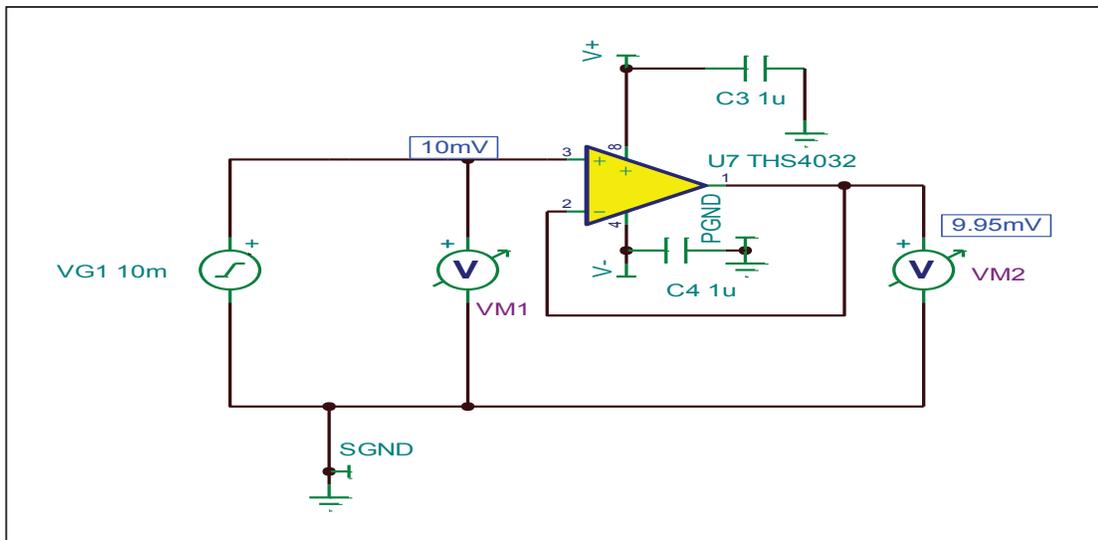


Figure 5.22 0-10V Receiver and buffer stage 50kHz channel (with 10mV input signal)

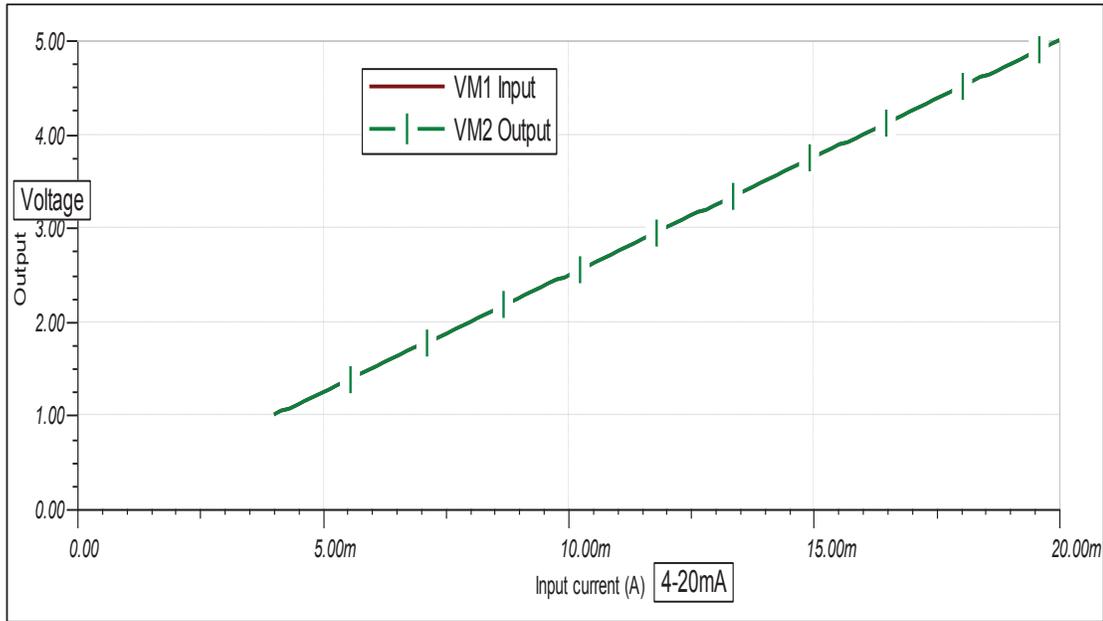


Figure 5.23 Input and output DC transfer characteristics for 4-20mA signal (50kHz channel) receiver and buffer stage

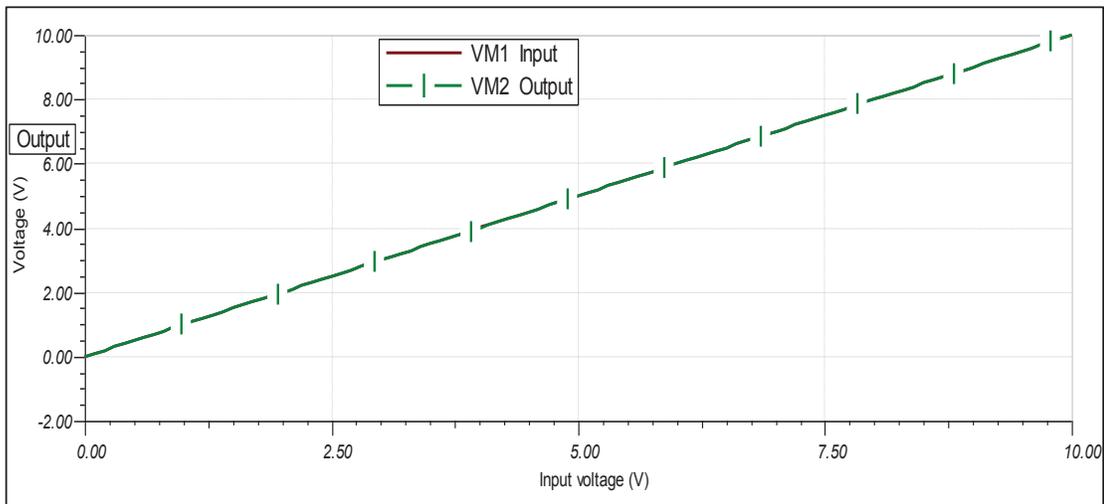


Figure 5.24 Input and output DC transfer characteristics for 0-10V signal (50kHz channel) receiver and buffer stage

The DC simulation is performed with the input buffer amplifier, differential amplifier and the scaling buffer stage cascaded together (Appendix Q). The output of the differential amplifier is fed through a voltage divider network into a buffer amplifier (scaling buffer stage). The buffer amplifier output voltage is 484.3mV with an input signal of 484.36mV, resulting in a 60 μ V error. The actual input signal was supposed to be 1V (4mA x 250 Ω), but it is equal to 999.24mV, and the input stage buffer output signal is 999.17mV; this output signal should be scaled by 0.5 to 499.59mV. Therefore, the expected signal from the output buffer is approximately 499.59mV. However, the actual output signal is 484.3mV from an input signal of 484.36mV. This means the THS4032 op-amp causes an error of 15.23mV (499.59mV - 484.36mV) in the scaling buffer input stage.

The reference ground is removed from the negative input in the input stage, thus generating a differential signal for the differential amplifier input stage. The simulation results are similar to results shown in Appendix Q. The shunt resistor is removed and then a 1V input signal applied in the input (Appendix R). The signal expected at the scaling buffer input is equal to 249.93mV (999.73mV x 0.25). Nonetheless, this expectation is not met (see Appendix R), as the actual signal value is 242.39mV. This means the THS4032 op-amp causes an error of 7.54mV (249.93mV - 242.39mV) in the input stage of the scaling buffer. The simulations suggest that the THS4032 op-amp has poor DC performance compared to the TL084BC op-amp.

The DC transfer characteristics of the 4-20mA and 0-10V signal are shown in Figure 5.25 and Figure 5.26.

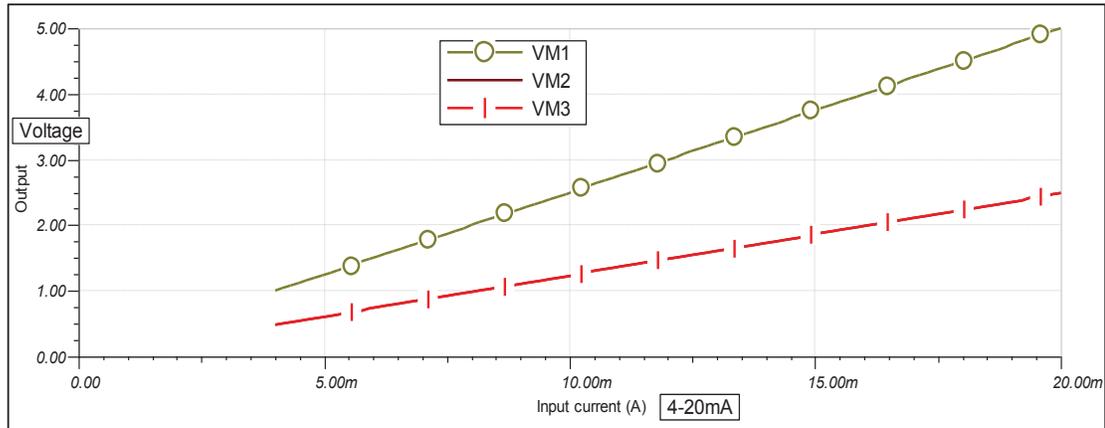


Figure 5.25 DC transfer characteristics for 4-20mA signal scaled by 0.5 (50kHz channel)

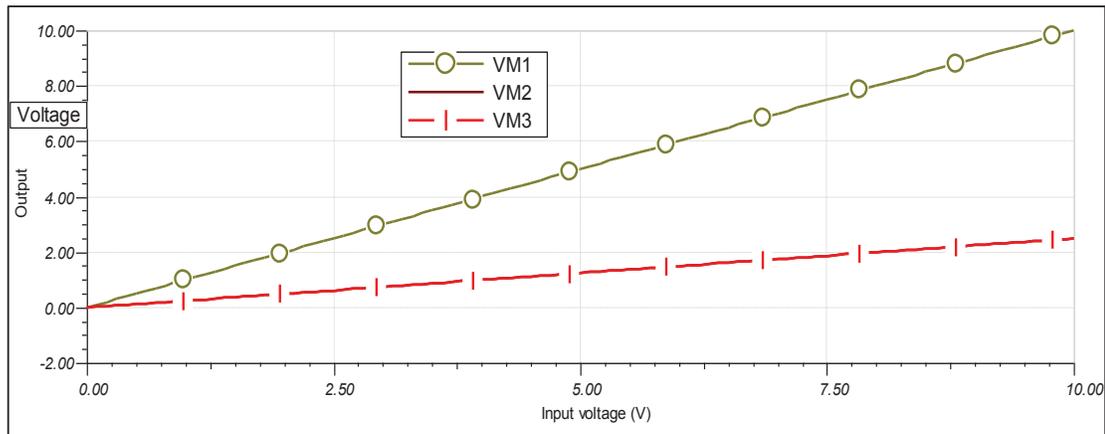


Figure 5.26 DC transfer characteristics for 0-10V signal scaled by 0.25 (50kHz channel)

From Figure 5.27, it can be seen that the results of the simulation are not as expected. The output signal is about 599.57mVrms and the expected is approximately 500mVrms, which gives an error of 99.57mVrms; this is with an input signal of 1Vrms ($4\text{mA} \times 250\Omega$). The error decreases to 10mV with an input signal of 5Vrms ($20\text{mA} \times 250\Omega$). From the simulation, the AC performance seems to be worst on the low side of the input range (4mA), which is similar to DC performance.

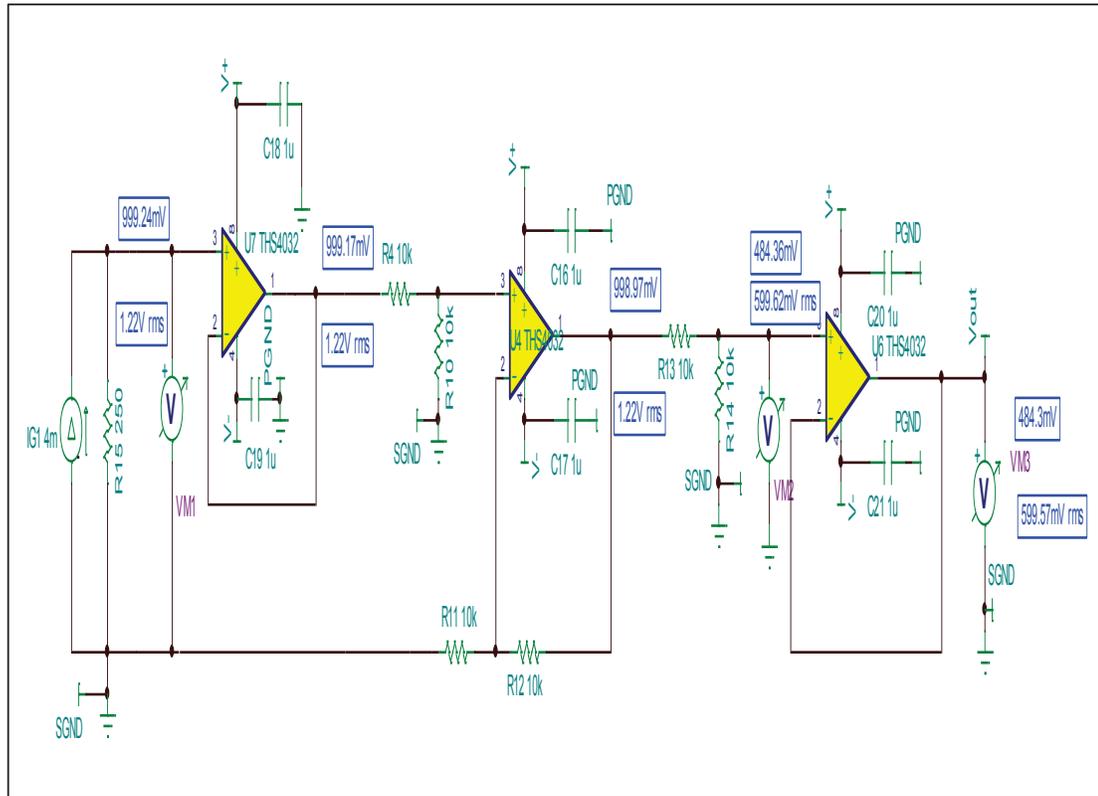


Figure 5.27 Signal receiver, buffer and scaling stages AC performance (50kHz channel)

The DC simulation is performed with the input buffer amplifier, differential amplifier and the scaling buffer, as well as the seven-order lowpass filter stage cascaded together. The seven-order filter stage circuit diagram is shown in Appendix S. With an input of 1V, there is an overall error of approximately 44mV ($1V - 956.21mV$) from the input through the seven-order filter. The DC performance of the 50kHz 4-20mA channel with 1V ($4mA \times 250 \Omega$) input has an overall DC error of approximately 59.22mV ($(1V \times 0.5) - 440.78mV$) and for a 5V ($20mA \times 250 \Omega$) input DC error of approximately 60mV ($(5 \times 0.5) - 2.44V$). The overall DC transfer characteristics of the 50kHz channel, with 4-20mA signal and 0-10V signal, are shown in Figure 5.28 and Figure 5.29.

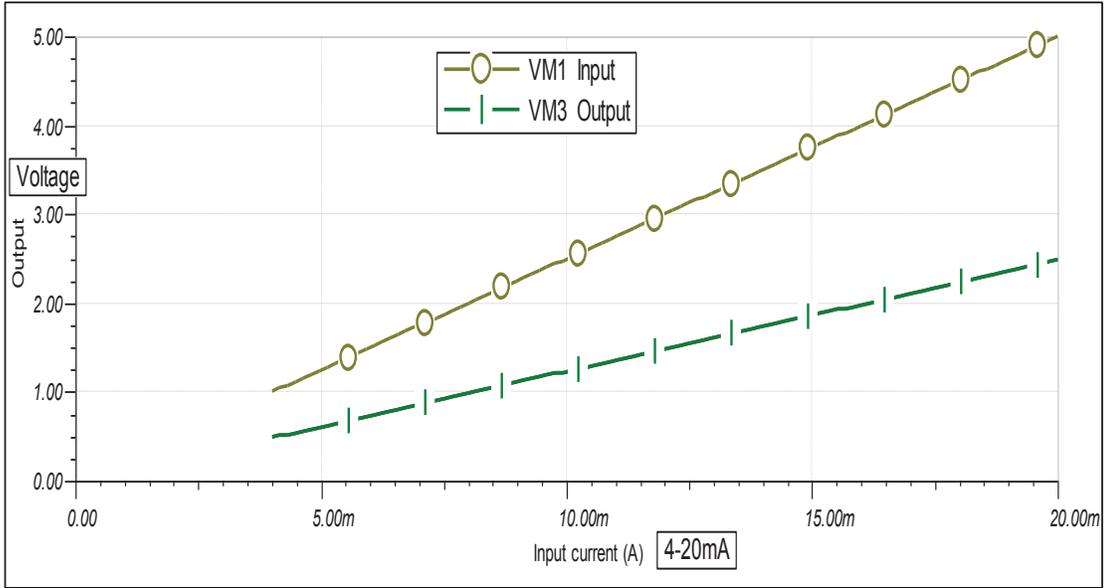


Figure 5.28 DC transfer characteristics for 4-20mA signal scaled by 0.5 (50kHz channel) with filter stage

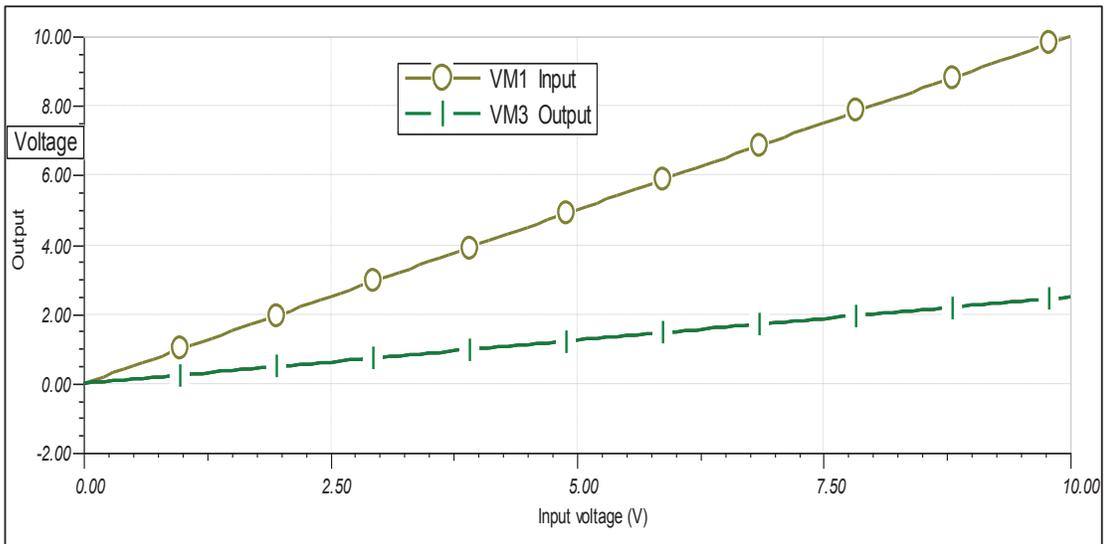


Figure 5.29 DC transfer characteristics for 0-10V signal scaled by 0.25 (50kHz channel) with filter stage

The 50kHz channel bandwidth is limited by the lowpass filter stage with a cut-off frequency of 50kHz, thus limiting the channel bandwidth to 50kHz. The AC transfer characteristic of the

filter stage is shown in Figure 5.30, and the -3dB point is shown in Figure 5.31 with a cut-off frequency of 49.98kHz. This value is comparable to the calculated value of $F_c = 49735.91 \text{ Hz}$ (49.74kHz).

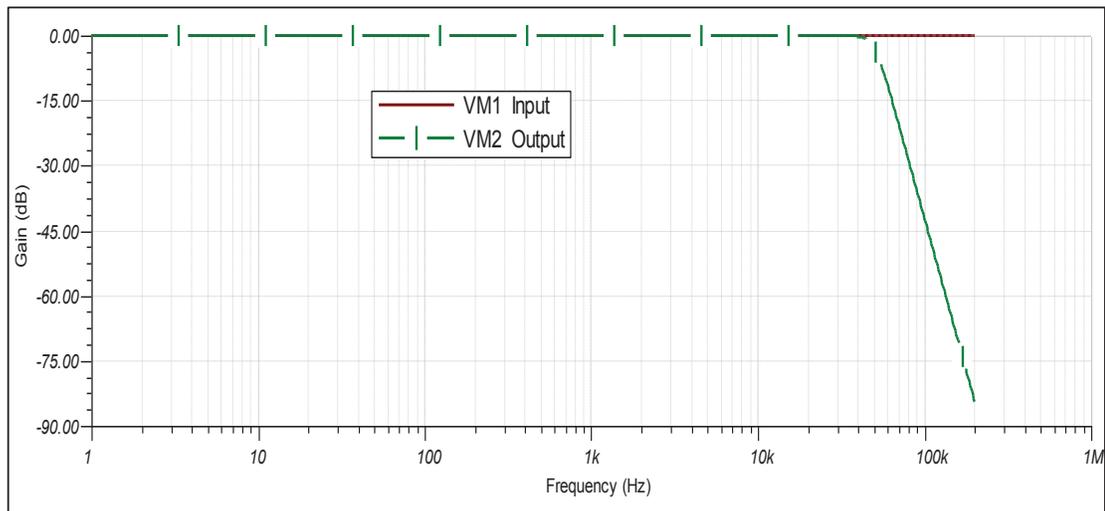


Figure 5.30 AC transfer characteristics lowpass seven-order filter (50kHz channel)

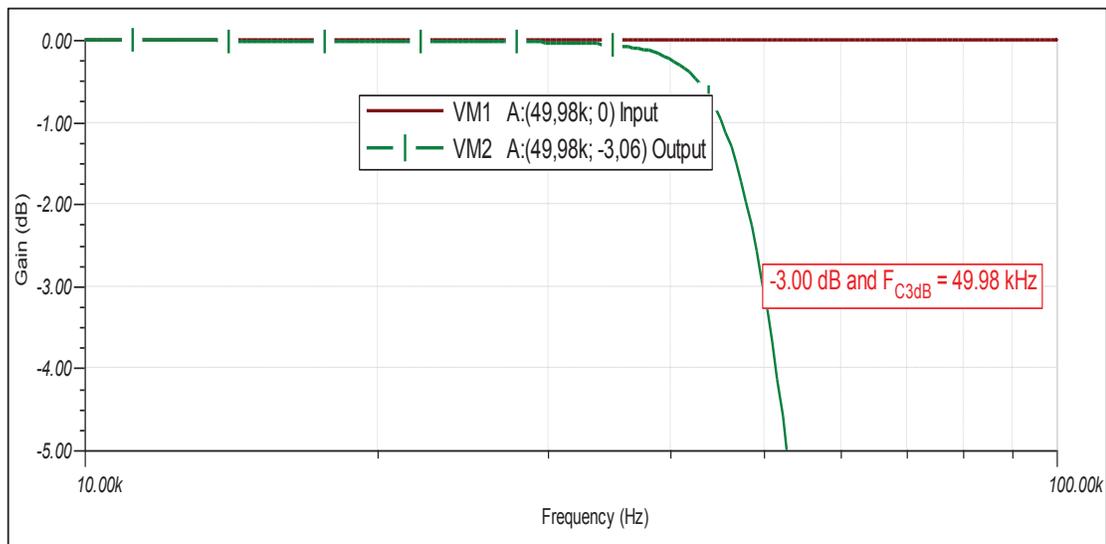


Figure 5.31 AC transfer characteristics lowpass seven-order filter -3dB point (50kHz channel)

The overall AC performance of the 50kHz channel is depicted in Figure 5.32 for 0-10V signal and in Figure 5.33 for 4-20mA signal. The input signal is swept from 1Hz to 200kHz.

The 50kHz channel has a sharper signal roll-off above the cut-off frequency. The signals above Nyquist bandwidth ($> 100\text{kHz}$) will be attenuated by about -45dB (0.005), which will attenuate a 2.5V signal down to 14mV .

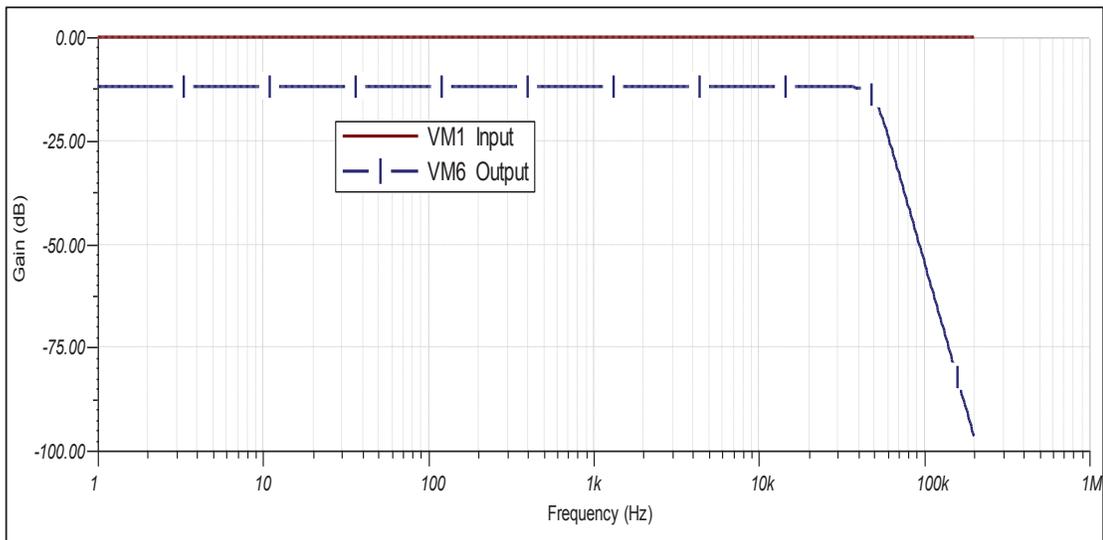


Figure 5.32 Overall AC transfer characteristics (50kHz channel) 0-10V signal

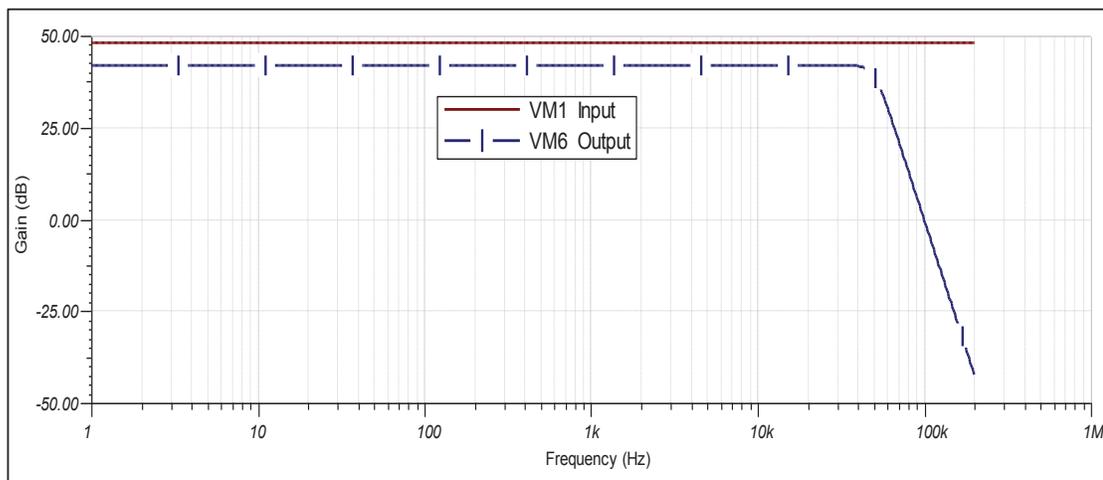


Figure 5.33 Overall AC transfer characteristics (50kHz channel) 4-20mA signal

5.3 ADC

The ADC stage's only task is to transform the analogue signal from the signal conditioning stage to a digital form. It must be able to scan through all the 16 channels, process each channel and output a stream of digital data representing input signals for each channel. This is essentially a multichannel data acquisition system. The aim is not to design an ADC from scratch, but it is to identify and choose the appropriate ADC architect and technology.

5.3.1 ADC Architecture

There are numerous ADC architects and topologies used in measurement science. However, the choice in this study will be limited to two favoured ADC architects and topologies used in measurement and instrumentation systems. These are:

- Sigma-Delta Converter
- Successive Approximation Register (SAR)

5.3.2 Sigma-Delta Converter

This converter uses an oversampling technique during the conversion process and is useful where a high resolution is required. The oversampling process helps in pushing out the quantisation noise out of the input signal bandwidth (Rauth and Randal, 2005) and allows the use of single-order anti-aliasing filters. This helps in improving the resolution and accuracy of the ADC, thus resulting in a good signal-to-noise ratio (SNR). However, the main disadvantage of the sigma-delta converter is the conversion speed; it has a lower conversion bandwidth.

5.3.3 Successive Approximation Register (SAR)

SAR is the most-used ADC architect in instrumentation, due to its relative low cost, is less complex, good resolution (less than that of delta-sigma converters) and higher conversion speed (more than that of sigma-delta converters). SAR ADC uses the binary search technique to perform the conversion process. Basically, a search is performed through all the possible

quantisation levels, converging to and outputting a value that is closest to the sampled input signal (Rauth and Randal, 2005).

SARs are the most favoured topology to be integrated into mixed signals IC. This mixed-signals IC uses the latest trends in very-large-scale integration (VLSI) techniques in order to integrate both analogue and digital circuits and subsystems into one chip. The digital subsystems will include a microcontroller or microprocessors, memory, multiplexer and I/O blocks. The analogue subsystems may include sample and hold section integrated into the same chip as the ADC (Ball, 2004). The advantage of this is that system costs are reduced. Additionally, in practice, these minimise system integration risks, in terms of interfacing a stand-alone ADC and a microprocessor.

Sigma-Delta ADC is suitable for high resolution and low throughput applications and is available as a stand-alone IC. SAR ADC offers both good resolution (> 8 bits) and high throughput and are used as ADC in microprocessor I/O blocks, thus giving an integrated solution with minimal interfacing risks.

5.3.4 ADC Design Specifications

As mentioned in literature, there are some critical ADC specifications that will influence the overall performance of the DAS system. The aim is to choose an ADC with good performance in order to minimise errors due to the relevant parameters.

5.3.4.1 Resolution

The full-scale input ranges of the analogue signal that can be expected are shown in Appendix V. The input range will depend on the scaling factor used to scale the output signal applied to filter stages. The resolution (equation (2.22)) is given as a function of the reference voltage and code width (number of bits) of the ADC. However, the resolution can also be taken as the number of bits transmitted out of the ADC after conversion (Pease, 2008). The reference voltage or full-scale range (FSR) is expressed as the difference between the minimum and maximum signal inputs. The minimum input signal in this case will be $1V \times \text{Scale Factor}$, and

the FSR is equal to $5V \times \text{Scale Factor}$. The typical resolution values are shown in Appendix V.

5.3.4.2 Acquisition and Conversion Time

The SAR ADC input stages (i.e. sample-and-hold) require a finite time to sample and capture the input signal. This finite time is known as the acquisition time (t_{acq}) (Pease, 2008). Some finite time is still required for the ADC to convert acquired signal to a binary or digital equivalent. This time is known as the ADC conversion time (t_{conv}) (ibid.). This means there must be some elapsed time before data is available from the ADC for a single channel data acquisition. This time is given as the sum of the acquisition and conversion time ($T_{ADC} = t_{acq} + t_{conv}$). This time parameter sets the maximum sampling frequency for a channel and expressed as (Kester, 2005a; Perez, 2002):

$$F_{smax} \leq \frac{1}{t_{acq} + t_{conv}} \quad (5.13)$$

In this case, the maximum sampling frequency is 200kHz ($F_{smax} = 200 \text{ kHz}$), thus $T_{ADC} = 5\mu\text{s}$. However, for ADC to handle this sampling frequency comfortably, T_{ADC} must be much less than $5\mu\text{s}$ (i.e. $T_{ADC} \ll 5\mu\text{s}$). This timing parameter (T_{ADC}) is only valid for single-channel acquisition. Nevertheless, for multichannel data acquisition, the required timing is reduced and is given as (ibid.):

$$F_{Sperchannel} = \frac{F_{smax}}{C_{no}} \quad (5.14)$$

where C_{no} is the number of channels and $F_{Sperchannel}$ is the sampling rate per channel. In this project, the number of channels required is 16 ($C_{no} = 16$). Therefore, the sampling rate per channel is $F_{Sperchannel} = 12.5\text{kHz}$. This limits the input signal bandwidth to approximately 5kHz. This means the minimum timing parameter T_{ADC} per channel is $80\mu\text{s}$. Thus, it is required that T_{ADC} per channel must be much less than $80\mu\text{s}$ (at least 10 times less) to comfortably handle signals with 50kHz bandwidth. Another ADC parameter which is a function of acquisition and conversion time is the ADC throughput or data rate. The ADC

throughput is given as (Hamid et al., 2013):

$$D_{rate} = Ch_{no} \times F_{smax} \times Sample_{size} \quad (5.15)$$

where D_{rate} is the ADC throughput (Kbytes or Mbytes) per second, Ch_{no} is the number of sampled channels, F_{smax} is the maximum sampling frequency and $Sample_{size}$ is ADC bits sizes in bytes. If $F_{smax} = 200 \text{ kHz}$, $Ch_{no} = 16$ and $Sample_{size} = 1.25$ (10 bits), then the ADC throughput is estimated to be 4 Mbytes per second. This is a large amount of data which must be handled by the processors. Because of the internal sample and hold mechanism employed by SAR, timing parameter T_{ADC} will be affected, thus the accuracy of the SAR as well (Figure 5.34). The signal source resistance (R_s) plays a significant role in reducing timing errors due to SAR input mechanism.

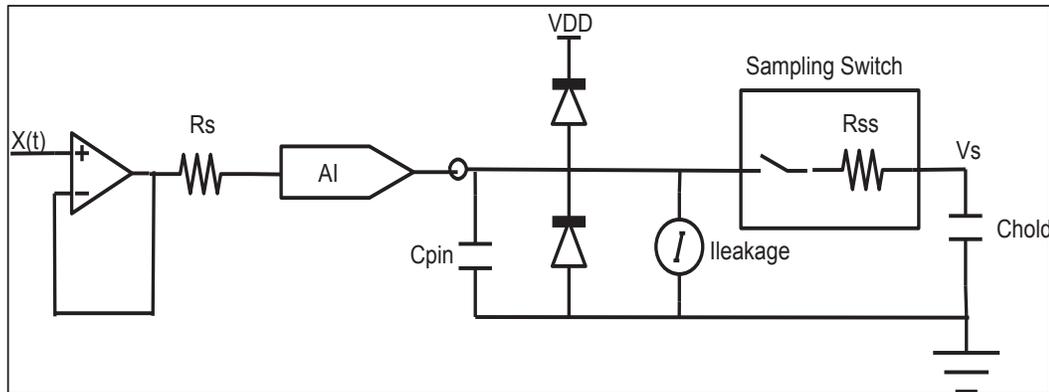


Figure 5.34 SAR input model

Source: Bowling (2002) and Pease (2008)

From Figure 5.34, the RC time constant of the signal path is given as:

$$t_{RC} = (R_S + R_{SS}) \times (C_{pin} + C_{hold}) \quad (5.16)$$

The voltage rise time across C_{hold} is approximately $(R_S + R_{SS}) \times (C_{pin} + C_{hold})$. In practice, errors due to leakage current and input capacitance (C_{pin}) are very small, thus negligible, and

switch resistance (R_{SS}) is usually between 20Ω - $1\,000\Omega$ (Baker, 2003; Pease, 2008). The hold capacitor (C_{hold}) and switch resistance (R_{SS}) values are specified in the manufacturer datasheet, and when choosing ADC, these values must be taken into account. Therefore, the main source of error will be signal source resistance (R_S), which must be as low as possible in order to minimise the filtering action of the ADC input stage. The minimum SAR ADC requirements are shown in Table 5.9.

Table 5.9 SAR ADC minimum requirements

| Parameter | Requirements | Notes |
|---|------------------------|----------------------------|
| Acquisition and Conversion Time T_{ADC} | $\ll 1\mu s$ | Acquiring all channels |
| No. of channels C_{no} | ≤ 16 | |
| Hold capacitor C_{hold} | $\ll 1nF$ | |
| Switch Resistance R_{SS} | $\leq 10\,000\ \Omega$ | |
| Code Width N | ≥ 10 bits | |
| Input capacitance (C_{pin}) | $\ll 7pF$ | Very low (i.e. negligible) |

5.3.4.3 DC Specifications

Ideally, the DC errors generated internally by the conversion process and semiconductor deficiency must be zero. However, in practice, there are finite errors that will affect the performance of SAR ADC. These errors include Gain, Offset, differential nonlinearity (DNL) and integral nonlinearity (INL) errors. However, both Gain and Offset errors can be corrected or calibrated out with software (Baker, 2011); nevertheless, DNL and INL errors cannot be corrected. The above errors have a major influence on the accuracy of the system; thus, a ADC with good DC specification is desired. These parameters are specified in ADC datasheets. The DC errors requirements are listed in Table 5.10.

Table 5.10 SAR ADC DC errors requirements

| Parameter | Requirements | Notes |
|---------------------------------|-----------------------------|---|
| Gain Error | $< \pm 6$ LSB | Can be calibrated out |
| Offset Error | $< \pm 6$ LSB | Can be calibrated out |
| Differential Nonlinearity (DNL) | ≥ 0 LSB and $< +1$ LSB | |
| Integral Nonlinearity (INL) | $\leq \pm 3$ LSB | |
| Absolute Error | $\leq \pm 3$ LSB | Includes Gain, Offset, DNL and INL errors |

5.3.4.4 AC Specifications

These specify the dynamic performance of an ADC. In other words, they specify the repeatability of the ADC conversion process (Pease, 2008) at specific sampling frequency and signal frequency. This indicates the level of input signal distortion due to the conversion process. These specifications are actually critical and important when measuring high bandwidth signals. In this project, the maximum input signal bandwidth is capped at 50kHz. These AC performance parameters are specified in ADC datasheets. The following errors are classified as ADC AC errors: SNR, Signal-to-Noise Ratio plus Distortion (SINAD), Effective Number of Bits (ENOB), Total Harmonic Distortion (THD) and Bandwidth. These errors cannot be corrected or calibrated out.

5.4 Controller

For the DAS system to be useful, it must acquire process and present data to the user correctly. This task is performed by the controller. The controller is actually a combination of hardware and software modules. The goal is not to design a microcontroller or microprocessor but to select the most suitable embedded platform based on the design requirements.

5.4.1 Embedded System Platform

This platform is the heart of the DAS system. It is the front end of the data acquisition system, where analogue signals are acquired, converted to digital form, pre-processed and streamed to the controller on the PC/laptop platform for real-time viewing and storage. In practice, embedded systems have limited resources (e.g. memory, I/O pins, processing speed) (Walls, 2012). This system must capture data in real time. This means it must acquire and process signal information without missing any signal changes generated by an external source (i.e. sensors), within specific time limits. The embedded system platform is made up of hardware and firmware. The hardware consists of a CPU, peripheral interfaces and other functional circuitries that the firmware (i.e. instructions) can access to control. The firmware is the instructions interpreted by a specific CPU to control all the internal hardware functions that interface to the external world (e.g. communication ports and I/O ports) in order to achieve a specific goal (i.e. reading sensors data).

It is proposed that a single-chip system (e.g. microcontroller or microprocessor) be used, as it has several advantages. These include lower costs and the fact that all data is processed and handled on the same CPU, thus making easier communication with the host system possible. This is because only a single protocol shall be required to send data from and to the host and embedded systems. The single CPU embedded system must have adequate peripherals, memory and processing capabilities.

5.4.1.1 Peripheral Requirements

The embedded system requires appropriate peripherals in order interface correctly to external devices (i.e. analogue signal conditioning board) and the host system. The following minimum peripherals are required on the embedded system:

- Analogue Interface
- Digital Interface

Analogue Interface

This interface provides the analogue to the digital interface required to collect the analogue signal. The embedded system shall provide an analogue interface to read and convert the analogue signal to digital form for processing by the CPU. This function is performed by a mixed-signal device (ADC) within the embedded system platform. The mixed-signal device should provide all the necessary supporting subsystems such as analogue switches or multiplexer and sample-hold circuitry (Figure 5.34) required on the ADC front end. It should also provide the control and status signals, and data bus or registers at the back end of the ADC.

Digital Interface

This interface shall provide the communication and single-bit input or output capabilities. The communication interface shall have the digital engine to send and receive data using specific protocol (i.e. RS232, USB, Ethernet) controlled by the CPU. It shall provide all the required electronic interconnect subsystems. Therefore, the digital interface's function is to provide the hardware layer required to communicate with the host system and drive the visual indicators (e.g. light-emitting diodes (LEDs)) showing system status. The minimum interface requirements are listed in Table 5.11.

Table 5.11 Digital interface requirements

| Parameter | Requirements | Notes |
|-----------------|--------------|--|
| Digital Inputs | 4 | Supporting 3.3V and 5V logics levels |
| Digital Outputs | 4 | Supporting 3.3V and 5V logics levels |
| Serial Port | 2 | UART or USB and supporting multi-baud rate |

5.4.1.2 Memory Requirements

An embedded system consists of two types of memory: one for temporary storage of data (RAM) and another for permanent storage of data (FLASH or EEPROM). The embedded system shall have adequate memory capacity for temporary storage of variables and data (RAM) and permanent storage of data such as program code and user data. The size of this memory is the function of how many variables will be used and the buffer size. The buffer size will be a major consumer of RAM memory. However, a larger buffer size will help in improving system performance (Stringham, 2009).

It is obvious from equation (5.15) that the ADC shall require a substantial amount of RAM from the embedded system. If $Ch_{no} = 16$, $F_{smax} = 200\text{kHz}$ and $Sample_{size} = 1.25$ bytes (10 bits) and 1.5 bytes (12 bits), then the amount of data from 10 and 12 bits ADC over one second would be 4 Mbytes/s and 4.8 Mbytes/s respectively.

The amount of data that can be stored in RAM for one second is estimated as (Heath, 2002):

$$D_{stored} = Data_{bits} \times Sample_{rate} \times T_D \quad (5.17)$$

D_{stored} is the data stored in bits, $Data_{bits}$ is the number of bits of the sampled data, $Sample_{rate}$ is the rate at which data is sampled and T_D is the interval in which data must be stored. It is evident from equation (5.17) that at the required sample rated of 200kHz, the memory requirements for 10 and 12 bits data is 250 and 300 Kbytes respectively. This is provided that data is accessed and transferred every one second. This memory size (RAM) is usually too big for most embedded systems (see Appendix T). However, in practice, T_D must be much smaller than one second, which reduces the required RAM substantially. Therefore, if T_D is kept at less than 10ms, then RAM requirements should easily be attainable for most low-cost embedded systems. The buffer shall be sized using guidelines provided in Table 7.1 (Stringham, 2009). The program memory (Flash or EEPROM) should be large enough to accommodate current and future firmware and data requirements.

5.4.1.3 Processor (CPU) Requirements

The processor is the heart of any embedded system, and the technology used is critical to the overall system performance. The processor's main task is to supervise (Doboli and Currie, 2011) and control the function of the entire embedded system. This will include data processing and control of interaction and interface to peripherals. This is achieved by executing instructions (stored in FLASH memory) developed to perform a specific task (Barrett and Pack, 2006).

The processor shall meet all the functional and non-functional requirements (Oshana, 2012b). The non-functional requirements include such parameters as bus timing and bandwidth. The functional requirements include adequate peripherals, memory size, I/Os and development tools (i.e. compilers, GUI programming environments, simulators, in-circuit programming capabilities (ISP)).

One of the fundamental differences between embedded systems CPU is the data and address bus bit width. This specifies the data register width (i.e. 8, 16 and 32 bits) and maximum addressable memory space (e.g. 0-256 = 2^8 , 0-65536 = 2^{16} and 0-4G = 2^{32}) that the CPU can handle. Therefore, this determines the bandwidth of the CPU. The bandwidth in this context specifies the rate at which data can be moved (Wolf, 2012) internally between subsystems, within the embedded system. This will include data through embedded system peripherals (i.e. serial ports and reading I/Os). The processor/CPU performance expressed as MIPS (millions instruction per second) is estimated as (Zhu and Dutt, 2009):

$$MIPS = D_{bandwidth} \times D_{work} \quad (5.18)$$

$D_{bandwidth}$ is the amount of data processed per second and D_{work} is taken as the number of instructions required to process data. The parameter D_{work} is highly dependent on the CPU architect, which is the function of CPU bit width. Therefore, depending on application of the CPU bit width, this may negatively affect the embedded system performance, especially an 8-bit architect. However, this depends on the CPU processing and clock speeds. The CPU

processing speed is derived from the clock speed and governs how fast instructions are executed, thus the rate the data (D_{work}) is processed. This also means the response time of the system is highly dependent on the CPU processing and clock speeds. It is advisable to choose an embedded system with a CPU having the highest MIPS and clock speed (Appendix T). The processor list shown in Appendix T is not exhaustive and should be used as a guide in choosing an appropriate microprocessor or microcontroller.

5.4.1.4 Firmware Requirements

The firmware is required to control the functioning of the embedded system hardware. As mentioned, firmware is a list of instructions developed to control the functional behaviour of an embedded system. Within the DAS context, the major function of the firmware is to acquire, pre-process (i.e. format data) and stream data to the host system. The DAS controller is a reactive system, which means it must respond to the event and react timeously. These events shall be generated from the user via the UI running on the host system. It shall perform all required functions in response to controller events request. From a firmware perspective, the DAS controller (reactive system) can be divided into three main modules (reactive subsystem):

- HW Initialiser
- Main Application
- Failure Recovery

HW Initialiser

This module will consist of all relevant functions required to initialise the embedded system hardware. Therefore, its main job is to initialise all the required embedded system hardware resources (e.g. ADC, I/Os, UART, DMA, timer). This may include local and global variables initialisation and also setting up interrupts and allocating buffer memory. This module functions may only need to run during system power-up or system reset or exceptions.

Main Application

The main application module will contain all the functions required to control, acquire, format and communicate data obtained from sensors. It will run after the HW initialising module.

The data controlling and acquisition module shall include the following routines or functions:

- Data Controller
- Data Formatting
- Data Communication

The functions data flow diagram is shown in Figure 5.35.

The Data Controller shall include the following algorithms:

- Data Sampling
- Data Monitoring

Data sampling controls the data sampling of the raw analogue signal by setting up the ADC control register with the required sampling rate and selecting ADC channels, initiating or stopping the conversion process. After each successful conversion, it will stream data containing the results of the conversion process. This is a low-level routine used to drive the ADC hardware and access registers containing conversion results. Data monitoring routine monitors the ADC status registers signals (e.g. end conversion, data ready, and errors) and shall report back any failures. This is illustrated in Figure 5.35.

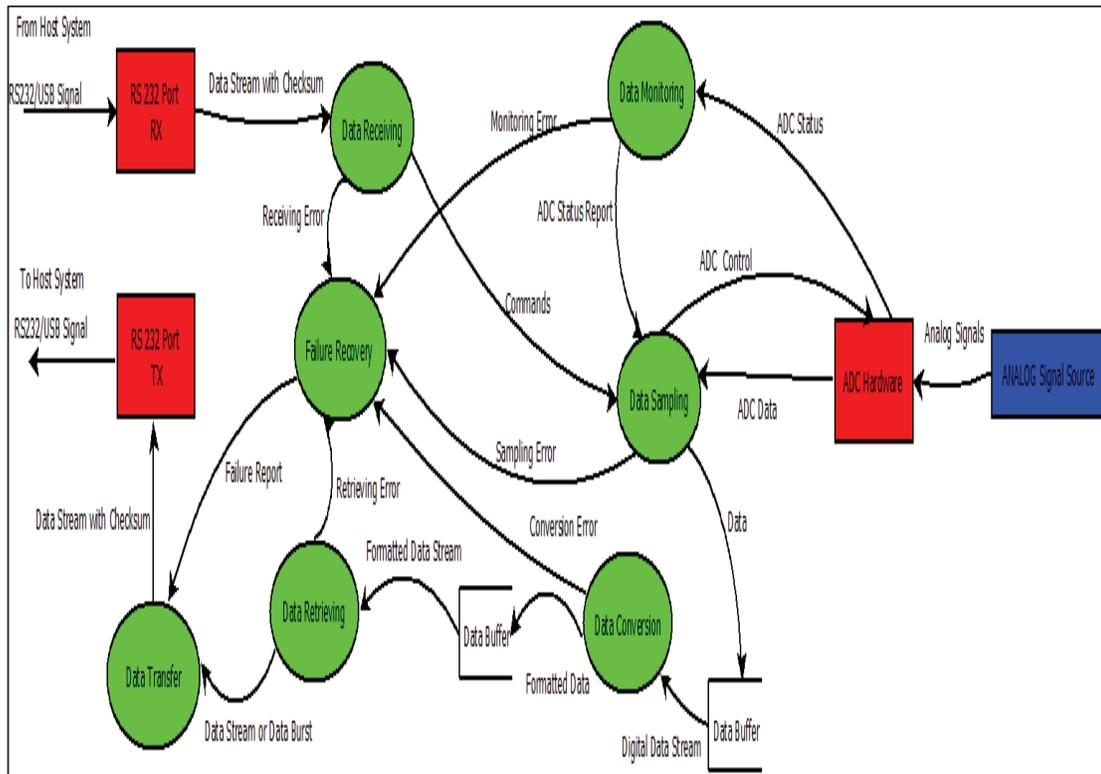


Figure 5.35 Data acquisition system firmware data flow diagram

The Data formatting function shall include following algorithms:

- Data Buffering
- Data Conversion

Data buffering routine reads and buffers the digital data stream after each successful ADC conversion. The data conversion routine transforms the raw digital data obtained from the buffer to a format appropriate for transmission. This may include transforming 10-bit or 12-bit data stream to a 16 or 32-bit data containing extra information, such as channel number and checksum. This shall include buffering the formatted data (Figure 5.35). The data communication module shall include algorithms performing:

- Data Retrieving

- Data Transferring
- Data Receiving

The data retrieving routine retrieves data from the data conversion routine after a specific time period. This process shall provide a continuous stream of data and also have the ability to support data burst. The data transferring routine shall obtain data from the data retrieving routine, generate a checksum and append the checksum to the end of the data stream. This data is sent to a serial transmitting peripheral (i.e. UART, USB) buffer (Figure 5.35). The hardware performs all the required protocol conversion (RS232) and the transmission of data. This routine shall also interact with hardware to monitor any errors during transmission and report back. It is a low-level routine required to interact with the serial communication hardware.

The data receiving routine shall read data received by the serial peripheral receiver buffer. The hardware performs the required interpretation of the serial interface protocol and fills the receiver buffer with data from the host system. The data will contain commands used to generate events required by other system modules to perform specific actions (Figure 5.35). It is also a low-level routine required to interact with serial communication hardware.

Both the data transferring and receiving routine's maximum data rate is limited by the peripheral hardware characteristics (i.e. electrical), thus setting the timing requirements for the routines used to perform the data transferring and receiving tasks. The amount of data that the system must handle is expressed as (Curtis, 2006):

$$D_{Ratemaximum} = \frac{\mathit{Baudrate}}{\mathit{Dataprotocol}} \quad (5.19)$$

The *Baudrate* is the system serial peripheral configurable parameter. The *Dataprotocol* parameter is the protocol used for transmission (i.e. RS232). For a RS232 protocol, the parameter is equal to 10 bits ($\mathit{Dataprotocol} = 8 \text{ bit data} + 1 \text{ start bit} + 1 \text{ stop bit}$). Therefore, equation (5.19) sets the maximum amount (peak rate) of data that can be sent

between the DAS and the host system per second. The peak rate includes all the system delays due to hardware limitations. Furthermore, in a data acquisition system, all modules and their routines are trigger subsystems (Gao, 2000). They are triggered by external events to launch a specific task within the subsystems. They will only execute tasks in response to the external event. In this case, the host system is the master, and it will provide the external stimulus required by the data acquisition system firmware. Therefore, the DAS shall not perform any operation, unless commanded by the host system.

Failure Recovery

The module will provide error reporting and system recovery functions. The routines shall log the error and abort any operation the data acquisition system must perform. The DAS system will communicate this error to the host system (master) (Figure 5.35).

5.4.2 Host System Platform

This platform provides the hardware and software resources required to implement the user interface (UI) to control the acquisition, storing and displaying of data.

5.4.2.1 Host Hardware Requirements

The limitation of the PC-based platform is due to the specification of the motherboard and CPU. The choice of the PC/laptop is a trade-off between the cost and the required system performance. The data rate required by the system might require high-speed serial peripherals, with high serial communication rates. This is usually a bottleneck with serial communications. The other limiting factor might be the amount of available disk space required to store the acquired data over a specific time period, especially continuous data stream at a high rate.

In this project, the PC/laptop host system shall have the following minimum hardware requirements:

- 64-bit Intel processor (@ 1.5GHz or better)

- 4GB RAM
- 500GB SATA Hard Drive
- Windows 8 Operating System
- 3 x USB 3 or 2 Serial Ports
- LCD Screen
- Keyboard

5.4.2.2 Host Software Requirements

The software performs the system configuration, control, storing and post-processing functions for displaying raw data. The system configuration software governs the DAS functionality and data acquisition process. The host software is decomposed into the following two modules:

- Data Processing Module
- Data Displaying and Storage Module

The data flow diagram is shown in Figure 5.36.

The Data processing modules shall consist of two routines:

- Data Transmitting and Receiving
- Data Buffering

The Data Transmitting and Receiving routine's tasks are to receive raw data from the embedded system platform and to transmit control and configuration data to the embedded system platform. These routines have access to the serial port registers for data receiving and transmission. The data buffering routine's main task is to buffer the raw data received from the slave system. The host system hardware will perform all the required protocol conversion (RS232) and the transmission of data. The data rate of the host system is the same as the embedded system and is set by equation (5.19).

As already highlighted, the host and the embedded system have a master-slave relationship. The host system shall always be the initiator of communication between the two systems. This is performed via graphical user interface components used to handle high-level tasks (Gao, 2000). These tasks include graphical displaying of raw data, control and configuration interface. The configuration and control parameters shall be provided by the user via an interactive graphical user interface (GUI) developed specifically for this purpose. These tasks shall be performed by routines contained within the data displaying and storage modules. The data displaying and storage modules shall contain the following routines:

- Data Converter
- Displaying
- Data Storage

Data Converter

The data conversion routine transforms the raw digital data obtained from the buffer to a format appropriate for displaying. This will include transforming the data stream and performing data processing. The data processing shall convert raw data to impropriated engineering units allocated to the chosen channel. This processed data may be streamed to a buffer.

Displaying Routine

This routine shall provide the required interface between the user and the system. The user will be able to invoke events to control and configure the DAS system. The event shall contain a command that controls the acquiring of data from the DAS. These events shall be invoked from graphical elements such as forms, buttons, checkbox and edit box developed using GUI development environment such as MATLAB® (Gupta et al., 2009) or any appropriate application.

Furthermore, this routine shall plot the data on a graphical element to display data in graph form. This data may be plotted against time and representing the acquired signal over some time period. The data shall be obtained from the data conversion routine, already converted to engineering units assigned to the chosen channel. As indicated, the user will have the ability to configure the DAS, hence enabling the user to choose which channel data must be acquired.

Data Storing

The routine will save the data obtained from the data conversion routine. The data will be saved automatically with an appropriate format such as binary, matfile (MATLAB® file format), CSV, text and ASCII. It is recommended to store data in CSV format, as this will make it easier to import data into Excel and any other data analysis applications.

In this project, no data analysis routines will be developed, as it is beyond the scope of this project.

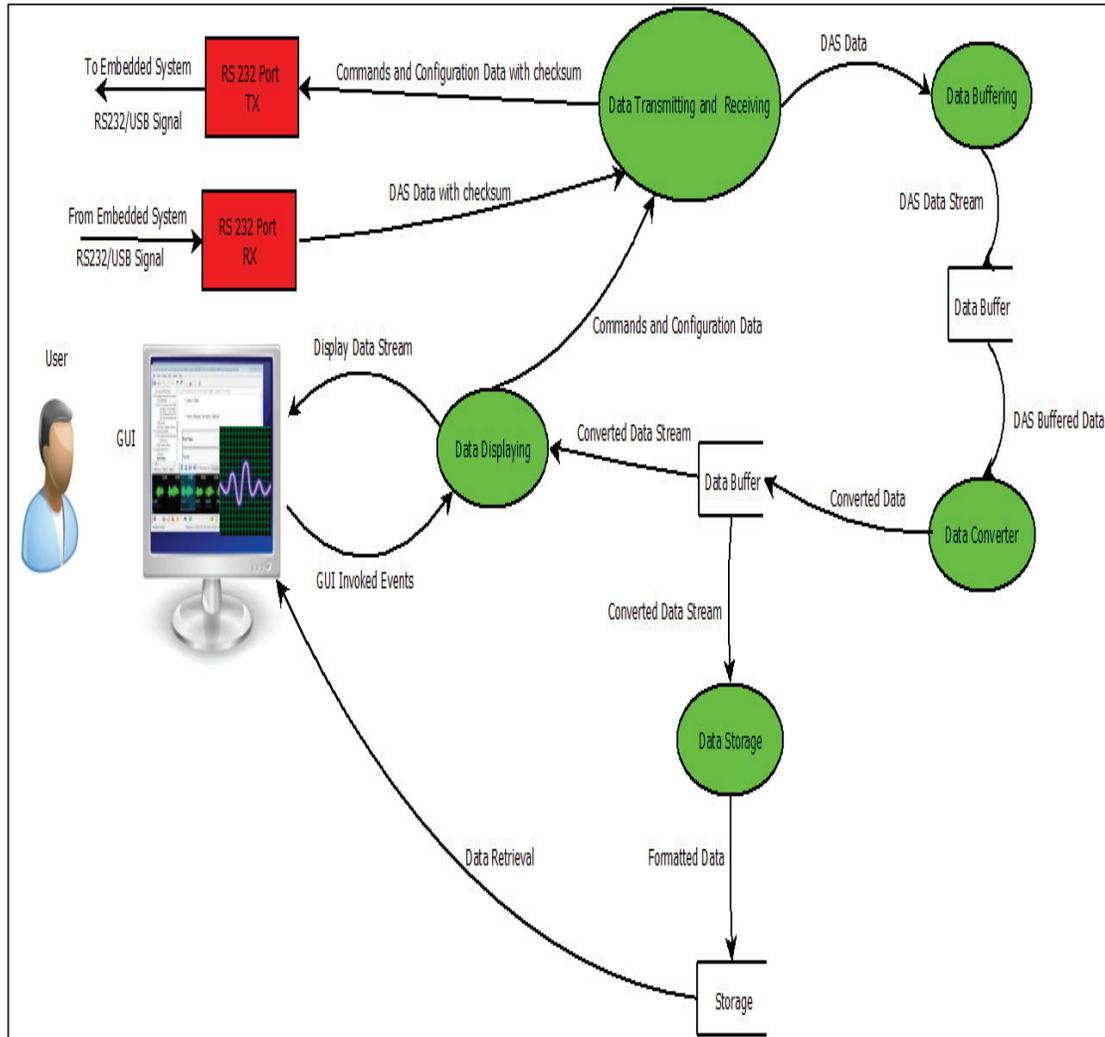


Figure 5.36 Host platform software data flow diagram

Chapter 6 DAS Hardware, Firmware and Software Implementation

6.1 Introduction

The preceding chapter gave attention to DAS hardware, firmware and software. The implementation of the DAS hardware, firmware and software is documented in this chapter. The implementation is based on the design requirements.

6.2 Signal Conditioning Card

The signal conditioning card performs one of the most crucial tasks in a DAS system. It pre-processes the raw analogue signal before it is fed to the ADC stage. The pre-processing includes the following tasks:

- Current-to-Voltage conversion (for 4-20mA signals)
- Signal scaling
- Filtering

6.2.1 5kHz Channel Implementation

The 14 channels are partitioned in two as follows: seven channels for converting 4-20mA analogue signals to voltage and other seven channels for reading 0-10V analogue signals. The 4-20mA channel input stage consists of an input resistor implemented with precision 249 Ω chip resistors (0805 size). The high-precision resistors have good performance parameters (Electronics, 2004). It has a resistance tolerance of $\pm 0.1\%$ and temperature coefficient of $\pm 10\text{ppm}/^\circ\text{C}$. The resistor was selected due to the high cost of the 250 Ω chip resistors with better parameters.

The buffer, input, scaling and filtering stages were implemented with the TL084BC op-amp. The IC consists of four op-amps within a 14-pin IC package. This will reduce the number of components required in implementing a channel. A single 5kHz channel will be made up with

one TL084BC IC, as illustrated in Appendix W. The op-amp has good overall DC and AC performance (Instruments, 2014).

The resistors used in the implementation of the scaling and filter stage were also high-precision components and the capacitors have good tolerances of less than $\pm 25\%$. The maximum resistor value used in the implementation of the voltage scaling and signal filtering stages was limited to 10k Ω . This will aid in reducing noise contribution due to component parameters.

6.2.2 50kHz Channel Implementation

The 50kHz channels were apportioned into two, with one channel for 4-20mA signals and the latter for 0-10V signals. The only difference between the 5kHz channel implementation and 50kHz channel implementation is the op-amp IC. The 50kHz channel is implemented with the THS4032 100MHz low-noise high-speed op-amp (Instruments, 2010), which has very good AC performance. It consists of two op-amps within an 8-pin IC package. Therefore, a single 50kHz channel was implemented with four THS4032 IC; this is illustrated in Appendix X.

6.2.3 PSU Implementation

For the signal conditioning circuitry to operate, it will require an external power source (+24V). Both the TL084BC and THS4032 op-amps require a dual (split) power supply of +12 and -12V to give good operating performance. Generally, dual supply op-amps have good performances compared to single supply op-amps. To reduce noise attributed to the external power supply, a combination of isolated DC-DC converter and a linear low-dropout (LDO) regulator was used to generate the +12V and -12V split supply. The DC-DC converter steps down the 24V input supply to generate the split voltages of +15V and -15V and also provide galvanic isolation. This means the input return ground is not electrically connected to the return ground of the +15V and -15V output voltages. The outputs of the DC-DC converter is then fed to a positive LDO regulator to generate a +12V and a negative LDO regulator to generate -12V. This helps in the reduction of noise generated by a DC-DC converter. The DC-DC converter was implemented using the THL 25-2423WI series DC-DC converter (Power,

2014). The PSU schematic is shown in Appendix Y.

6.2.4 Noise Decoupling

The decoupling capacitors are implemented on the power bus to reduce power supply noise and any other noise (i.e. EMI and RFI (Jung, 2005)) picked by the power bus. The bypass capacitors are implemented on both positive and negative power bus connecting to the power pins on the op-amp. A parallel combination of four capacitors was used to implement a noise decoupling capacitor bank. The values used were 1nF, 10nF, 100nF and 10uF. This provided a wider noise-bypassing capability. The capacitor bank is illustrated in Figure 6.1 and was implemented on each op-amp power pin.

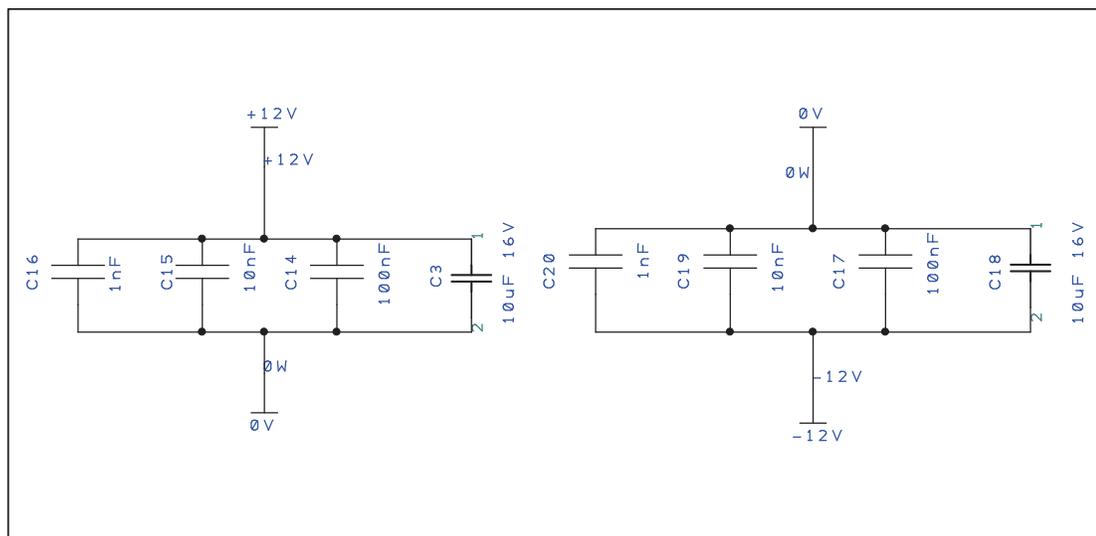


Figure 6.1 Bypassing capacitors

6.2.5 PCB Implementation

The signal condition PCB is implemented with a four-layer stack layout. The layers perform different roles. The top and bottom layers are used for component footprints, signal and supply traces routing. The inner layers are used for the two ground planes: power (0V) and signal (Vss) return planes. This is shown in Figure 6.2. The layout will provide the required separation between return grounds for the power and analogue signals. It is a good PCB

layout practice for reducing noise coupling between analogue and power signals.

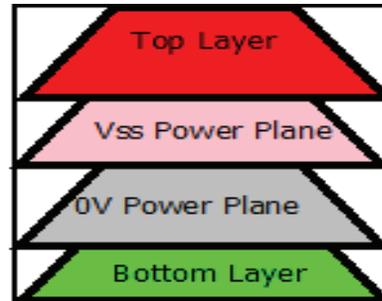


Figure 6.2 PCB layer stack

The one plane is connected to the power return ground (0V) and the latter to the signal return ground (Vss). The two return grounds are only connected at one point on the PCB, as illustrated in Figure 6.3. This will provide different return paths for power and analogue signals and aid in reducing noise on the analogue signals.

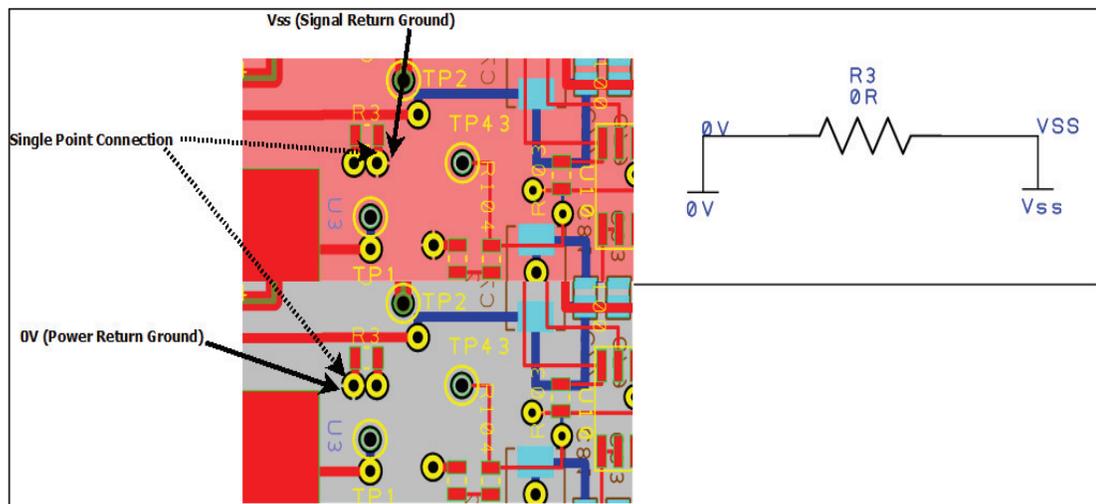


Figure 6.3 Grounding scheme

The bypass capacitors were placed on both positive and negative power bus, and near to the power pins on the op-amp IC. The interface to the sensor and embedded system was implemented with general purpose connectors and headers. The schematic diagram for the

external interface is included in Appendix A. The 3D view (top and bottom) of the PCB are illustrated in Appendix B. It shows provisions (headers) made to interface with different embedded system development boards. The bare signal conditioning PCB is illustrated in Appendix C.

6.3 Analogue-to-Digital Converter (ADC) and Embedded System

The ADC and embedded system is implemented using Arduino® Due hardware platform. The firmware required to control all the function of the data acquisition process, pre-processing and communication of data is developed and implemented in a MATLAB® Simulink® model-based development environment.

6.4 Controller Firmware

The controller firmware is modelled in Simulink® and represents the operation of the DAS controller. This model will be simulated, refined, tuned, tested and compiled and loaded to the Arduino® platform for verification. The high-level model depicts the system algorithm. The model is based on the system requirements, discussed in the design phase. The model is implemented using a common block set from Simulink® library²¹ and dedicated block set library for Arduino® Due platform²². The Arduino® support package library is specifically developed to implement functions such as I/O control and set-up, serial communication control and set-up, and ADC control and set-up for the Arduino® Due hardware platform; these are the software drivers. The Simulink® Arduino® support package common library set used to implement the system is shown in Appendix H.

The one main disadvantage encountered with model-based code development on the Arduino® Due platform is that one loses the low-level implementation details (i.e. operating speed). This means that the programmer does not have control over how the final code is implemented by the model compiler. Therefore, parameters such as timing and code

²¹ Simulink® User Guide R2014b

²² <http://www.mathworks.com/hardware-support/arduino-simulink.html>

efficiency are hidden from the programmer. It was also difficult in mapping the sampling time set during simulation with real-time requirements of the DAS controller. The current version of the Arduino[®] block library is also limited, as it does not provide software drivers for accessing and controlling useful peripherals of the powerful ATSAM3X8EA-AU²³ 32-bit ARM processor such as timers, counters, and real-time clock. The DAS controller firmware implementation (model) is shown in Figure 3.5, and the Data Acquisition implementation is shown in Figure 3.8. Data acquisition subsystem contains the ADC Channels, Relay Data, Data Formatting and Data Transmission subsystems implementations.

6.4.1 ADC Channels Subsystem

The ADC Channels subsystem task is to read the analogue inputs and pack sampled data. It is made of 12 Analogue input blocks which measure the signal connected to channel A0 to A11 on the Arduino[®] Due hardware platform and the vector concatenate block (MathWorks, 2014a). The ADC Channels subsystem model is presented in Figure 3.9. The vector concatenate block packs the measured data into a 12 x 1 vector matrix (contiguous locations in memory).

The Arduino[®] library block only allows a default setting of ADC bits resolution, which is set at 10 bits and the voltage reference, which is set at 3.3V. This is a limitation, as the ADC hardware on the ATSAM3X8EA-AU processor can be configured to have a 12 bits resolution (at the expense of sampling speed) and the reference voltage set at 2.56V via internal voltage regulator (i.e. gaining resolution). However, in this project, the default settings are adequate for the implementation of the ADC to perform the data acquisition process. This sets the ADC resolution at 3.2227mV/bit.

The analogue input subsystem model is shown in Figure 6.4. The analogue block reads the signal and converts it to a 16-bit unsigned integer (uint16). The output of the 10-bit ADC ranges from 0-1023, which is saved into a 2-byte variable of type uint16. The data is then

²³ <http://www.atmel.com/AT91SAM>

converted to double, to allow data compatibility for Simulink® simulations. The analogue input subsystem also has an enable input, which allows individual channel enabling and disabling. The enable input status is passed on as an output to indicate the status of the analogue input subsystem (Figure 6.4).

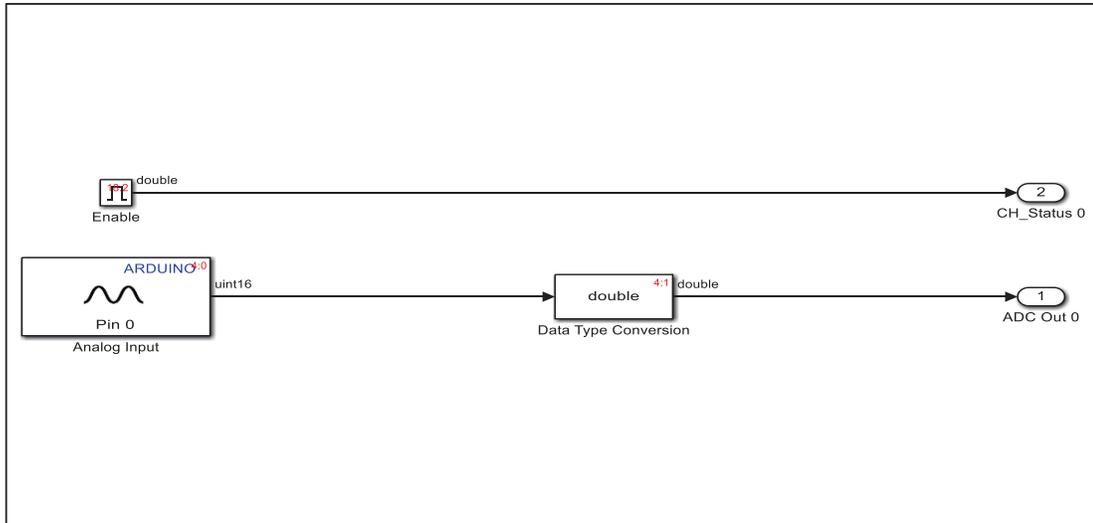


Figure 6.4 Analogue input subsystem

The analogue library block²⁴ is used to select the channel and configure the sampling time. This allows each analogue input block to be set at different sampling rates. The minimal sampling time (or maximum sampling frequency) is limited at $1\mu\text{s}$ (or 1MHz). This corresponds with the ADC hardware maximum sampling frequency limit of the ATSAM3X8EA-AU processor. In this implementation channel, 1 to 10 sampling time was set to $50\mu\text{s}$, which sets the sampling frequency at 20kHz (20 000 samples per second). The channels should be able to measure a signal with bandwidth of 5kHz. Also, for channel 11 to 12, the sampling time was set to $10\mu\text{s}$, putting the sampling frequency at 100kHz (100 000 samples per second). This will limit the maximum measured signal bandwidth to approximately 41kHz.

²⁴ <https://www.mathworks.com/help/supportpkg/arduino/ug/analoginput.html>

6.4.2 Relay Data Subsystem

This subsystem is a block made of MATLAB® Function²⁵ which contains the user code. The block function is to relay data obtained from the ADC channels subsystem. Only the data of the enabled analogue input subsystem gets relayed to the Data Formatting subsystem. The output of the subsystem (Data_Channel array) will always contain 12 elements with data or zeros, depending on the channels selected. The MATLAB® embeddable C code is provided in Appendix I.

6.4.3 Data Formatting Subsystem

The Data Formatting subsystem model is shown in Figure 6.5. The main function of this subsystem is to format data into data packets that are sent through the serial port. The analogue data type is first converted from double to uint16, using the data type conversion block. The data is then passed through the Convert subsystem, which divides the uint16 data into two bytes without losing the ADC conversion results. The Convert subsystem is a MATLAB® Function²⁶ which contains the conversion code. The code is shown below:

```
function y = Convert(u)
%convert 16 bit to 2 bytes
Inarray = u;
Intarray = typecast(Inarray, 'uint8');
y = Intarray; %return two bytes of 8 bit wide upper byte D2 and lower byte
D1
```

The `typecast`²⁷ function divides the data contained in variable **u** into two bytes segmented without changing the value of the contents. The value is preserved within the two bytes segment. The data conversion is required because the output of the ADC channel is 10 bits wide and also because data can only be transmitted one byte (8 bits) at a time through the serial port.

²⁵ <http://www.mathworks.com/help/simulink/slref/matlabfunction.html>

²⁶ <http://www.mathworks.com/help/simulink/slref/matlabfunction.html>

²⁷ <http://www.mathworks.com/help/matlab/ref/typecast.html?searchHighlight=typecast>

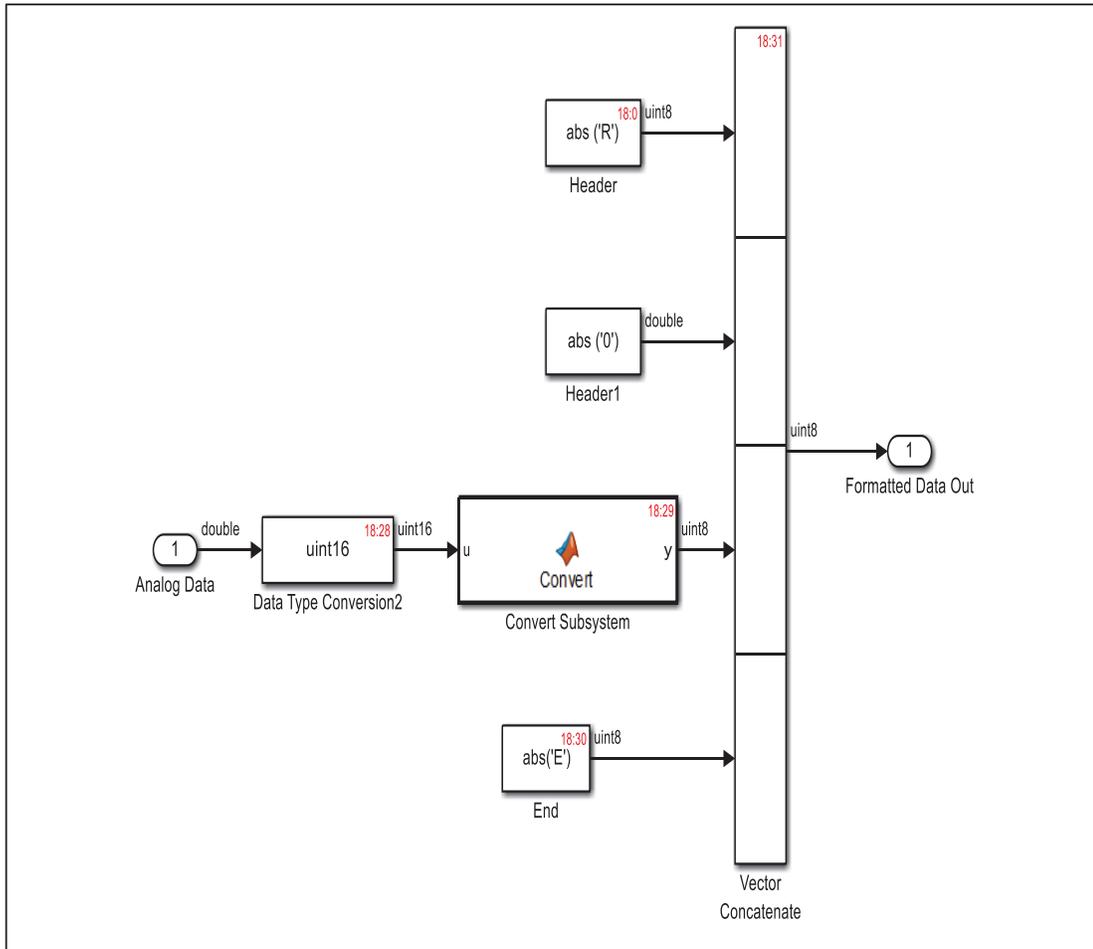


Figure 6.5 Data formatting subsystem

The data is packed according to the message structure shown in Figure 6.6. The message starts with character “R”, indicating start of message (reply). It is followed by a character “0”. This is then followed by 24 bytes, containing ADC channel data.

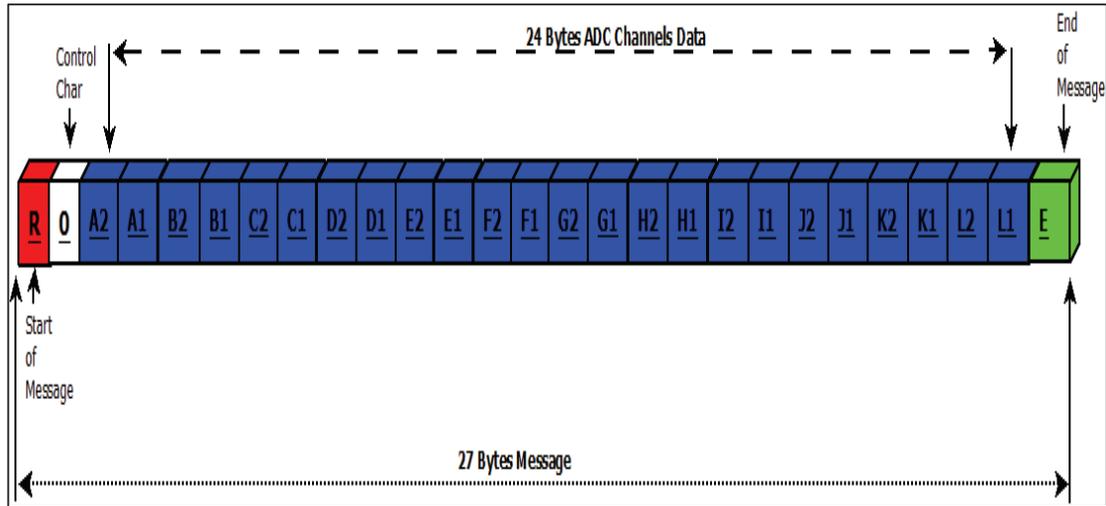


Figure 6.6 Data message structure

The upper byte is contained in location A2, and the lower byte is in location A1 for channel 1. For channel 2, the upper byte is contained in location B2 and the lower byte in location B1. The channel data is saved in this way up to location L1 and L2, which contain channel 12 lower and upper bytes respectively. Lastly, the message structure ends with character “E”, indicating end of message. The data packing is achieved through the vector concatenate block, which packs data into contiguous locations in memory. Therefore, the output of the Data Formatting subsystem will be a 27 x 1 vector matrix.

6.4.4 Data Transmission Subsystem

The data transmission model consists of only the serial transmit block which handles all the buffering and data transmission. The model is illustrated in Figure 6.7, and the physical interface is shown in Figure 3.7.

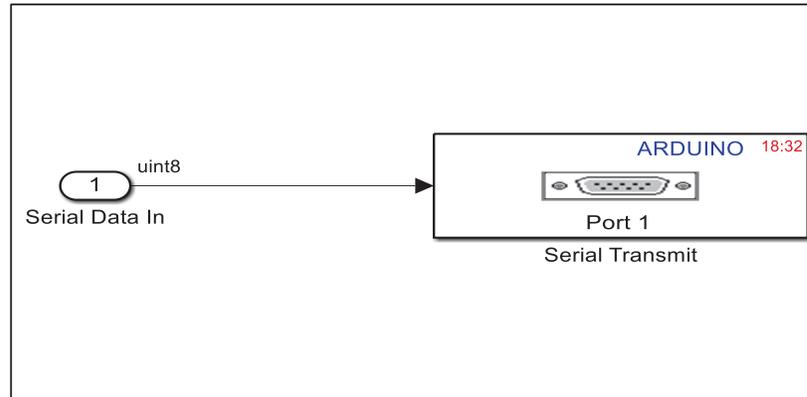


Figure 6.7 Data transmission subsystem

The block only accepts uint8 (unsigned 8-bit integer) data type. The only parameter setting required on the serial transmit block is to set the port number. The port number was set to 1, and this selects port 1 as the transmitting port. Other configuration settings required for successful communication via serial port are done through the model configuration parameter dialog box (shown in Figure 6.8). The serial transmit block model baud rate can be set to a maximum of 500 000.

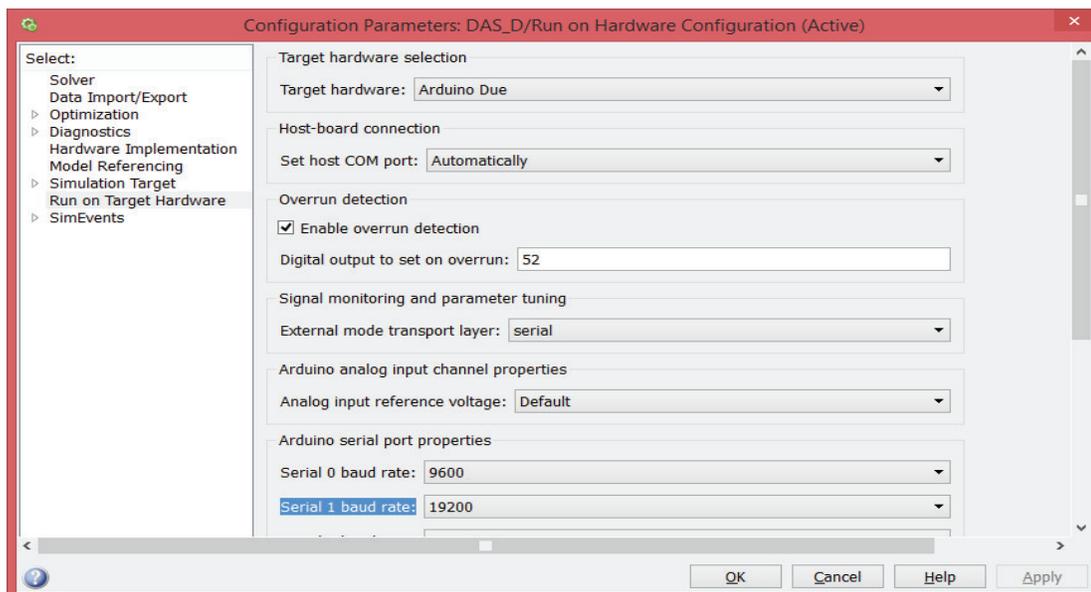


Figure 6.8 Serial port configuration

6.4.5 Control Subsystem

The model is implemented with an Arduino® serial receiver block, a Command processing and Control signal subsystems (Figure 3.12). The serial receiver block provides the buffering and reading of the serial data obtained from the host system. The block reads data from the serial port one byte at a time and outputs a type uint8 data. The block was set to receive data on port1 and the sampling time set at 5ms. This time set the rate the block reads the serial port buffer. In this case, it is 200 times per second. The baud rate setting is the same as the serial port transmit block. The block provides a status output signal, which goes high when a byte is available in the receiver buffer. These outputs are then fed to the Command processing subsystem. The Command processing subsystem contains the MATLAB® function block, with a code to read and store received data into a 20 x 1 matrix. The output of this block is fed to the Control signal subsystem. The function block code that performs command processing is listed next.

```
function Commands = Text_Command(Data, DataReady)
%#codegen
Max_Data_Storage = 20; %set the size of the matrix
Max_Count = 18; %set the maximum data to read from receiver (two
extra bytes read to clear the buffer)
persistent Txt count; %create memory to store data which retains its
contents on call of this function.
if isempty(Txt,count) %initialize variables
    Txt = zeros(Max_Data_Storage,1);
    count = 1;
end

%Read data from the receive buffer
if DataReady == 1 %read data
    Txt(count)= Data; %store data at location index set by count
    count = count + 1; %increment count to point to next memory location
end

if count == Max_Count %reset count (index) if expected maximum number of
bytes are received
    count = 1;
end
Commands = uint8(Txt); %return 20x1 matrix containing received bytes
```

The command message structure used to start the data acquisition process is shown in Figure 6.9. The message is sent by the host system to start the data acquisition process.

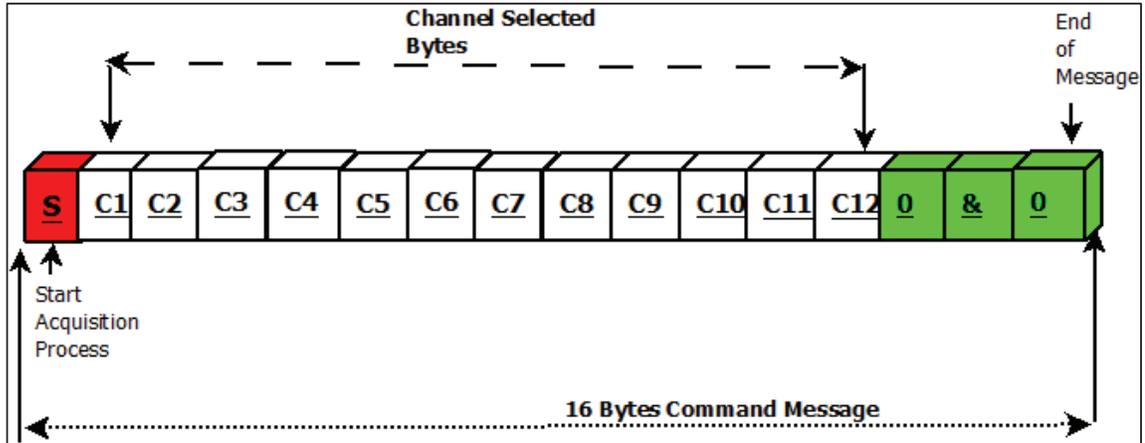


Figure 6.9 Start of data acquisition process command message structure

The message starts with a character “S” interpreted by the Command processing subsystem, as an instruction to start the data acquisition process. The next character “Cx” represent the channel selected for data acquisition. The information that character “Cx” can represent and its interpretation is shown in Table 6.1. If “Cx” value is 1 or Z, then channels are selected, otherwise, no channels are selected. The next three characters “0”, “&” and “0” are control or filler characters.

Table 6.1 Character “Cx” interpretation

| Cx | “Cx” Data | Interpretation |
|----|-----------|---------------------|
| 1 | “1” | Channel 1 Selected |
| 2 | “1” | Channel 2 Selected |
| 3 | “1” | Channel 3 Selected |
| 4 | “1” | Channel 4 Selected |
| 5 | “1” | Channel 5 Selected |
| 6 | “1” | Channel 6 Selected |
| 7 | “1” | Channel 7 Selected |
| 8 | “1” | Channel 8 Selected |
| 9 | “1” | Channel 9 Selected |
| 10 | “1” | Channel 10 Selected |

| Cx | “Cx” Data | Interpretation |
|-----|-----------|-----------------------|
| 11 | “1” | Channel 11 Selected |
| 12 | “1” | Channel 12 Selected |
| ALL | “Z” | All Channels Selected |

The stop command message structure used to instruct the Control subsystem to stop the data acquisition process is shown in Figure 6.10. This message is generated by the host system, and it is sent to the DAS system via a serial port. The message contains two characters, namely, “X” and “+”, which are interpreted by the Control subsystem as an instruction to stop the data acquisition process. The last three characters are filler characters.

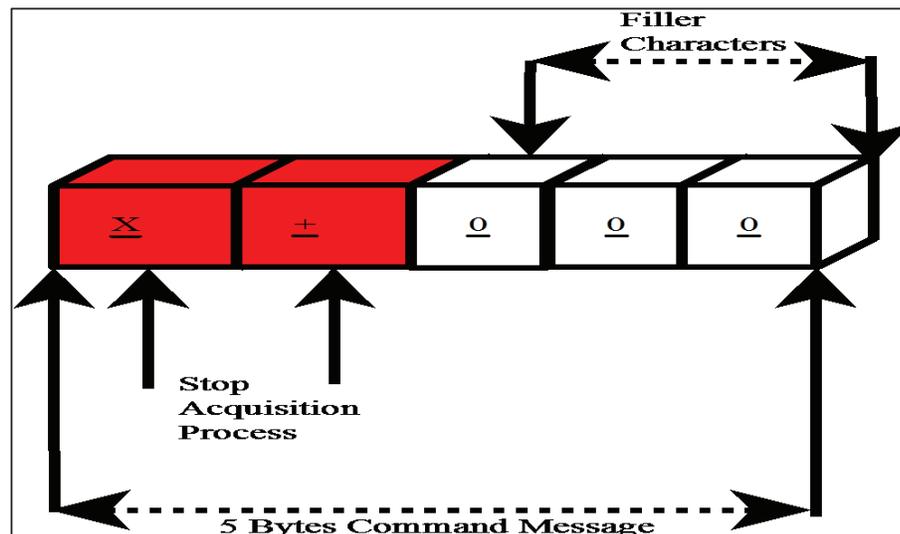


Figure 6.10 Stop of data acquisition process command message structure

The control signal function block receives the 20 x 1 matrix containing the command message. The block decodes and processes the command. It then generates the control signals required for controlling the Data Acquisition subsystem. The function block code that performs the decoding and generation of the control signals is listed in Appendix F.

6.4.6 System Alive Subsystem

This subsystem’s function is to flash LED at a rate of 20 000 times per second, with a duty

cycle of 50%. This will indicate if the system is still alive (not crashed). The subsystem model is shown in Figure 6.11. This subsystem is implemented with an Arduino® digital output²⁸ block and a Simulink® pulse generator source²⁹ block. The pulse generator is configured to generate a 20kHz pulse, with the pulse width set at 50%. The output of the pulse generator block is fed to the digital output block, which will generate the required voltage levels on the hardware platform to drive the LED. The digital output block was configured to output the signal on pin 51 of the Arduino® Due hardware platform.

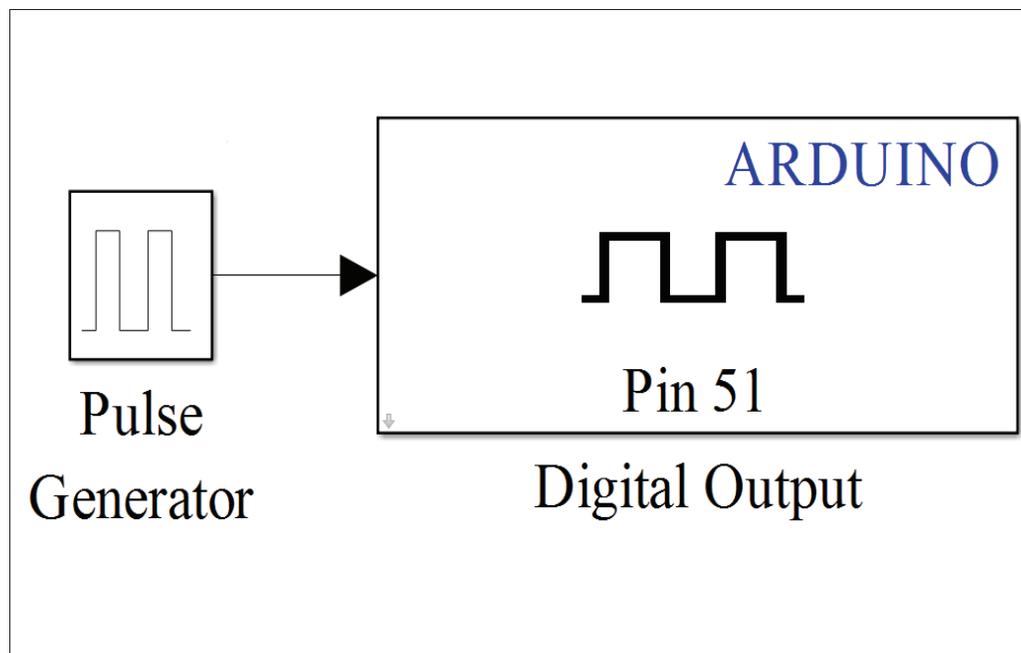


Figure 6.11 System Alive subsystem model

6.5 Host System Software

The host system code used to process the data is implemented with MATLAB® programming language. The entire host system implementation code is listed in Appendix G.

²⁸ <http://www.mathworks.com/help/supportpkg/arduino/ug/digitaloutput.html?refresh=true>

²⁹ <http://www.mathworks.com/help/simulink/slref/pulsegenerator.html?searchHighlight=Pulse%20Generator>

6.5.1 DAS Host Graphical User Interface

The host GUI contains components and a graphical display that enable a user to perform event and interactive tasks. The components implementing the DAS host user interface and layout is shown in Figure 6.12. The 13 checkboxes are grouped together on the panel object. The checkboxes are used to select the channel measurements that must be taken. The panel object is also used to group listbox components. The listbox will allow the user to select engineering units associated with the selected channel. The two button components labelled “Start” and “Stop” are used to start and stop the data acquisition process. The axes component is used to plot the captured data. The checkbox, buttons, listbox and panel objects generate events as the components are clicked or selected. These events provide a means to implement algorithms that will perform an operation on the data and generate messages to provide information to the user. In MATLAB® GUI programming environment, this event results in callback³⁰ function execution. The callback function is linked to the event, and the code is located within the callback function body. Figure 6.13 shows the DAS GUI when running in a MATLAB® GUI programming environment.

The user must select a channel or channels used to measure the analogue signal. This will result in a callback function linked to the listbox being executed. After that, the start button can be clicked to execute the callback function linked to the start button event. The main code that performs data processing, data storing, serial communications and data plotting is contained within the body of the start button callback function (see Appendix G). The code will open the serial port device (if available); thereafter, it prepares a start command message (Figure 6.9). The message is then transmitted via a serial port device to the DAS controller platform. The host code waits for a response from the DAS controller (Figure 6.6), and each channel data is then processed.

The data processing includes the concatenating of the two-byte ADC data into one uint16 data type and conversion of the data into raw measured value (equation (7.1). After processing,

³⁰ https://nf.nci.org.au/facilities/software/Matlab/techdoc/creating_guis/ch_pro16.html

the data is stored in a text file and plotted on the axes component. The data streamed by the DAS controller is stored and plotted until the stop button is selected. This will result in a callback function executing code that prepares and sends a stop command message (Figure 6.10) to the DAS controller via the serial port device. This stops the data acquisition process.

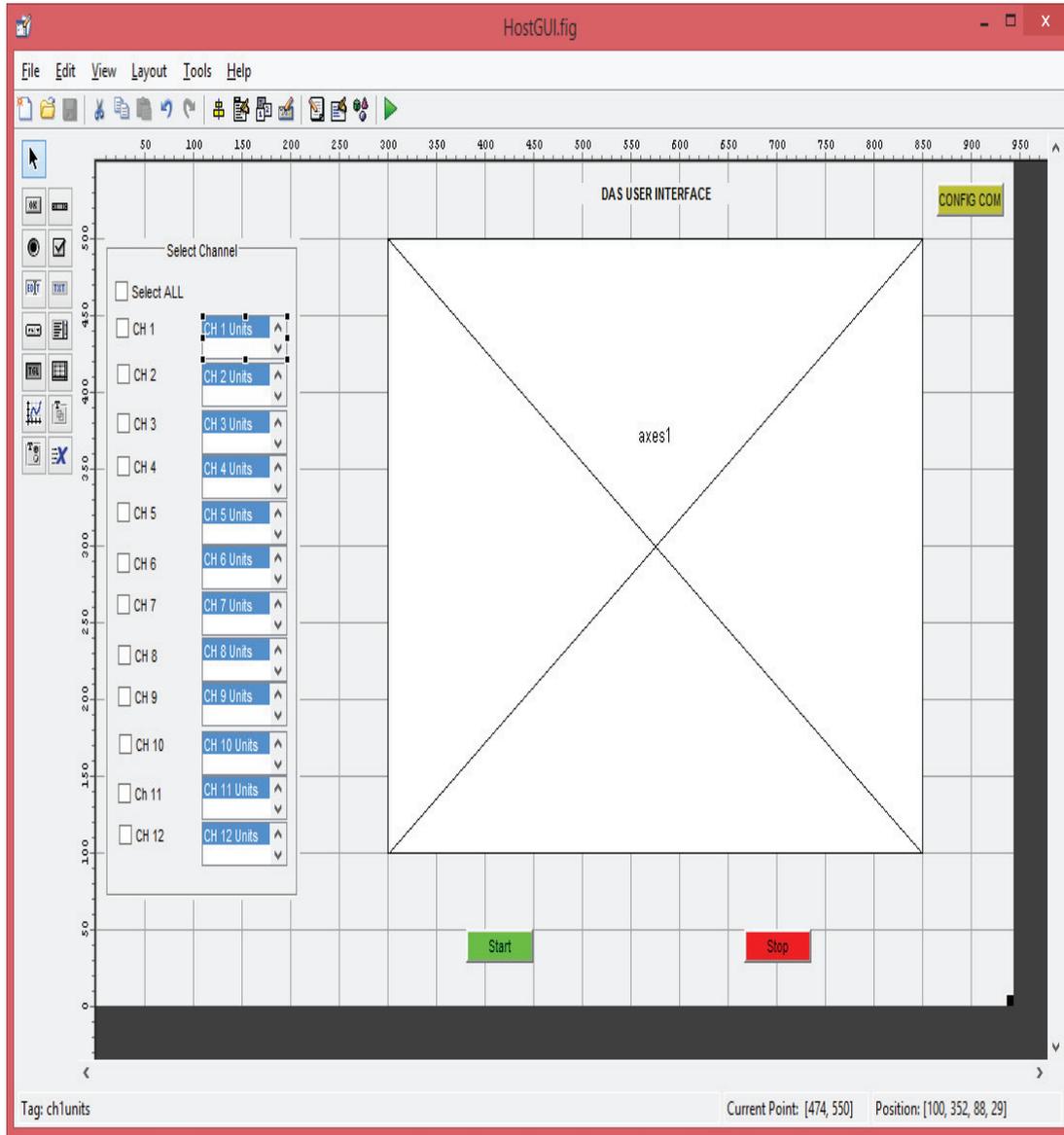


Figure 6.12 DAS user interface components layout

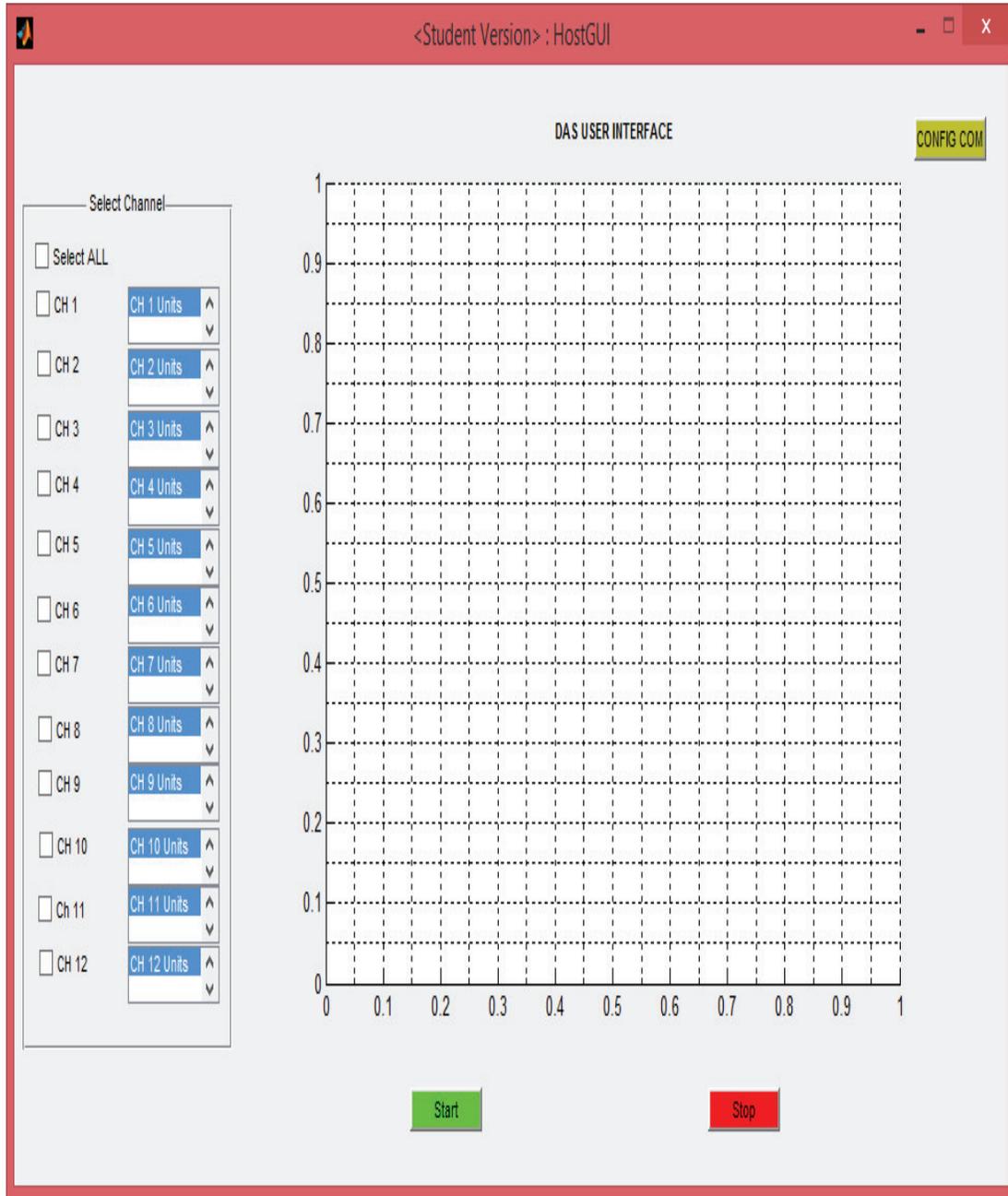


Figure 6.13 DAS user GUI running

Chapter 7 Experimental Results and Analysis

7.1 Introduction

It is crucial that the DAS system is tested and verified. The DAS system testing includes characterising and validating the signal conditioning card, ADC, controller and host subsystems. The foregoing chapter was a discussion on the DAS hardware, firmware and software implementation. In this chapter, the results used to verify each DAS subsystem are presented. The methods and experimental set-ups are detailed in Chapter 3. The signal conditioning card results are also presented. The ADC subsystem is tested and its measurements are verified by comparing them to calibrated instrument measurements and theoretical values. The DAS controller and GUI software's main functions were tested and verified. Lastly, all the subsystems are integrated together, and the entire system is calibrated. The results of each test are presented and analysed, and the limitations are presented and discussed.

7.2 Signal Conditioning Card Tests

The signal conditioning card under test is shown in Appendix D, and the set-up is illustrated in Appendix E.

7.2.1 DC Test Results

PSU Ground Isolation Results

The isolation between the power supply rails and return grounds was measured. High isolation resistance between power rails and return is good, as it indicates no shorts and unexpected low impedance paths to ground (Table 7.1).

Table 7.1 Power supply and ground isolation results

| Isolation Resistance/Impedance Requirements | Min (k Ω) | Measurement Results |
|---|----------------------|------------------------|
| +12V and Power Return Ground | 10 | >1.656M Ω |
| +12V and Signal Return Ground | 10 | >1.656M Ω |
| -12V and Power Return Ground | 10 | 25.5k Ω |
| -12V and Signal Return Ground | 10 | 25.5k Ω |

PSU Voltage Test Results

This tested the power supply voltages used to power the op-amps and bias circuits implemented on the signal conditioning card. The PSU voltages were measured with no signal applied to the inputs of the signal conditioning card.

The PSU voltage was within specification (Table 7.2).

Table 7.2 PSU voltage measurement results with no input signal

| PSU Voltages | Min (V) | Typ (V) | Max (V) | Test Condition | Measurement Results (V) |
|--------------|------------|------------|------------|-----------------|-------------------------------|
| +12V | 11.50 | 12 | 12.5 | No input signal | +12.5 |
| -12V | -11.50 | -12 | -12.5 | No input signal | -11.98 |

PSU Voltage Test with DC Input

A DC voltage was applied to the channel inputs, and the PSU voltages were measured. The +12V was slightly off specifications because of the power rail loading (Table 7.3). This did not affect the op-amp operation and performance as it is “specified” to operate up to +15V.

Table 7.3 PSU voltage measurement results with +1.5V DC input signal

| PSU Voltages | Min (V) | Typ (V) | Max (V) | Test Condition | Measurement Results (V) |
|--------------|------------|------------|------------|--|-------------------------------|
| +12V | 11.50 | 12 | 12.5 | +1.5V DC input applied to the channel inputs | +12.56 |
| -12V | -11.50 | -12 | -12.5 | +1.5V DC input applied to the inputs | -11.96 |

PSU Voltage Test with AC Input

An AC signal was applied to the channel inputs, and the PSU voltages were measured.

The +12V rail was slightly off specifications (Table 7.4).

Table 7.4 PSU voltage measurement results with +1.5V AC input signal

| PSU Voltages | Min (V) | Typ (V) | Max (V) | Test Condition | Measurement Results (V) |
|--------------|------------|------------|------------|---|-------------------------------|
| +12V | 11.50 | 12 | 12.5 | 1V sinusoidal signal applied to the channel inputs | +12.56 |
| -12V | -11.50V | -12V | -12.5 | 1V sinusoidal signal applied to the channel inputs | -11.98 |

7.2.2 Signal Chain Results

These characterise the signal chain path. The signal chain measurements were compared to the simulation and theoretical results.

DC Offset Measurement Input Grounded

The channel inputs were connected to ground (0V), and the channel outputs were measured. The offset measured on the 5kHz channels were comparable to the datasheet specifications and much larger than the simulation results (Table 7.5). The 5kHz channel's offset voltages were below the ADC resolution (3.2227mV/bit).

Table 7.5 DC offset voltages with input at 0V

| Channel | Test Condition | Channel Output Measurement Results (mV) | Notes |
|------------|----------------|---|------------------------|
| Channel 1 | Input grounded | -2.5 | 4-20mA channel (5kHz) |
| Channel 2 | Input grounded | -1.7 | 4-20mA channel (5kHz) |
| Channel 3 | Input grounded | -1.7 | 4-20mA channel (5kHz) |
| Channel 4 | Input grounded | -2.1 | 4-20mA channel (5kHz) |
| Channel 8 | Input grounded | -1.0 | 0-10V (5kHz) channel |
| Channel 9 | Input grounded | -0.2 | 0-10V(5kHz) channel |
| Channel 11 | Input grounded | -54.5 | 4-20mA channel (50kHz) |
| Channel 12 | Input grounded | -53.3 | 0-10V channel (50kHz) |

The 50kHz channel's offset voltages were higher than specified in the datasheet and also much larger than the simulation results (Table 7.5). They were high in comparison to the 5kHz channel results. The 50kHz channel's offset voltages were much more than the ADC resolution (3.2227mV/bit).

DC Offset Measurement Input Floating

The channel inputs were left open (floating), and the channel outputs were measured.

Table 7.6 Channel DC offset voltage with input open (floating)

| Channel | Test Condition | Channel Output Measurement Results | Notes |
|------------|----------------|------------------------------------|------------------------|
| Channel 1 | Input open | -2.5mV | 4-20mA channel (5kHz) |
| Channel 2 | Input open | -1.7mV | 4-20mA channel (5kHz) |
| Channel 3 | Input open | -1.6mV | 4-20mA channel (5kHz) |
| Channel 4 | Input open | -2.0mV | 4-20mA channel (5kHz) |
| Channel 8 | Input open | 169.3mV | 0-10V channel (5kHz) |
| Channel 9 | Input open | -1.277V | 0-10V channel (5kHz) |
| Channel 11 | Input open | -55.5mV | 4-20mA channel (50kHz) |
| Channel 12 | Input open | -1.383V | 0-10V channel (50kHz) |

The 4-20mA channel's (5kHz) offset voltages were similar to the results shown in Table 7.5. However, the 0-10V channel outputs were much higher than expected (Table 7.6). This was due to the fact that the input was floating and there was no path for the op-amp input bias current to flow. This resulted in imbalances in the op-amp input stage causing high output voltages, sometimes as low as -10.89V. Figure 7.1 shows the input stage of the 0-10V channels. U10a is a buffer that provides the high-input impedance, for the signal source. It operates correctly provided the signal and bias current path exist. However, when the input was left open, no path existed for the bias current coming from the op-amp +IN pin, thus causing current imbalances and resulting in high offset voltages. This was solved by putting a high-value resistor of about 1M Ω on R97. It provided the high-input impedance seen by the signal source.

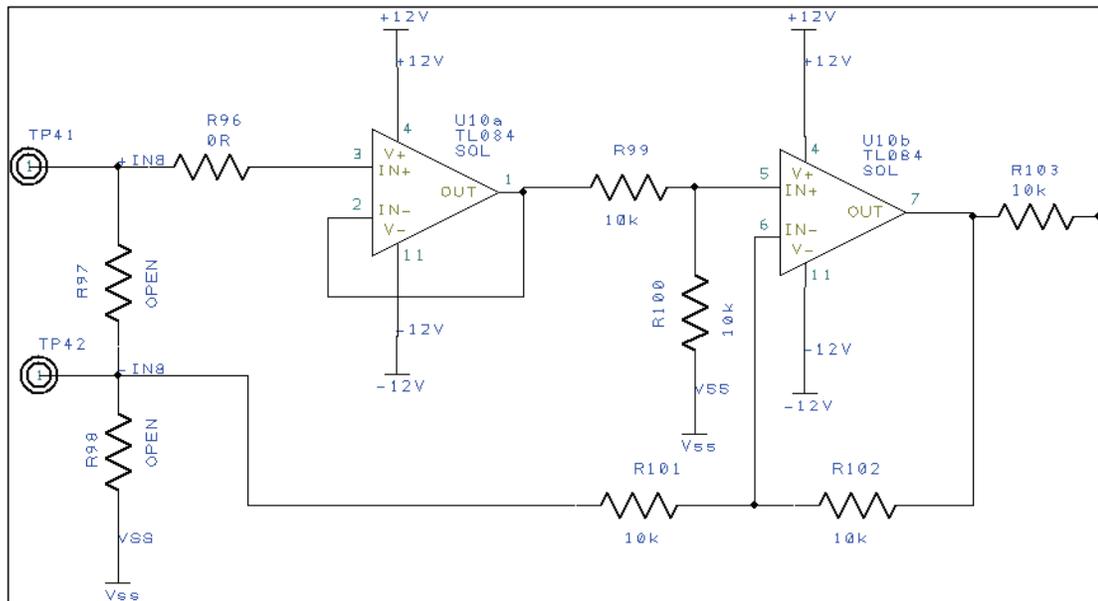


Figure 7.1 0-10V channels input stage

The same tests were repeated with 1MΩ resistors populated on the board (R97) (Figure 7.1). This resulted in channels 8 and 9 offset voltages decreasing to -0.6mV and 0.4mV respectively. For channel 12, the offset voltage was reduced to -54.7mV, which was similar to the result shown in Table 7.5.

DC Input Test

A DC voltage was applied to the channels input, and the channel outputs were measured.

Table 7.7 Signal chain DC input results

| Input Channel | Test Condition Actual measured input voltage | Channel Output Measurement Results (V) | Notes |
|---------------|--|--|--|
| Channel 1 | Input +1.604V | 0.352 | 4-20mA channels (5kHz) are scaled by 0.2194 |
| Channel 2 | Input +1.611V | 0.352 | 4-20mA channels (5kHz) are scaled by 0.2194 |
| Channel 3 | Input +1.601V | 0.353 | 4-20mA channels (5kHz) are scaled by 0.2194 |
| Channel 4 | Input +1.611V | 0.352 | 4-20mA channels (5kHz) are scaled by 0.2194 |
| Channel 8 | Input +1.617V | 0.391 | 0-10V channels (5kHz) are scaled by 0.2424 |
| Channel 9 | Input +1.615V | 0.392 | 0-10V channels (5kHz) are scaled by 0.2424 |
| Channel 11 | Input +1.507V | 0.298 | 4-20mA channels (50kHz) are scaled by 0.2194 |
| Channel 12 | Input +1.513V | 0.332 | 0-10V channels (50kHz) are scaled by 0.2424 |

The outputs of channel 1 to 4 were similar, which were comparable to the simulation results. The outputs of channels 1 to 4 were scaled by 0.2194; thus, for channel 1 with an input of 1.604V, the output was 0.352V (Table 7.7). These are acceptable results for all the 4-20mA 5kHz channels. The 0-10V 5kHz channels also gave good results. With an input of 1.617V applied to channel 8, the output was 0.391V; the input was scaled by 0.2424. Similar results were observed with all the 0-10V 5kHz channels.

However, channels 11 and 12 show slightly poorer results as compared to other channels (Table 7.7). The output voltage of channel 11 with an input of 1.507V was 0.298V. The input voltage was scaled by 0.2194. The expected output voltage was 0.330V, and the actual measured voltage was 0.298V, resulting in an error of 32.64mV. For channel 12, the input was 1.513V, and the measured output was 0.332V; the input was scaled by 0.2424. The expected output voltage was 0.367V, resulting in an error of 34.75mV. This was observed with only channels 11 and 12. The overall DC performances of all channels were similar and comparable to the design and simulation results.

AC Input Test

An AC signal with an amplitude of 250mV and set at different frequencies was applied to the channel inputs, and the channel outputs were measured. Channel measurements show no signal distortion for both 4-20mA and 0-10V channels (Figure 7.2). Similar results were observed with 50Hz, 500Hz and 5kHz signals. The signal was reduced by -3dB at 5kHz. The output of 50kHz test signal applied to the 5kHz channels is highly attenuated (Figure 7.3). The signal is attenuated by the 5kHz filter stage to about -40dB. The output was comparable to the simulations, as the filter was designed to attenuate the signal at a rate of -20dB per decade.

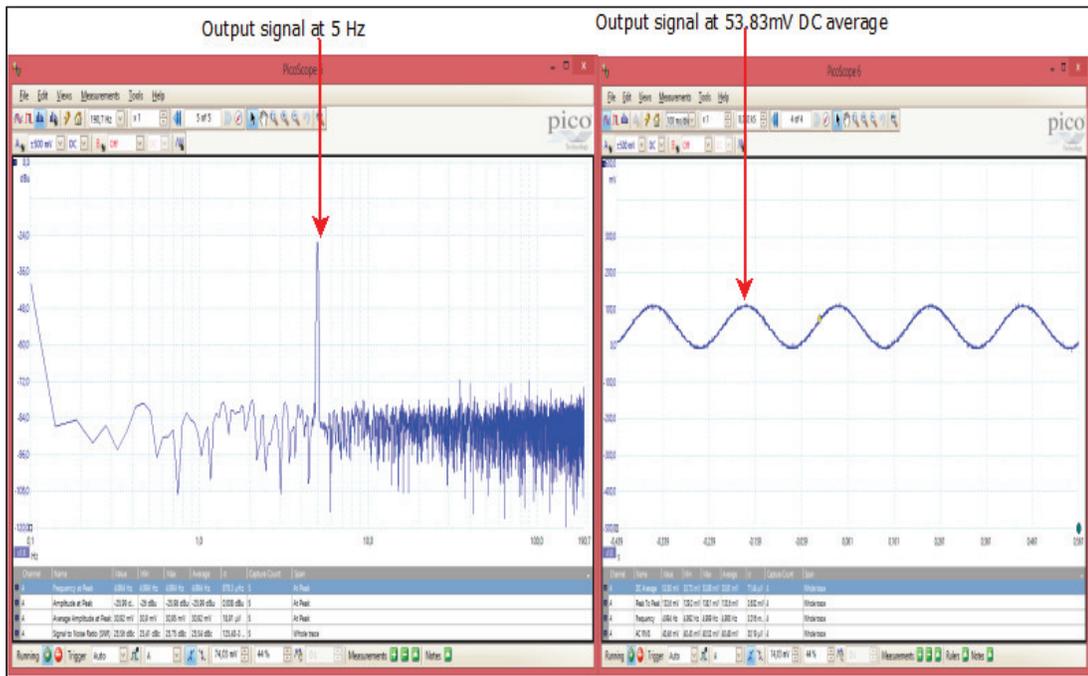


Figure 7.2 4-20mA outputs at 5Hz input signal (5kHz channel)

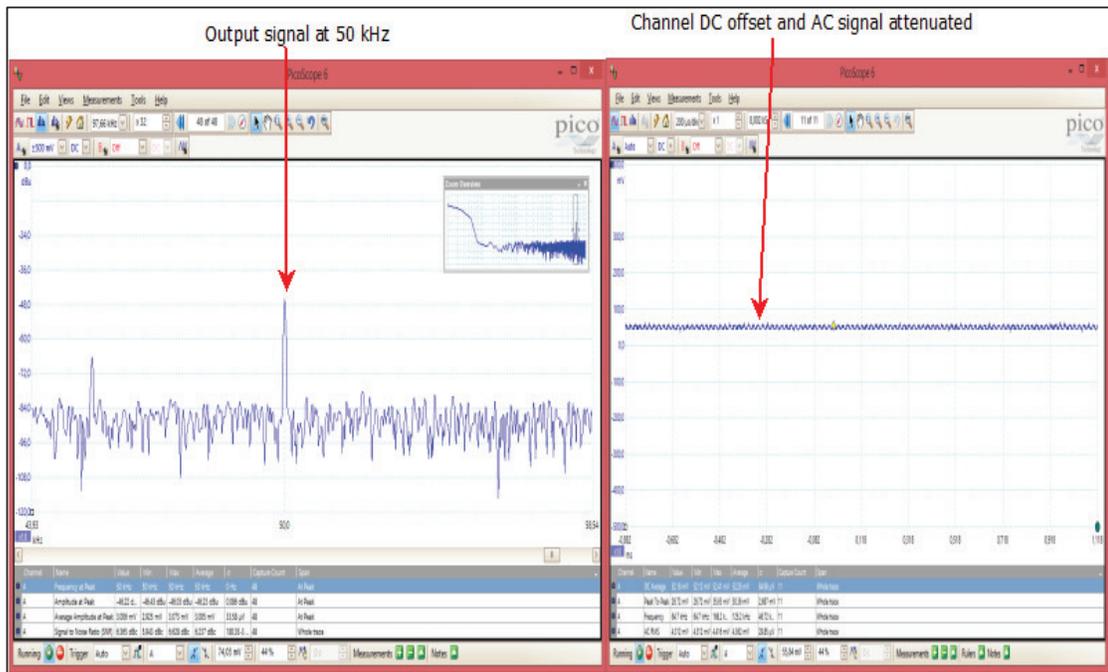


Figure 7.3 4-20mA channel output at 50kHz input signal (5kHz channel)

Similar results were observed in both the 50kHz channels. No signal distortions were observed in both the 4-20mA and 0-10V channels. The signal was attenuated by the 50kHz filter stage to -72dB at 100kHz (Figure 7.4). The filter was designed to attenuate the signal above 50kHz at a rate of -80dB per decade.

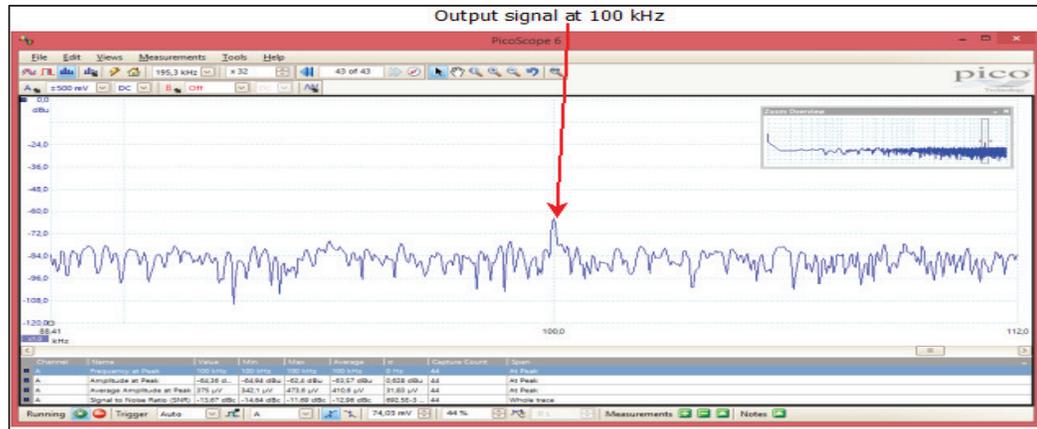


Figure 7.4 0-10V channel output at 100kHz input signal (50kHz channel)

Common Mode Input Test Results

A common mode signal was injected into the inverting and non-inverting inputs. The output voltage was zero, as no common mode signal should pass through to the output stages (Figure 7.5). The input stages reject and attenuate the common mode signal. Similar results were observed on other channels.

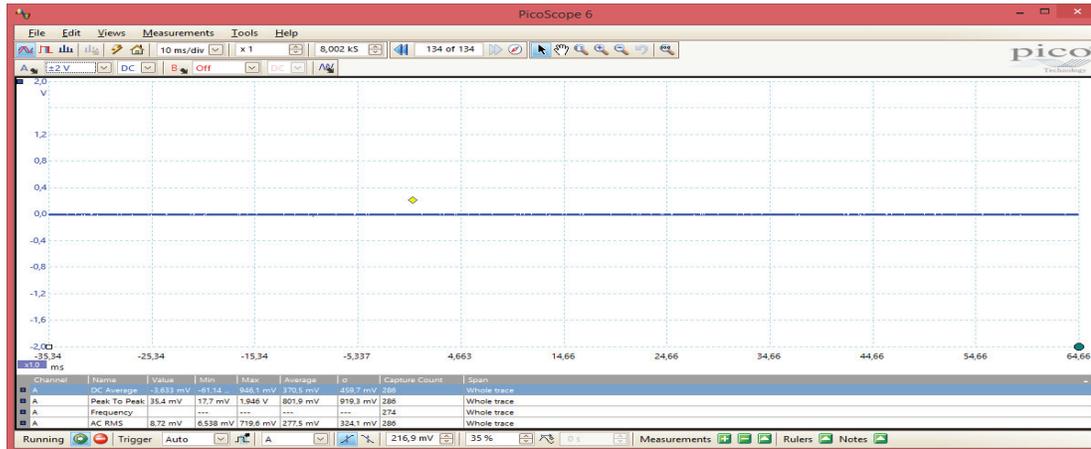


Figure 7.5 4-20mA channel output with common mode signal input

7.3 DAS Controller Test Results

The smaller step size setting resulted in the time step being too fast for the Arduino® Due hardware platform in external mode simulation (Herniter, 2014). The side effect of this was a task overlap condition. This resulted in the model not running in real time on the target platform in external mode. The model was still able to produce measurements, but at much a slower rate. This only allowed the optimisation, tuning, verifying and validation of the model in external mode. Thereafter, the model was deployed (programmed) to the target platform, and some real-time behaviour was observed. The sample time setting of all the models was an integer multiple of the step size. This means the sample time of the library block will not be set lower than the fixed step size. This indicated that the target hardware execution speed was directly related to the fix step setting. However, it was not clear how this relationship was defined.

7.4 DAS PC Software Test Results

The DAS controller platform was able to start and stop data capturing on request from the DAS GUI. The DAS controller streamed the selected channel data to the DAS GUI application, and data was plotted in real time (Figure 7.6 and Figure 7.7). The results also verified the command structure and the communication protocol.

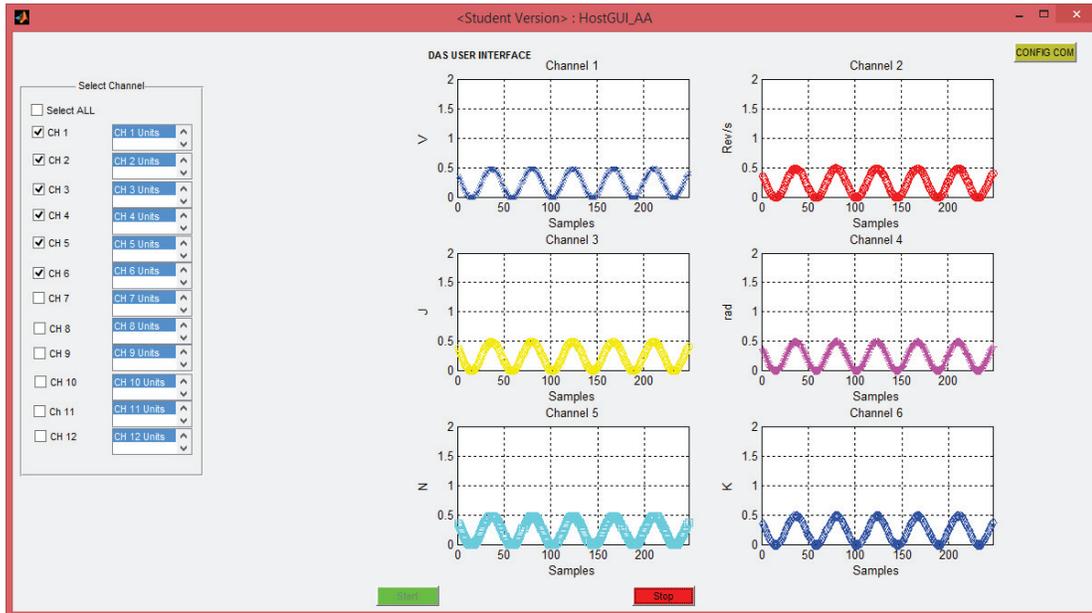


Figure 7.6 DAS GUI real-time plots channel 1-6

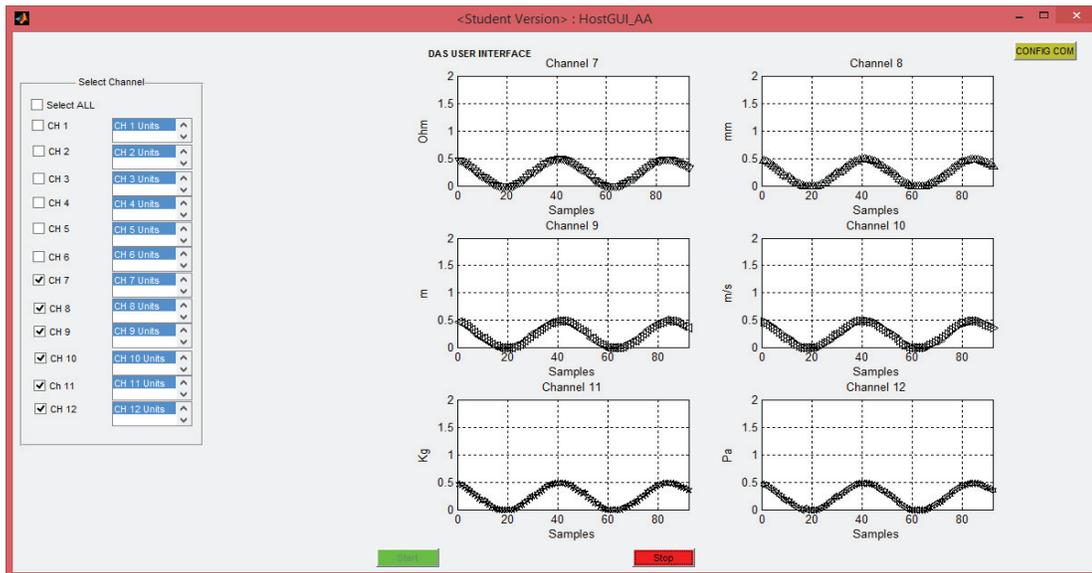
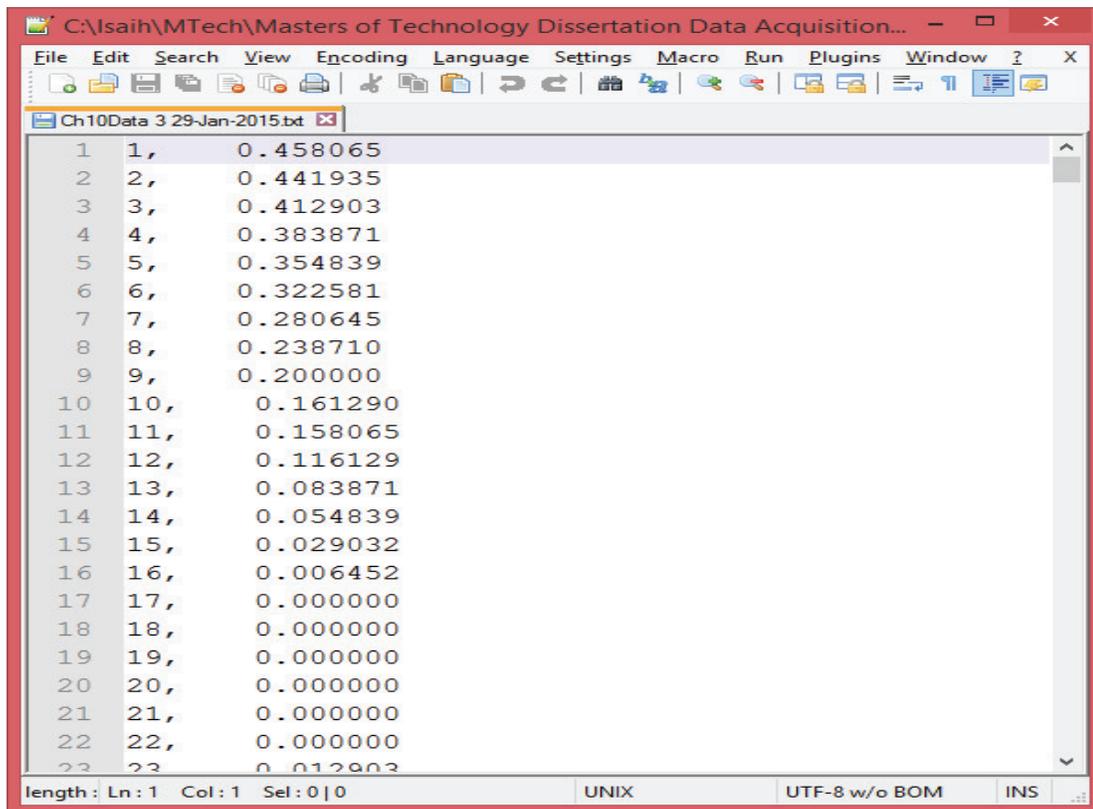


Figure 7.7 DAS GUI real-time plots channel 7-12

The streamed data was stored in a text file and was located in a directory containing the DAS GUI application (Figure 7.8). The file name was date-stamped and increments every time a new data acquisition session was started. However, for testing and debugging purposes, the

data acquisition process was forced to stop in software, only after 500 sampling points were collected. The acquired data was recorded successfully in a data file (Figure 7.8). It was observed that with large data collection (> 500 points) the real-time plotting slows down, thus taking longer to plot the data.



| Line | Value |
|------|----------|
| 1 | 0.458065 |
| 2 | 0.441935 |
| 3 | 0.412903 |
| 4 | 0.383871 |
| 5 | 0.354839 |
| 6 | 0.322581 |
| 7 | 0.280645 |
| 8 | 0.238710 |
| 9 | 0.200000 |
| 10 | 0.161290 |
| 11 | 0.158065 |
| 12 | 0.116129 |
| 13 | 0.083871 |
| 14 | 0.054839 |
| 15 | 0.029032 |
| 16 | 0.006452 |
| 17 | 0.000000 |
| 18 | 0.000000 |
| 19 | 0.000000 |
| 20 | 0.000000 |
| 21 | 0.000000 |
| 22 | 0.000000 |
| 23 | 0.012903 |

Figure 7.8 Saved data file

7.5 ADC Subsystem Test Results

The ADC subsystem test set-up is shown in Figure 3.4. The data was captured using the DAS Host GUI application.

7.5.1 DC Test Results

The measured ADC results were calculated using the following equation:

$$V_{out} = \left(\frac{V_{ref} * ADC_{output}}{FS} \right) * Scale_{factor} \quad (7.1)$$

where V_{out} is the calculated voltage measurement; V_{ref} is the ADC reference voltage, which is +3.3V; FS is full-scale reading of the ADC, which is 1023; ADC_{output} is the actual raw ADC output streamed by the DAS controller; and $Scale_{factor}$ is the scale factor, which was set at one. This was the data stored in the file and plotted on the GUI.

+1.5V DC Input Test Results

The ADC measurement on average gave an output of +1.633V (Figure 7.9). This voltage corresponds with the open circuit voltage of the cell, which was given as 1.6V³¹. The scope measured an average of 1.633V (Figure 7.10). The multimeter measured about 1.624V.

³¹ www.energizer.com

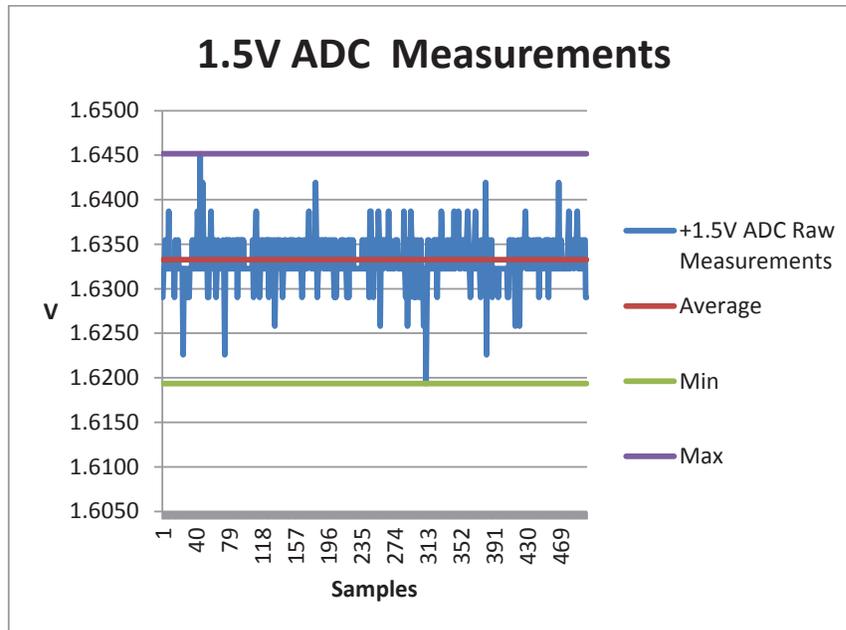


Figure 7.9 +1.5V input ADC measurements

The maximum value measured was 1.645V, which was 12mV above the average value (1.633V). This gave an ADC error of approximately +3 LSB. Furthermore, the minimum measured was 1.62V; this result in an ADC measurement error of approximately -4 LSB. The random output values of the ADC indicated the random measurement errors generated by the ADC (Figure 7.9).

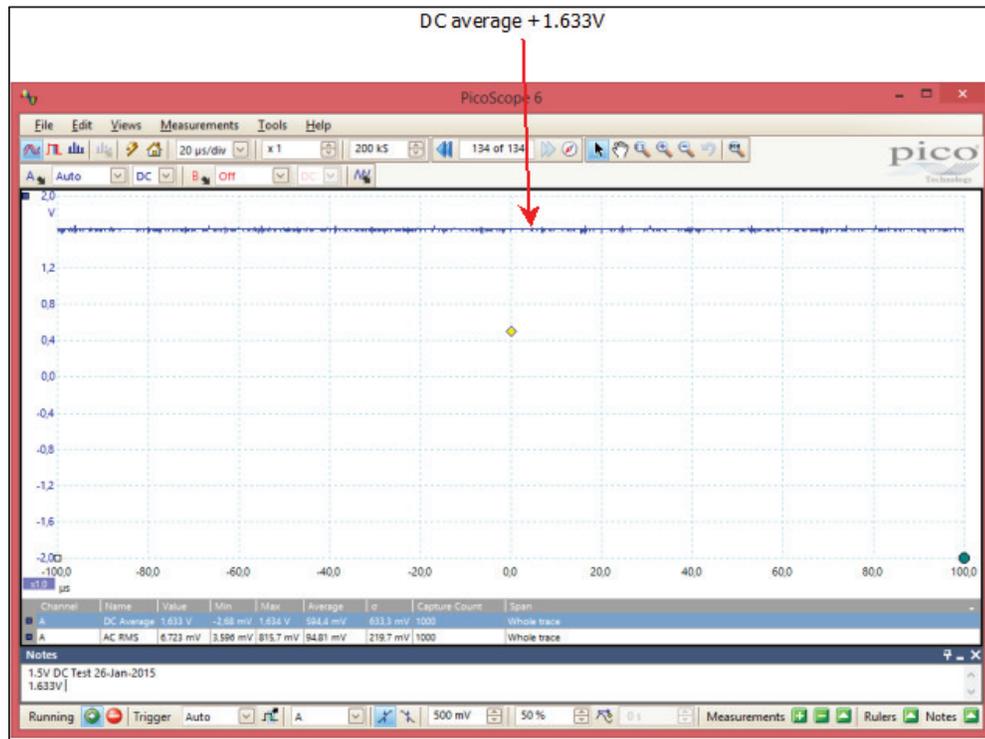


Figure 7.10 PC scope 1.5V measurement

This output assumes a theoretical (true value) reference voltage of 3.3V. However, if the actual reference voltage is measured with a scope (which measured 3.287V) and the result was substituted in equation (7.1), the ADC measurement in this case gave an output of 1.626V (Figure 7.11). This resulted in an error of 7mV, which was about two counts (-2 LSB) less than the ADC output (ADC_{output}) count with true value reference.

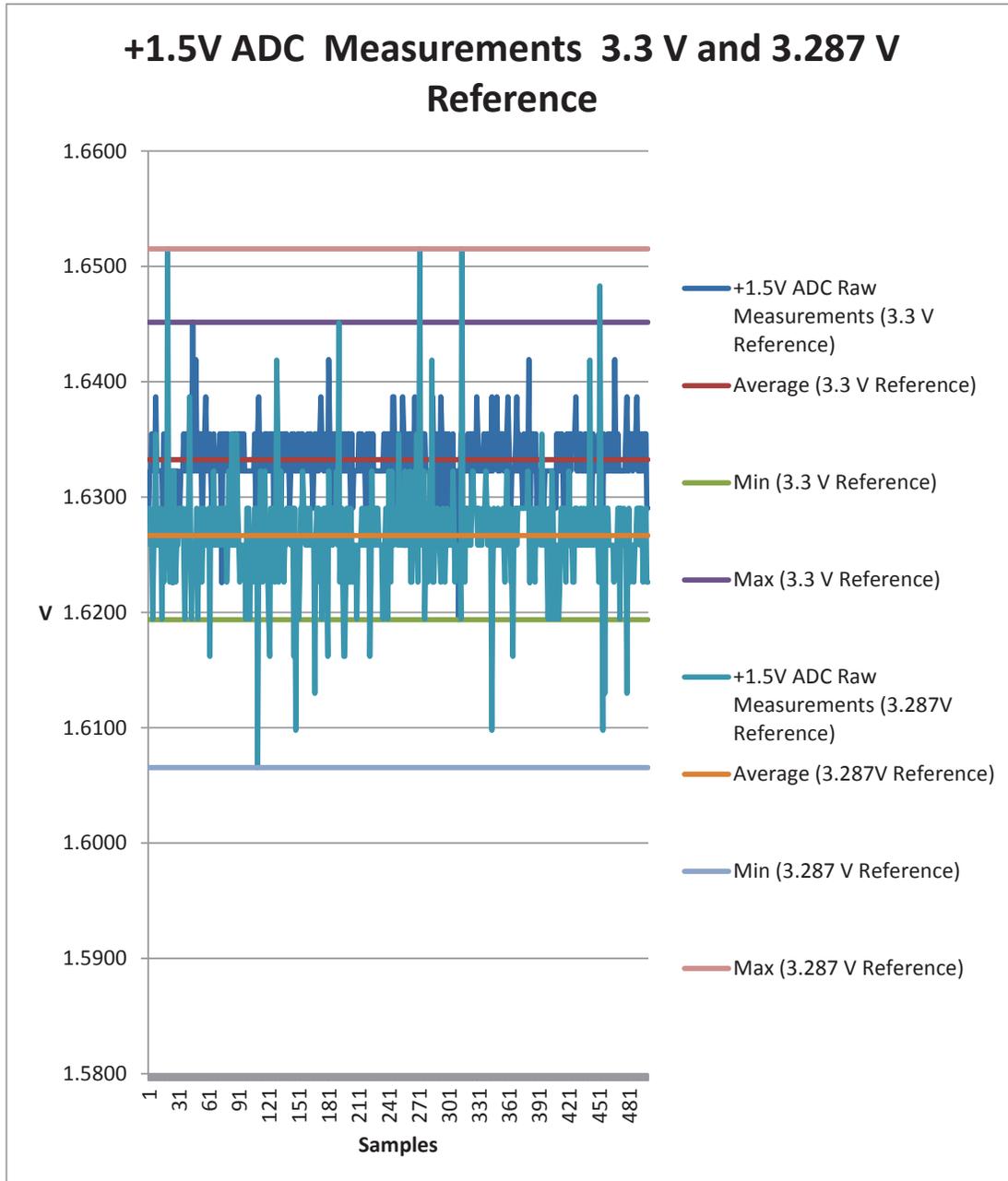


Figure 7.11 +1.5V ADC measurements comparison between 3.3V and 3.287V reference voltage

0V DC Input Test Results

With the input to channels is set at 0V (grounded), the ADC outputs zero volts. This means the ADC was seeing voltage below the +1 LSB, or the ADC has an offset error that was much larger than +1 LSB (3.2227mV).

+3.3V DC Input Test Results

The input to the channels was set at +3.3V; the actual measured value was 3.287V. This voltage was generated on the Arduino® Due hardware platform. The ADC measured average was 3.299V, which is 1mV below the true value (3.3V) and is 12mV more than the scope-measured value of 3.287V (Figure 7.12). The results suggest that the ADC had a gain error of about +3 LSB (12mV / 3.2227mV) plus 0.5 LSB quantisation noise if the actual input signal was taken as 3.287V, assuming a true value reference voltage of 3.3V.

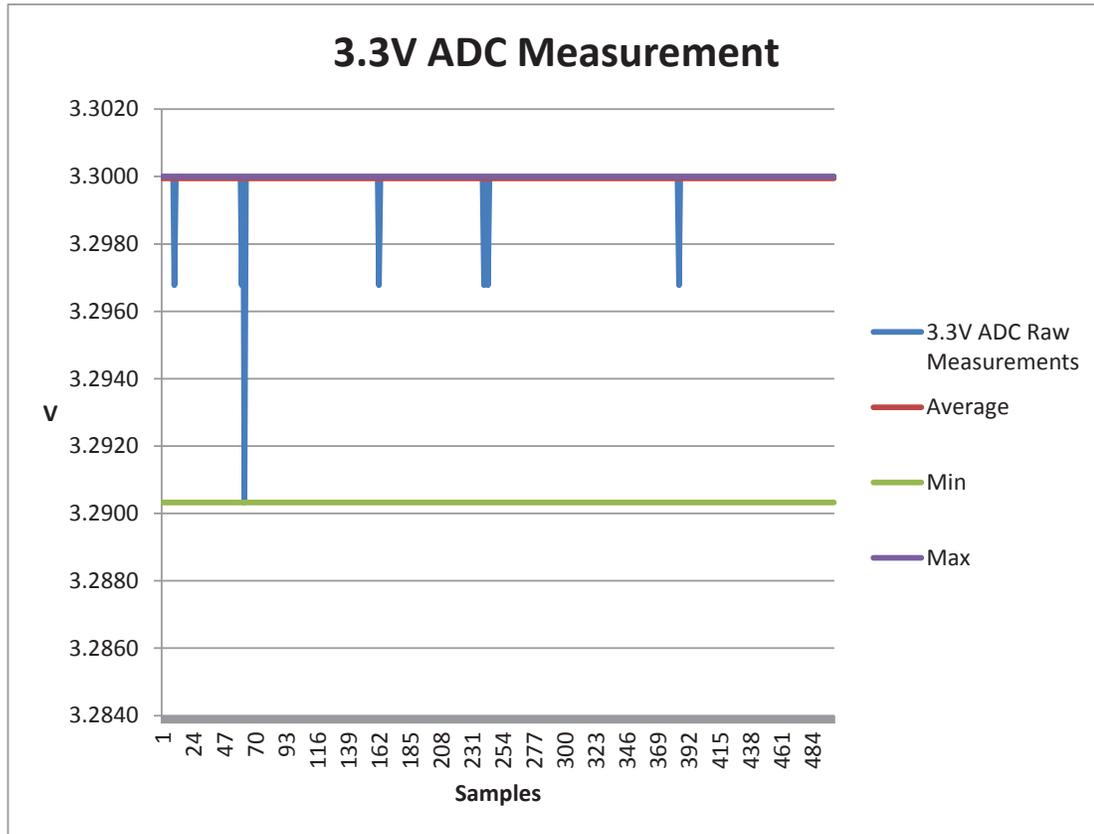


Figure 7.12 +3.3V input ADC measurements

7.5.2 AC Test Results

AC signals set at different frequencies were applied to the channel input, and the output signal was measured. These signals were applied to the ADC inputs, and the ADC outputs were captured and stored in a text file. The AWG test signals were set at 500mV peak-to-peak or 250mV peak; this is shown in Figure 7.14. The data was plotted up to 80 sampling points, but 500 sampling points were captured.

250mVpeak 1Hz Input Signal Test Results

The test was performed using the model discussed in Chapter 6, loaded on the Arduino® Due hardware platform. The sampling rate setting for channel 1 to 10 was 20kSPS, and for channels 11 and 12, it was 100kps. Channels 1, 2, 11 and 12 results are similar, giving an

average voltage of 231mV and a peak-to-peak voltage of 490mV (Figure 7.13). Similar results were observed on other channels (3 to 10) and are comparable to the input signal (Figure 7.14). Similar tests were repeated with signal amplitude increased to 500mV, and 1V and the signal frequency was kept at 1Hz. The results for channel 1 to 12 were still similar, giving identical average and peak-to-peak voltages, which were comparable to the test signal levels.

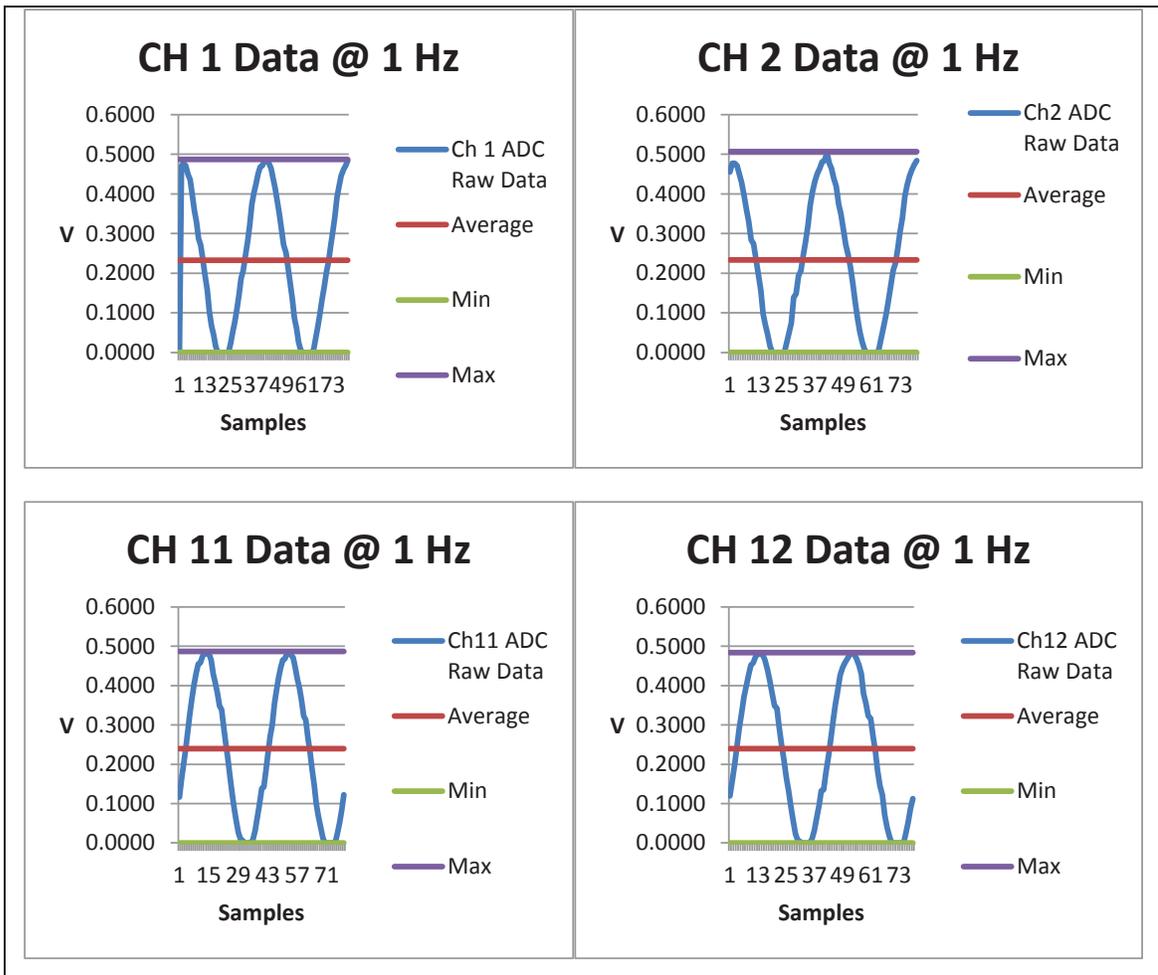


Figure 7.13 250mV 1Hz signal test results

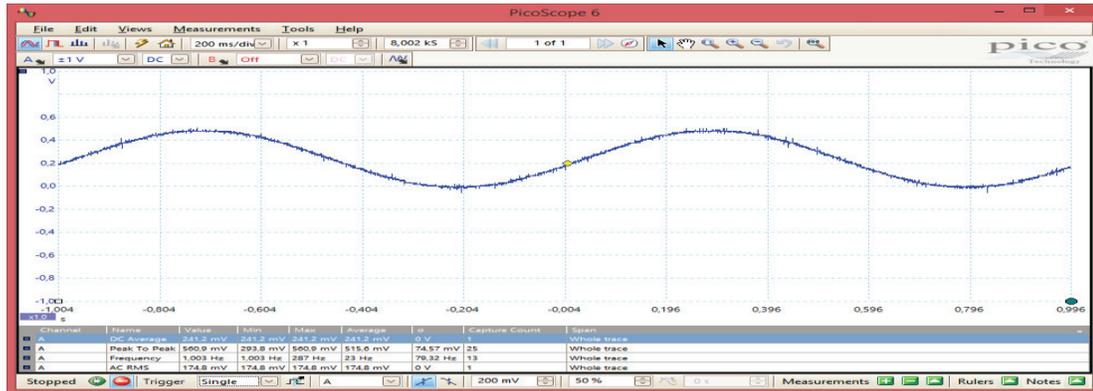


Figure 7.14 Test signal 250mVpeak 1Hz sine wave

250mV 5Hz Input Signal Test Results

The same test was repeated with the signal frequency increased to 5Hz. The output signal was not the same, as the input signal and the output signal had an average voltage of 208mV and peak-to-peak voltage of 448mV (Figure 7.15). This indicated that the ADC channels (1-12) were not able to sample the input signal and showed undersampling, thus giving an erroneous average and peak-to-peak voltages, compared to the test signal levels. The test was repeated with 500mV and 1V amplitude signals, which yielded similar results.

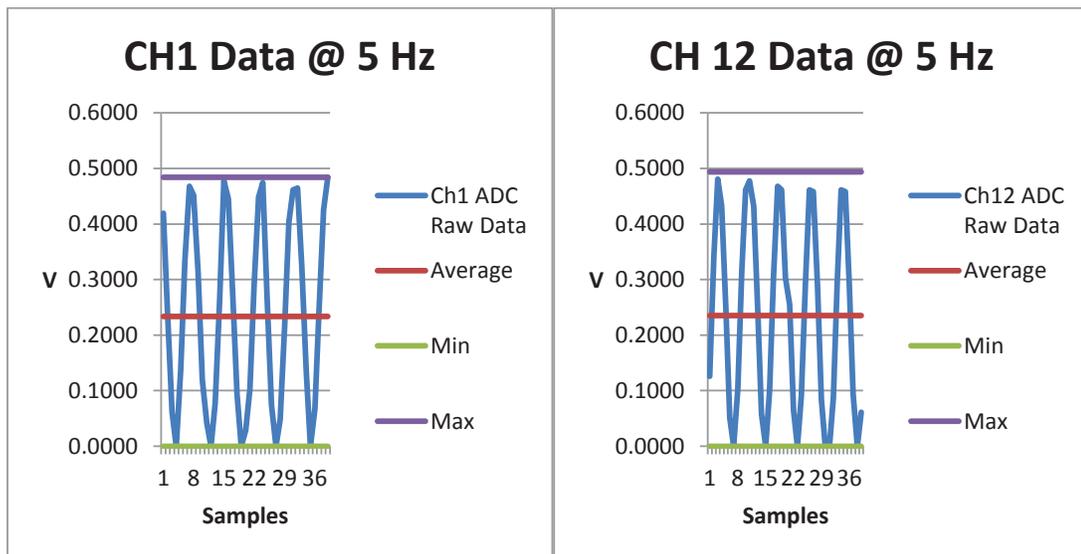


Figure 7.15 250mVpeak 5Hz signal test results (running Arduino Simulink® model)

The sampling setting in the ADC model was changed to 0.000001s (1 μ s) for all the 12 ADC channels, which is the minimum it can allow. This set the maximum sampling rate to 1MSPS. The same test signal of 250mV 5Hz was used.

The foregoing indicated that the signal is not sampled at the optimal rate (Figure 7.16 and Figure 7.17). This was shown by sharp signal edges, which resulted in the sampled signal changing shape from sine wave to a triangular wave. This was observed on other channels.

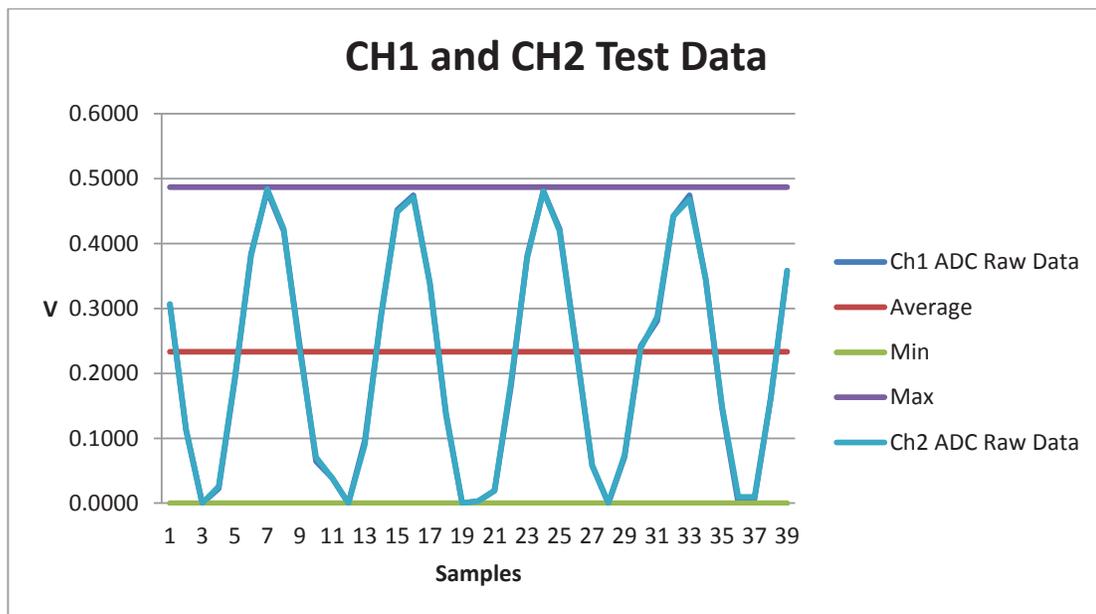


Figure 7.16 Channel 1 and 2 250mV 5Hz signal test results (running Arduino Simulink® model)

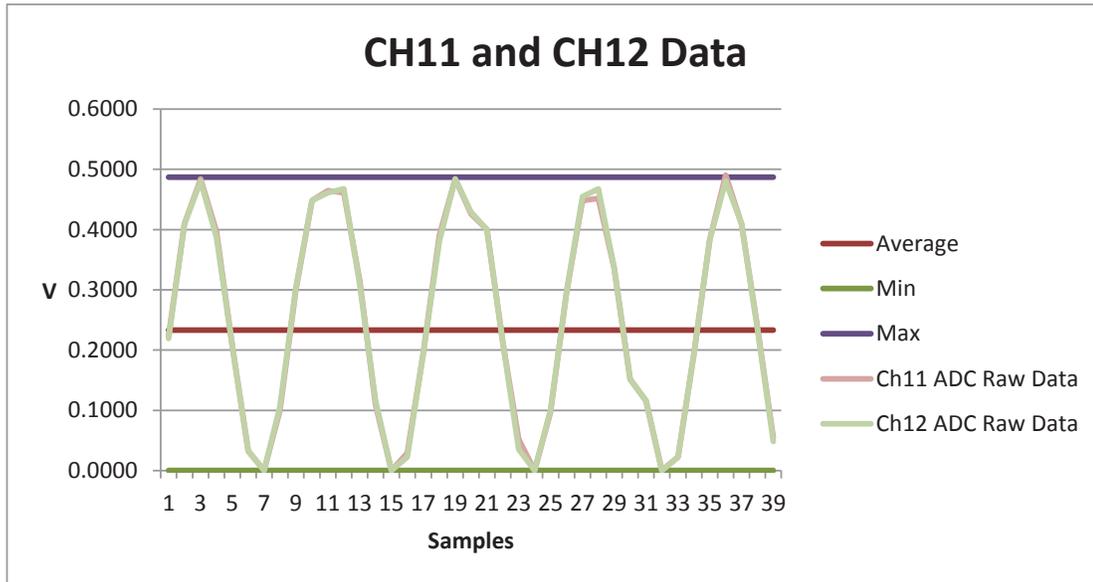


Figure 7.17 Channel 11 and 12 250mV 5Hz signal test results (running Arduino Simulink® model)

The results suggest that the sampling rate is lower than expected. In the model, channels 1 to 10 were initially set to sample at a rate of 20kSPS, and channels 11 and 12 were set at a rate of 100kSPS and then changed to 1MSPS. This was possible as the ADC on the Arduino® Due hardware platform is rated at 1MSPS per channel. Therefore, for 12 channels, it should be able to sample at a rate of 83kHz per channel (equation (5.14)). Moreover, taking processing overheads into account, it should be able to sample at least at a rate of 20kSPS. The test signal frequency was reduced to 2Hz and tests repeated. This indicated that the actual sampling was at a much lower rate and that the input signal bandwidth was at its limit for the sampling rate (Figure 7.18). It seems from the results that the input signal bandwidth was limited to approximately 2Hz for all channels, thus being below design expectations.

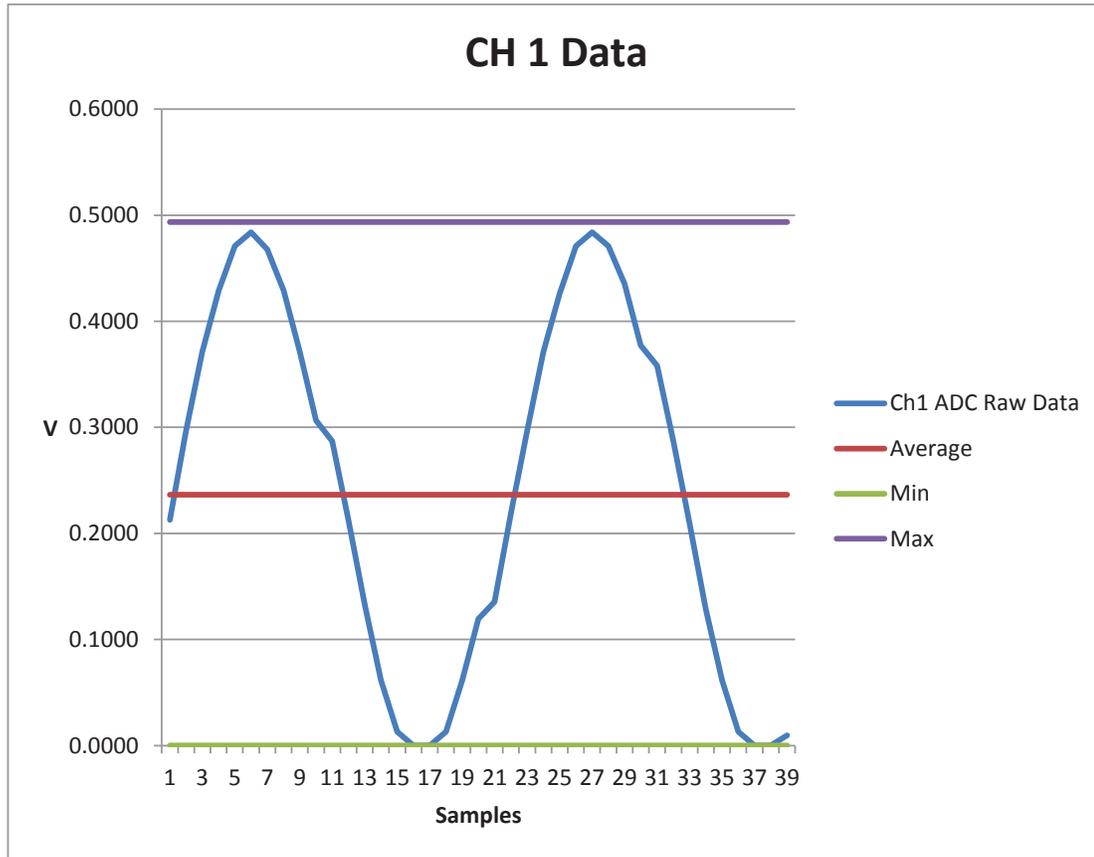


Figure 7.18 Channel 1 250mV 2Hz signal test results (running Arduino Simulink® model)

The aforementioned warranted further investigation to determine the cause of the low sampling rate. The System Alive LED output signal was used to debug and determine the cause of the low sampling rate. When the system is not streaming data, the output signal frequency was about 1.1kHz (Figure 7.19). This was well below what was set in the model (which was 20kHz). The System Alive output signal frequency was reduced by about 18 times when not streaming data. However, when data was streamed, the frequency was further reduced to about 8.2Hz, which was 2 439 times less than the desired frequency (20kHz) (Figure 7.20). Furthermore, the timing or sampling requirements set in the models were not translated to real-time operating and processing speeds. The reduced real-time processing speed affected the sampling rate, hence affecting the input signal bandwidth. It should also be noted that no library blocks were provided to configure and set processor and peripherals

control registers (i.e. used to set ADC clock or operating frequency).

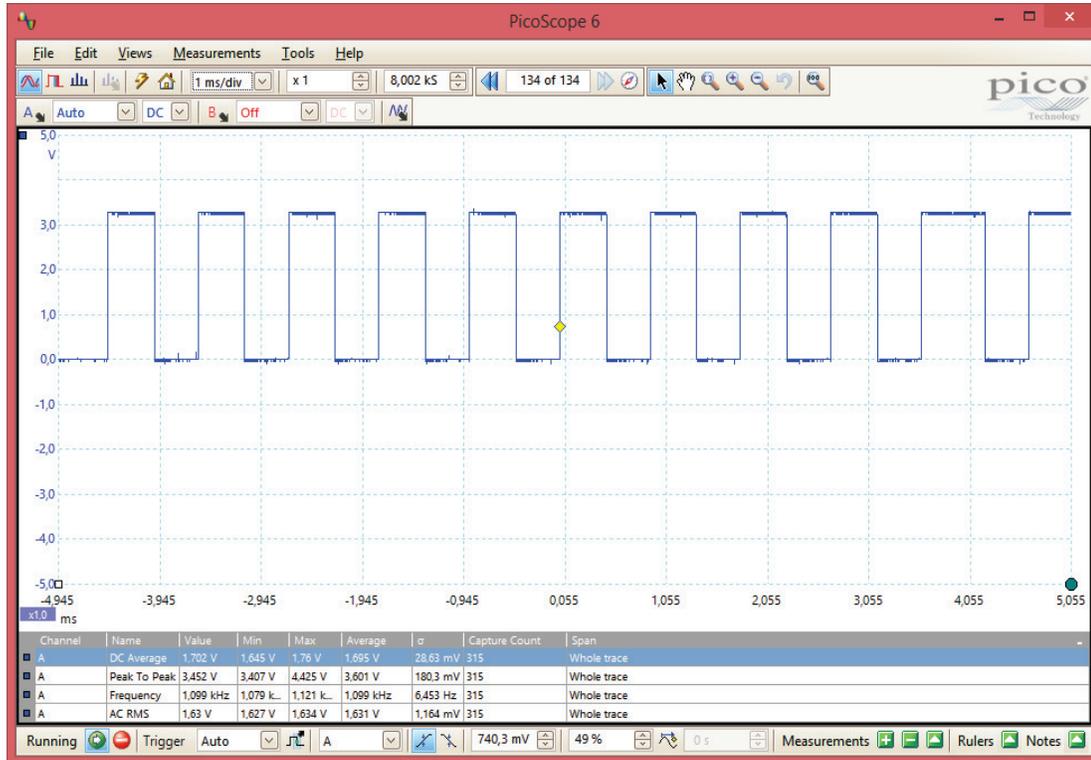


Figure 7.19 System Alive output frequency 1.1kHz no data streaming (running Arduino Simulink® model)

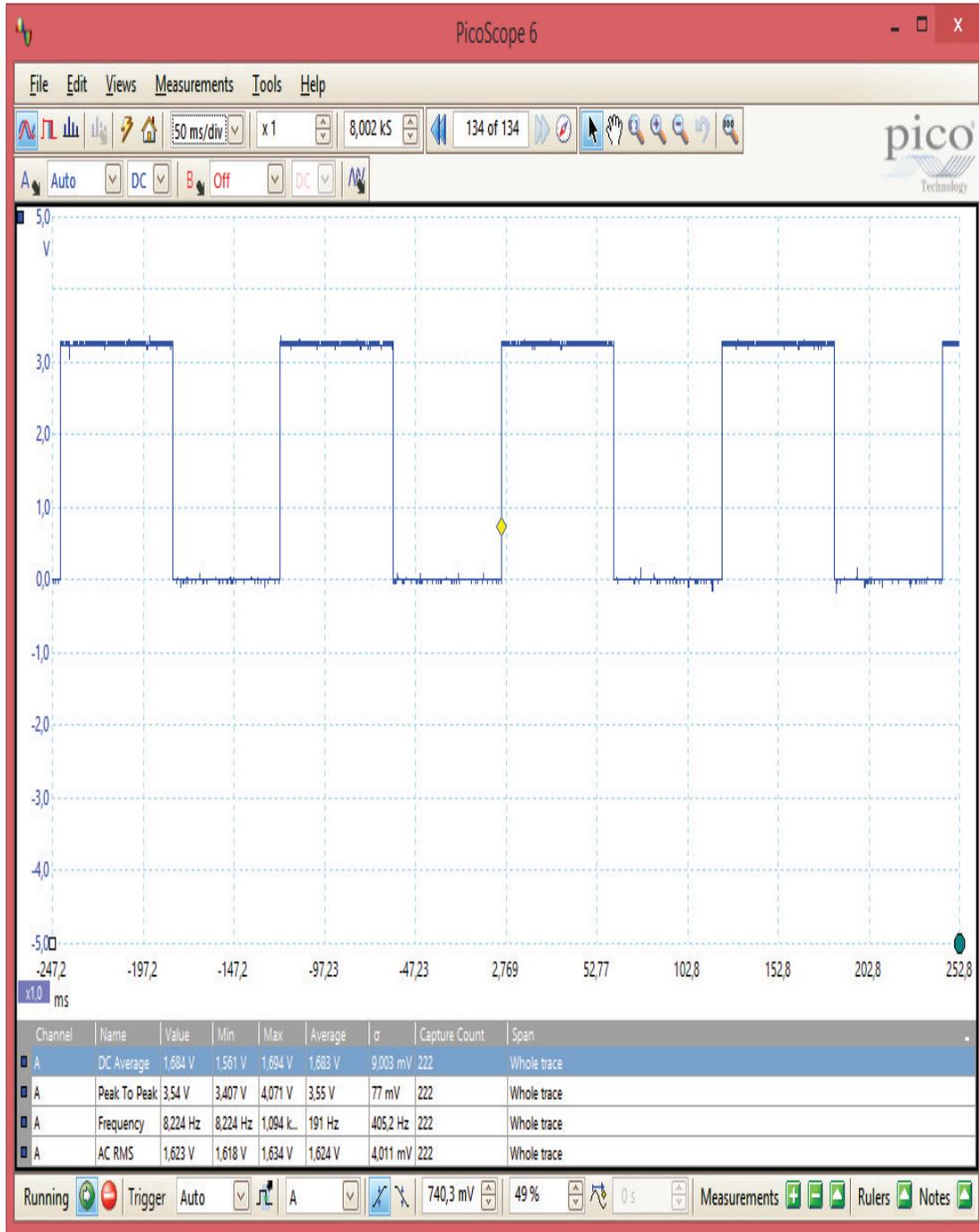


Figure 7.20 System Alive output frequency 8.2Hz data streaming (running Arduino Simulink® model)

Further tests were carried out to verify and confirm the limitations of the provided Arduino[®] library blocks. This included developing a C++ code using the Arduino 1.5.8 Beta IDE³² for Arduino[®] Due hardware platform. The code was based on the algorithm developed for the DAS controller model, thus operates the same as the model, but with the exception of System Alive output. The System Alive output changes states at the beginning and end of the main code. The complete code is listed in Appendix J.

The System Alive output signal was used to determine the operating speed, when no data was streamed and with data streamed from the DAS controller. The System Alive output signal frequency was 96kHz with no data streaming and reduced to 432Hz with data streaming (Figure 7.21 and Figure 7.22). This was much better timing or operating frequency than when running the model.



Figure 7.21 System Alive output frequency 96kHz no data streaming (running Arduino C++ code)

³² <http://arduino.cc/en/Main/Software>

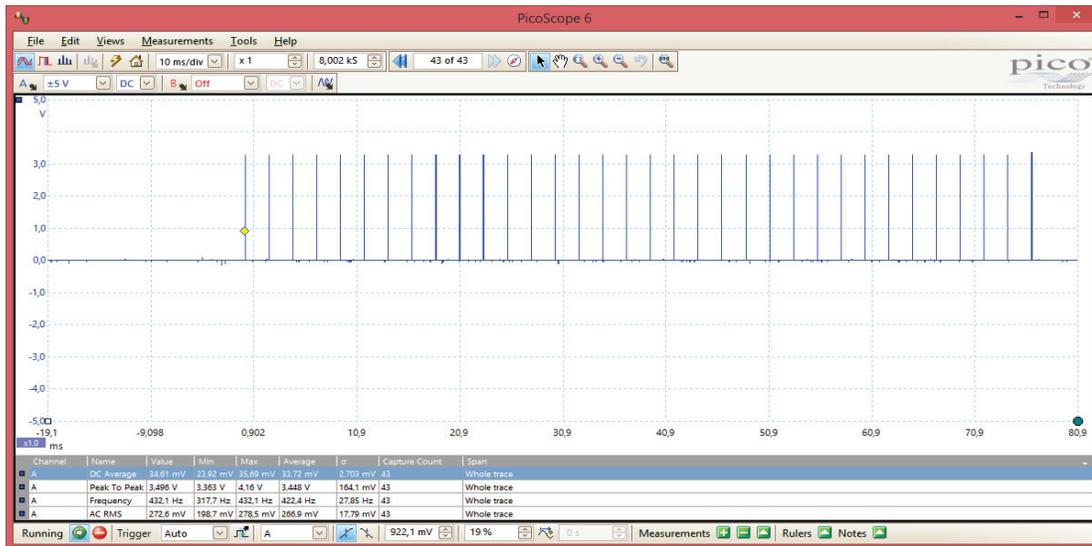


Figure 7.22 System Alive output frequency 432Hz data streaming (running Arduino C++ code)

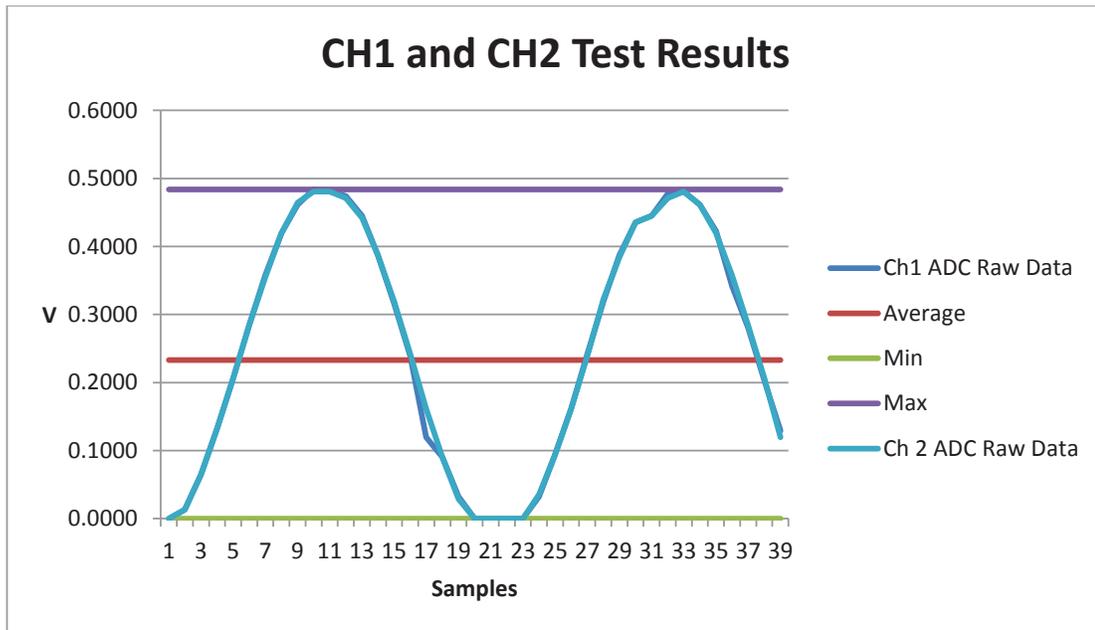


Figure 7.23 Channel 1 and 2 250mV 2Hz signal test result (running Arduino C++ code)

The same tests with a 2Hz signal were repeated. The output signal showed a slight improvement (Figure 7.23.) compared to the output signal in Figure 7.18. Lastly, a test was

performed with a 250mV 5Hz test signal. The output signal in Figure 7.24 was similar to output signals in Figure 7.15, Figure 7.16 and Figure 7.17, indicating undersampling.

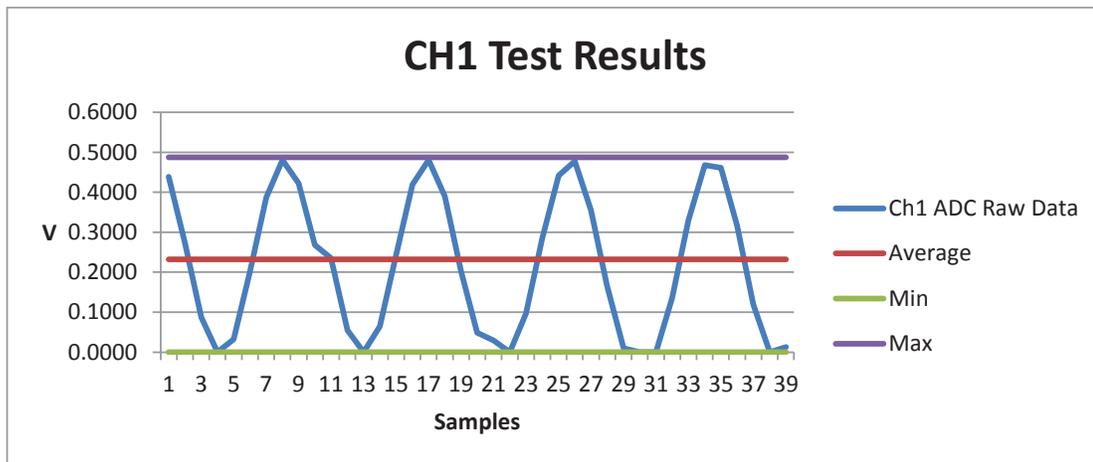


Figure 7.24 Channel 1 250mV 5Hz signal test result (running Arduino C++ code)

Input signals above 5Hz resulted in undersampling and, therefore, generated incorrect data (i.e. missed input signal transitions) (Figure 7.24). The input signal bandwidth was limited at 2Hz (Figure 7.23), which was far below design specifications. The code developed relied on the provided Arduino® library functions³³. The above tests were performed with the system baud rate set at 384 000, which was the maximum the Arduino® Due hardware can handle without communication failures.

7.6 System Calibration

The complete integrated DAS system was calibrated to quantify systematic errors.

7.6.1 Up-scale Calibration

The current loop simulator was used to step up manually through seven set points listed in Table 7.8. The signals in the table were applied to the 4-20mA channels.

³³ <http://arduino.cc/en/Reference/HomePage>

Table 7.8 Current loop calibration set points step-up measurements

| Cal Point | CH1 ADC Measure | CH1 Scope Measure | CH2 ADC Measure | CH2 Scope Measure | CH11 ADC Measure | CH11 Scope Measure | True Value |
|--------------|-----------------|-------------------|-----------------|-------------------|------------------|--------------------|------------|
| 0% (4mA) | 186.6mV | 224.4mV | 185.7mV | 222.5mV | 131.3mV | 168.7mV | 218.5mV |
| 10% (5.6mA) | 274.8mV | 308.8mV | 273.5mV | 308.5mV | 219.3mV | 254.5mV | 305.9mV |
| 25% (8mA) | 406.2mV | 439.5mV | 405.1mV | 438.7mV | 351.3mV | 385.5mV | 437.0mV |
| 50% (12mA) | 626.3mV | 659.3mV | 623.7mV | 657.1mV | 570.6mV | 600.9mV | 655.4mV |
| 75% (16mA) | 846.6mV | 875.2mV | 843.5 mV | 873.3mV | 790.6mV | 820.5mV | 873.9mV |
| 90% (18.4mA) | 978.5mV | 1.004V | 974.6mV | 1.001V | 923.1mv | 952.9mV | 1.0050V |
| 100% (20mA) | 1.0672V | 1.097V | 1.0631 | 1.091V | 1.011V | 1.035V | 1.0924V |

The measurements in Table 7.8 were used to plot the transfer function of the channels. The best-fit line was obtained using Excel® trend line function. The best-fit line was linear and represented a straight line equation. Channel 1 transfer function was $y_{ch1} = 55.013x - 0.0336$ (Figure 7.25) and channel 2 transfer function was $y_{ch2} = 54.81x - 0.0335$ (Figure 7.26), where x was the input voltage ($I_{4-20mA} * R_{in}$). The y-intercepts for y_{ch1} and y_{ch2} are almost identical, which suggests the same offsets for the channels. The slope (sensitivity) difference could be due to components tolerances, ADC errors, op-amp errors and measurement uncertainty. This was observed with other 4-20mA 5kHz channels.

For channel 11, the transfer function was $y_{ch11} = 54.976x - 0.0887$ (Figure 7.27). The y-intercept or offset was much larger than the other channels (i.e. channel 1-6 for 4-20mA signals). This was because of the different op-amp used in implementing this channel. The channels transfer functions represented a unique relationship between the input signal and the ADC channel output. The transfer functions parameters were adjusted to compensate for channel errors.

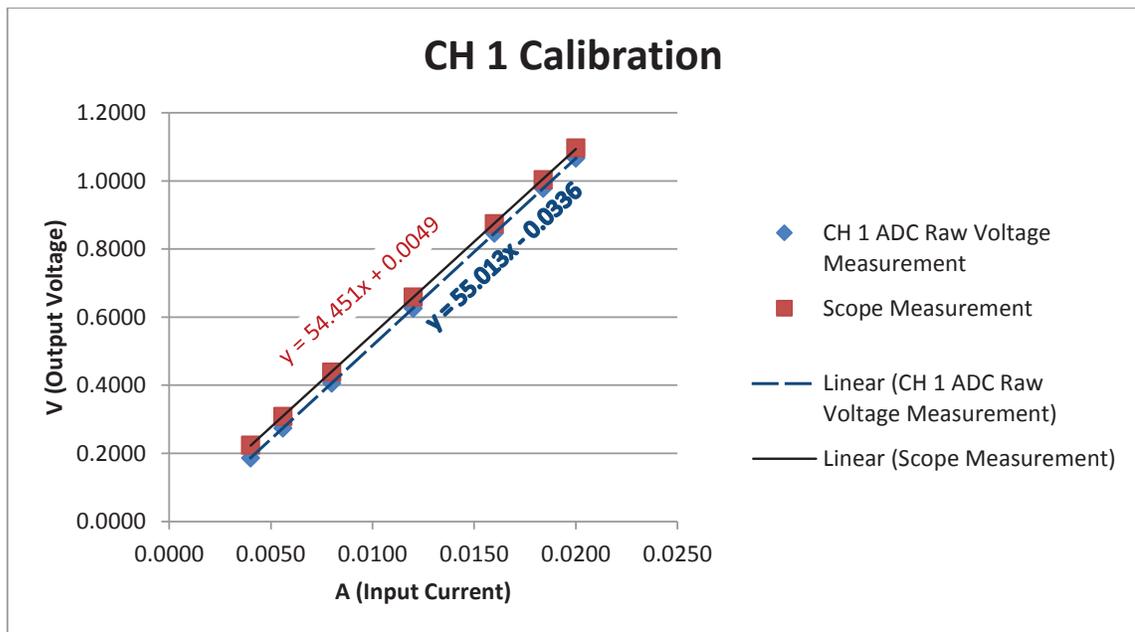


Figure 7.25 Channel 1 transfer function

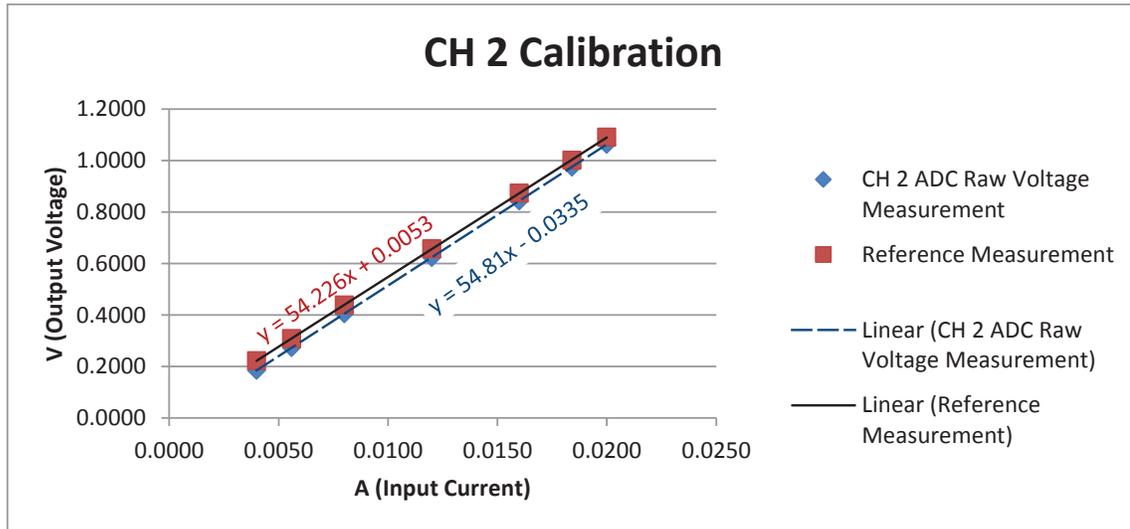


Figure 7.26 Channel 2 transfer function

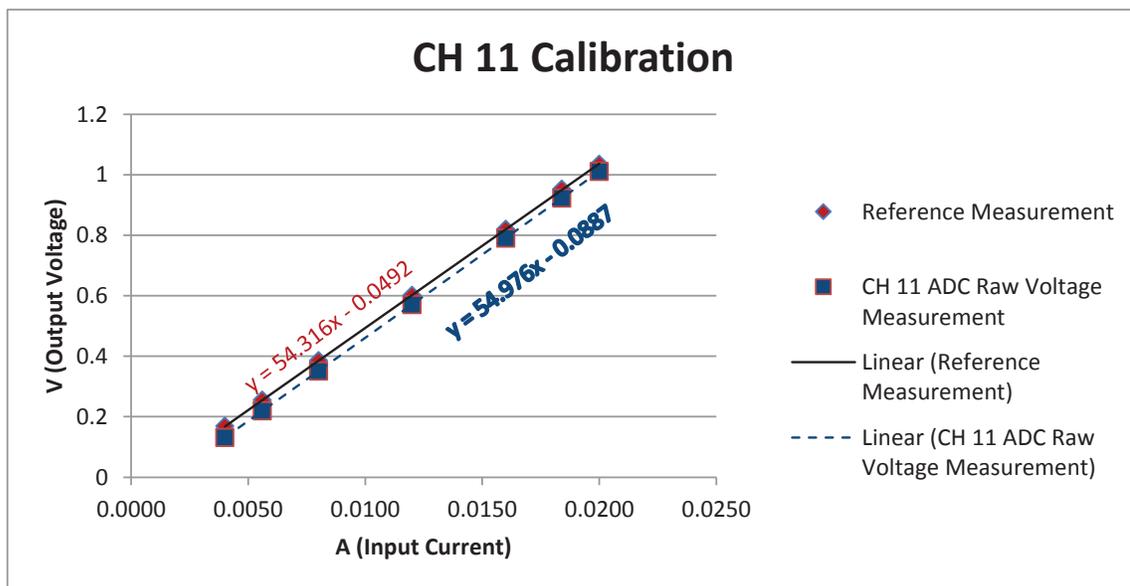
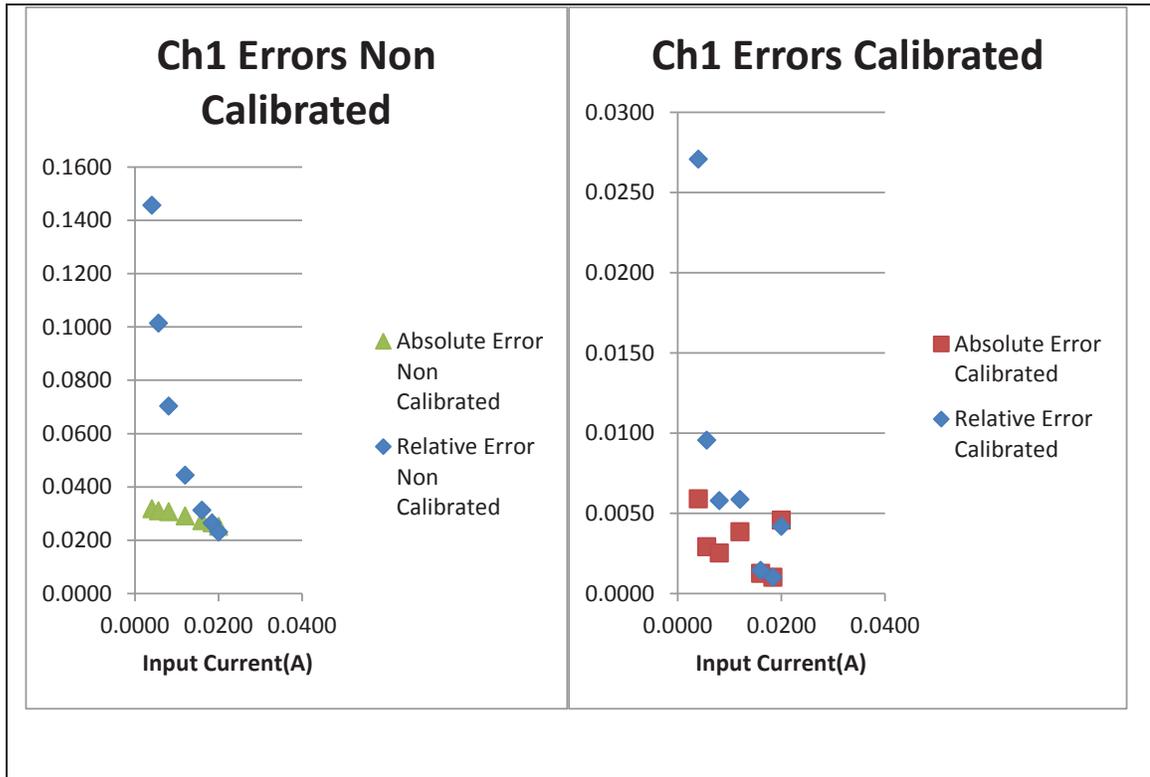


Figure 7.27 Channel 11 transfer function

Each channel was calibrated individually, using the transfer function obtained for the channel. The calibration curve for channel 1 was given as $y = 54.5451x + 0.0049$ (Figure 7.25); for channel 2, it was $y = 54.226x + 0.0053$ (Figure 7.26); and for channel 11, it was $y = 54.316x -$

0.0492 (Figure 7.27). The values obtained from the DAS system were adjusted to fit into the calibration curve obtained for each 4-20mA channel.

From Table 7.8 data, the absolute³⁴, relative³⁵ error and accuracy³⁶ for channels 1 and 2 are shown in Figure 7.28 and Figure 7.29, and for channel 11, they are shown in Figure 7.30. The accuracy was worse on the low-end measurements (4mA) for the 4-20mA channels because of the low-level voltage generated at this current. These voltages were affected by the op-amp DC offsets, ADC errors and components tolerances. Channel 11 had worse accuracy due to the op-amp choice. The absolute and relative errors were decreased after calibration and with accuracy increasing.



³⁴ absolute error = $|true\ value - measured\ value|$ (Dunn, 2014; Rabinovich, 2010)

³⁵ relative error = $\frac{Absolute\ error}{|true\ value|}$ (Dunn, 2014; Rabinovich, 2010)

³⁶ accuracy = $1 - relative\ error$ (Dunn, 2014)

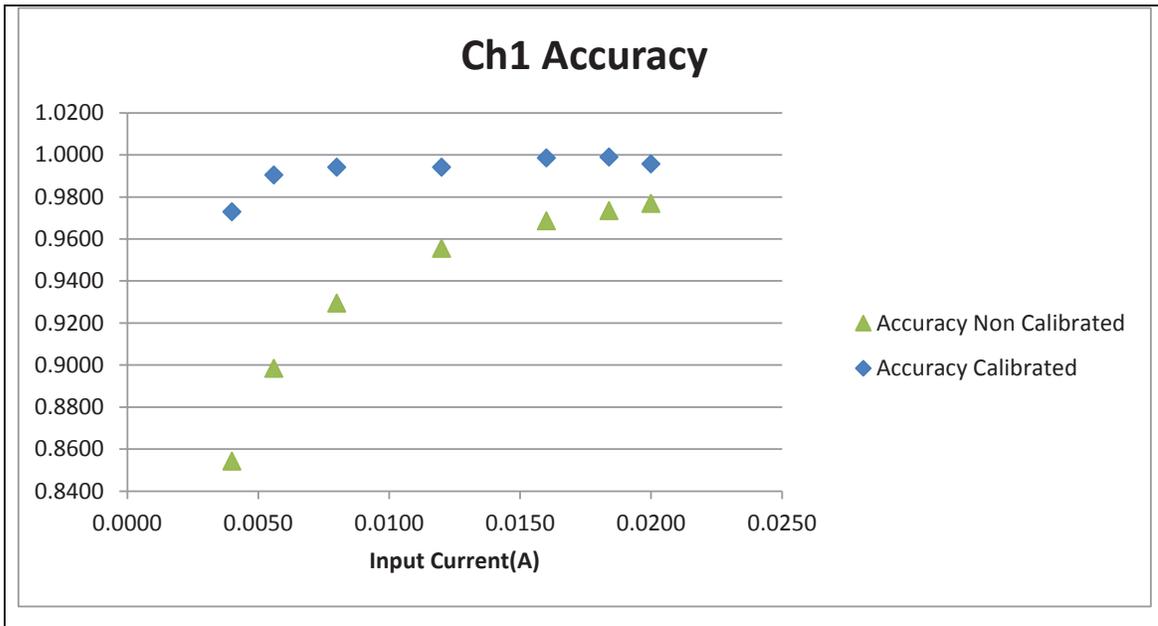
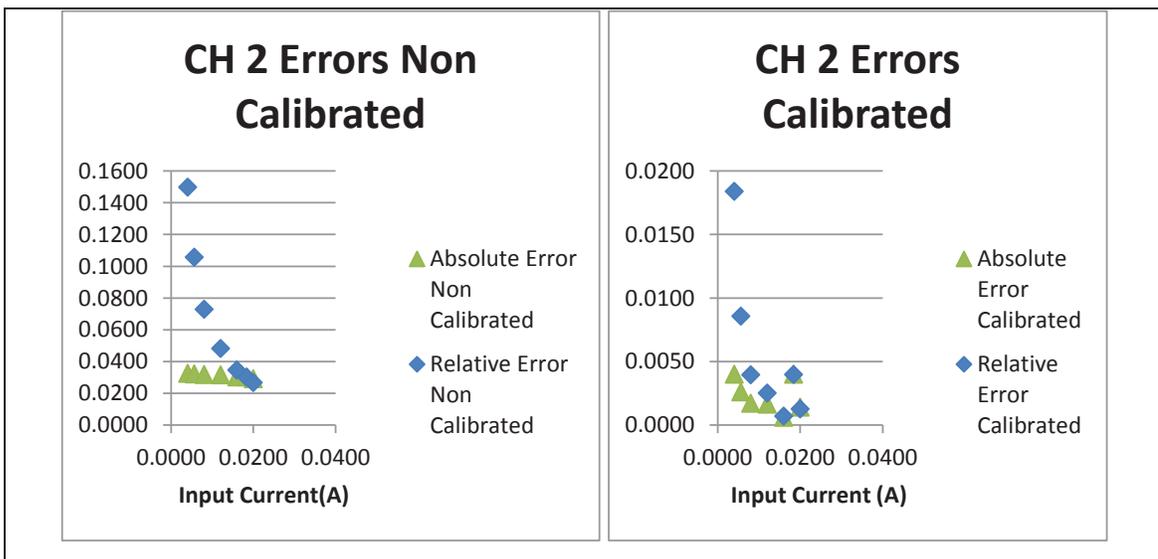


Figure 7.28 Channel 1 absolute, relative error and accuracy



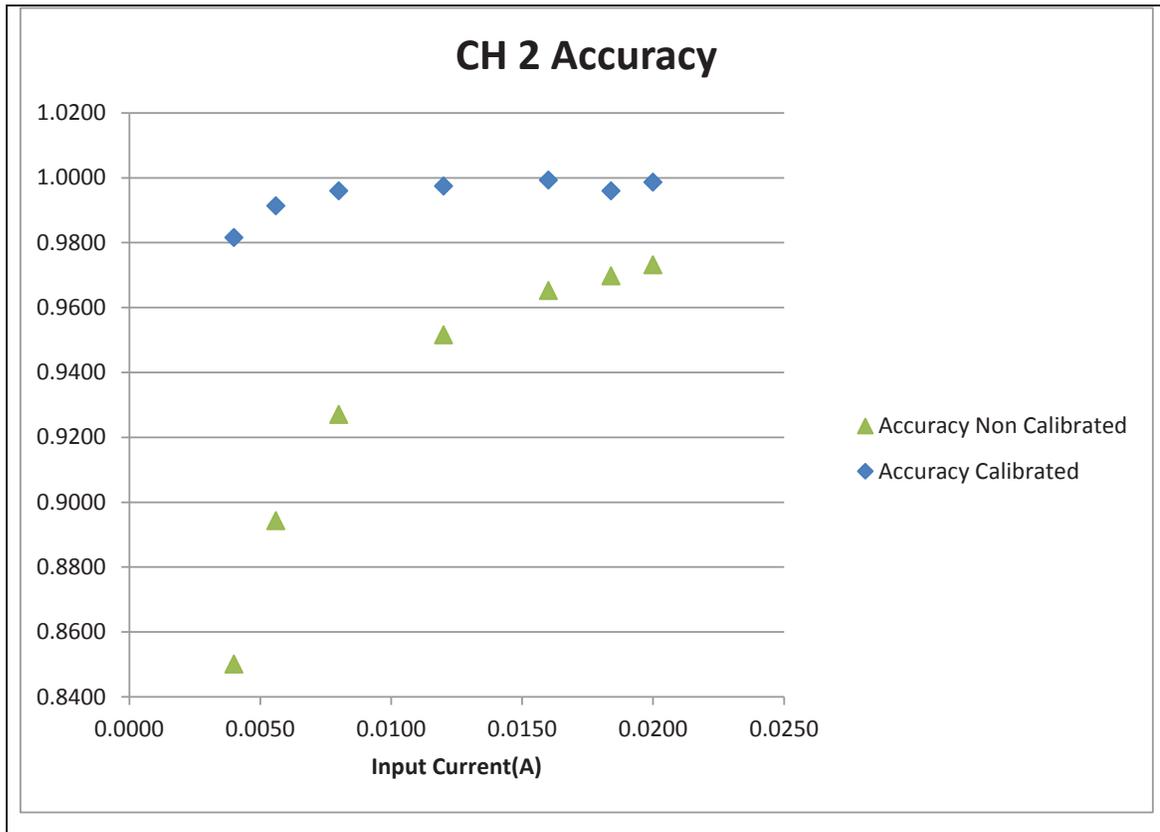


Figure 7.29 Channel 2 absolute, relative error and accuracy

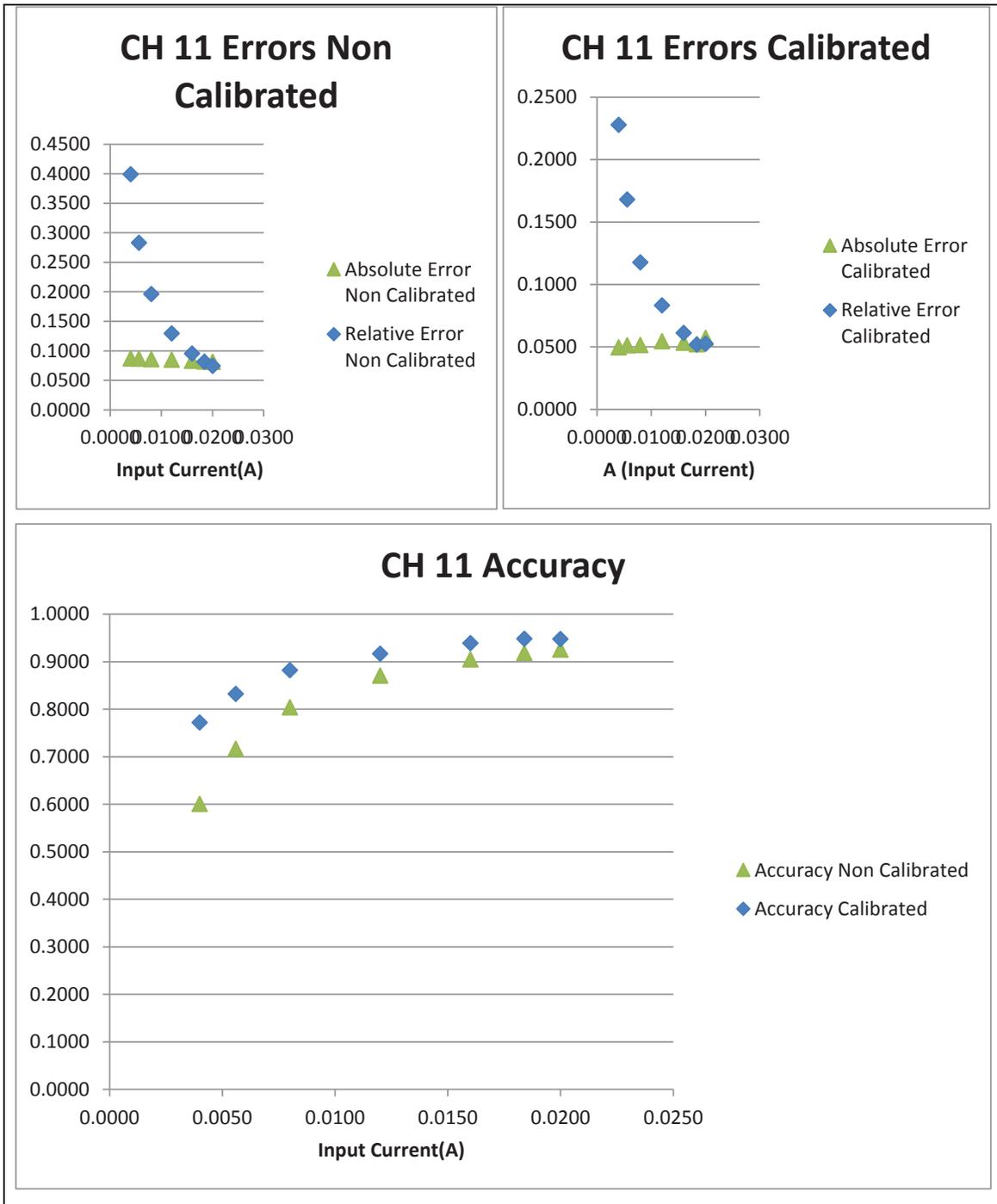


Figure 7.30 Channel 11 absolute, relative error and accuracy

The voltage loop set points and measurement results are listed in Table 7.9. This signal was

applied to the 0-10V channels.

Table 7.9 Voltage loop calibration set points step-up measurements

| Cal Point | CH8 ADC Measure | CH8 Scope Measure | CH9 ADC Measure | CH9 Scope Measure | CH12 ADC Measure | CH12 Scope Measure | True Value |
|------------|-----------------|-------------------|-----------------|-------------------|------------------|--------------------|------------|
| 0% (0V) | 0V | 0V | 0V | 0V | 0V | -53.43mV | 0V |
| 10% (1V) | 212.2mV | 246.7mV | 213.8mV | 248.4mV | 148.4mV | 190.1mV | 242.4mV |
| 25% (2.5V) | 577.3mV | 608.5mV | 580.1mV | 609.8mV | 512.3mV | 552.7mV | 606.1mV |
| 50% (5V) | 1.1878V | 1.211V | 1.1910V | 1.212V | 1.1545V | 1.157V | 1.2121V |
| 75% (7.5V) | 1.7998V | 1.817V | 1.8033V | 1.819v | 1.7675V | 1.759V | 1.8182V |
| 90% (9V) | 2.1664V | 2.186V | 2.1705V | 2.187V | 2.1335V | 2.135V | 2.1818V |
| 100% (10V) | 2.4103V | 2.409V | 2.4152V | 2.409V | 2.3808V | 2.368V | 2.4242V |

The best-fit line was obtained using Excel® trend line function. Channel 8 transfer function was given by $y_{ch8} = 0.2426x - 0.0195$, where x was the input voltage (Figure 7.31). Moreover, channel 9 transfer function was $y_{ch9} = 0.2415x - 0.0186$ (Figure 7.32). The y-intercept and slope were almost identical. This was also observed with other 0-10V 5kHz channels.

For channel 12, the transfer function was expressed as $y_{ch12} = 0.2433x - 0.0594$ (Figure 7.33). The slope were identical to other channels (i.e. channel 7-10 for 0-10V signals), and the offset was slightly larger than the other channels. The 0-10V channels (7-10) can be represented by a single transfer function, with a relatively minimal error (Figure 7.31, Figure 7.32 and Figure

7.33).

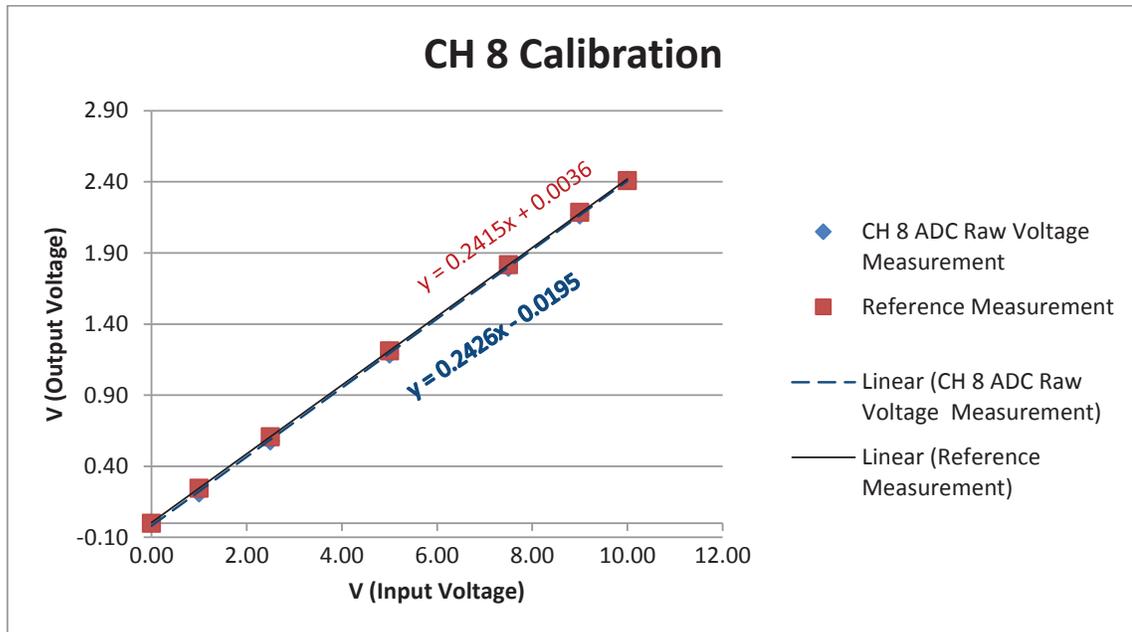


Figure 7.31 Channel 8 transfer function

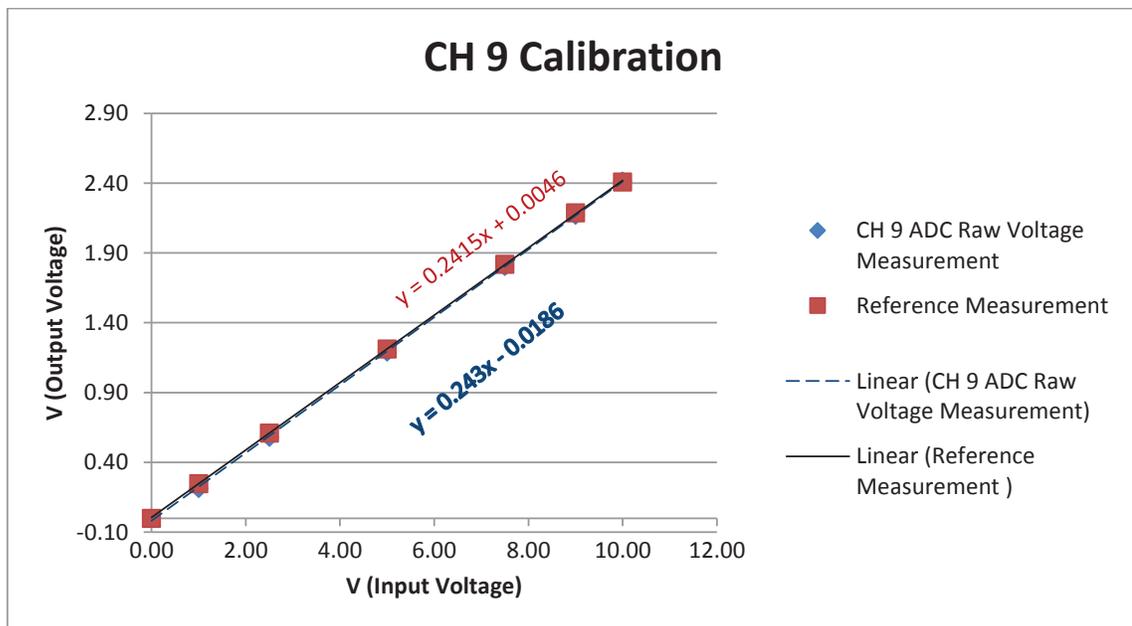


Figure 7.32 Channel 9 transfer function

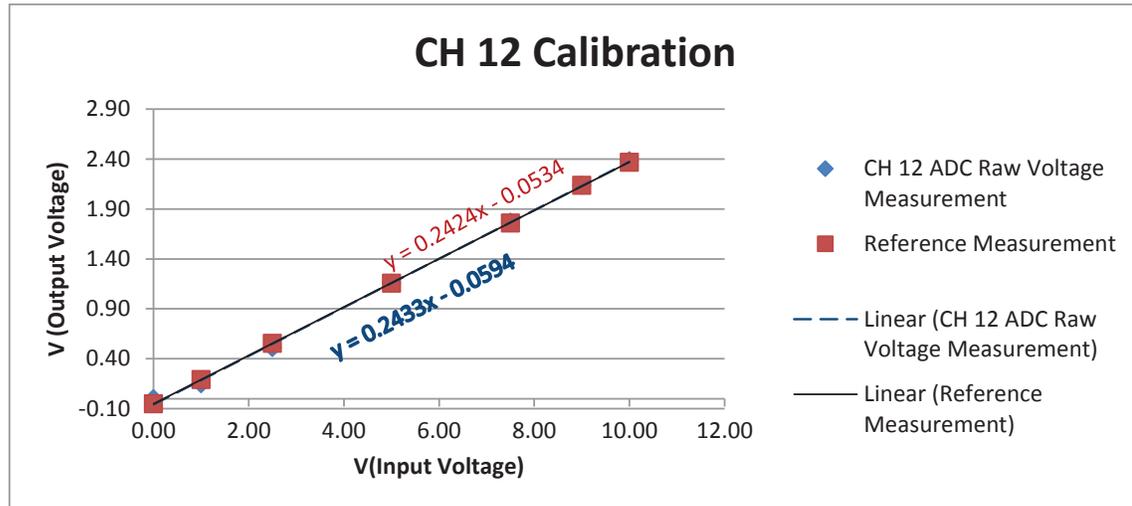


Figure 7.33 Channel 12 transfer function

The calibration curve for channel 8 was $y = 0.2415x + 0.0036$; for channel 9, it was $0.2415x + 0.0046$; and for channel 12, it was $y = 0.2424x - 0.0534$. The transfer functions for channel 7-10 were almost identical, suggesting that one calibration curve can be used for calibration and also that not much correction was required (Figure 7.31, Figure 7.32 and Figure 7.33).

However, for channel 12, the difference was the negative offset value. The values obtained from the DAS system were adjusted to fit into the calibration curve obtained for each 0-10V channel. The transfer functions obtained for the channels represent the whole signal chain path characteristics. This shows the relationship between the input (4-20mA or 0-10V signal) to the output (ADC raw data), which helped in identifying systematic errors. Therefore, the calibration adjustments will compensate for DC offset errors due to op-amp, gain and offset errors of the ADC and aid in reducing the systematic errors of the channels.

From Table 7.9 data, the absolute, relative error and accuracy of the 0-10V channels 8 and 9 are shown in Figure 7.34 and Figure 7.35. Additionally, for channel 12, they are shown in Figure 7.36. Channel 12 had a poor absolute and relative error and accuracy (as low as 22%) at low-end inputs 0-1V (Figure 7.36) because of the poor DC performance of the op-amp used in the implementation of the channel. The channels for 0-10V signals had relatively low accuracy ($\pm 2\%$) at low-end inputs (Figure 7.34 and Figure 7.35), which was due to their

sensitivity to the effects of op-amp DC offset, ADC errors and components tolerances and measurement uncertainty.

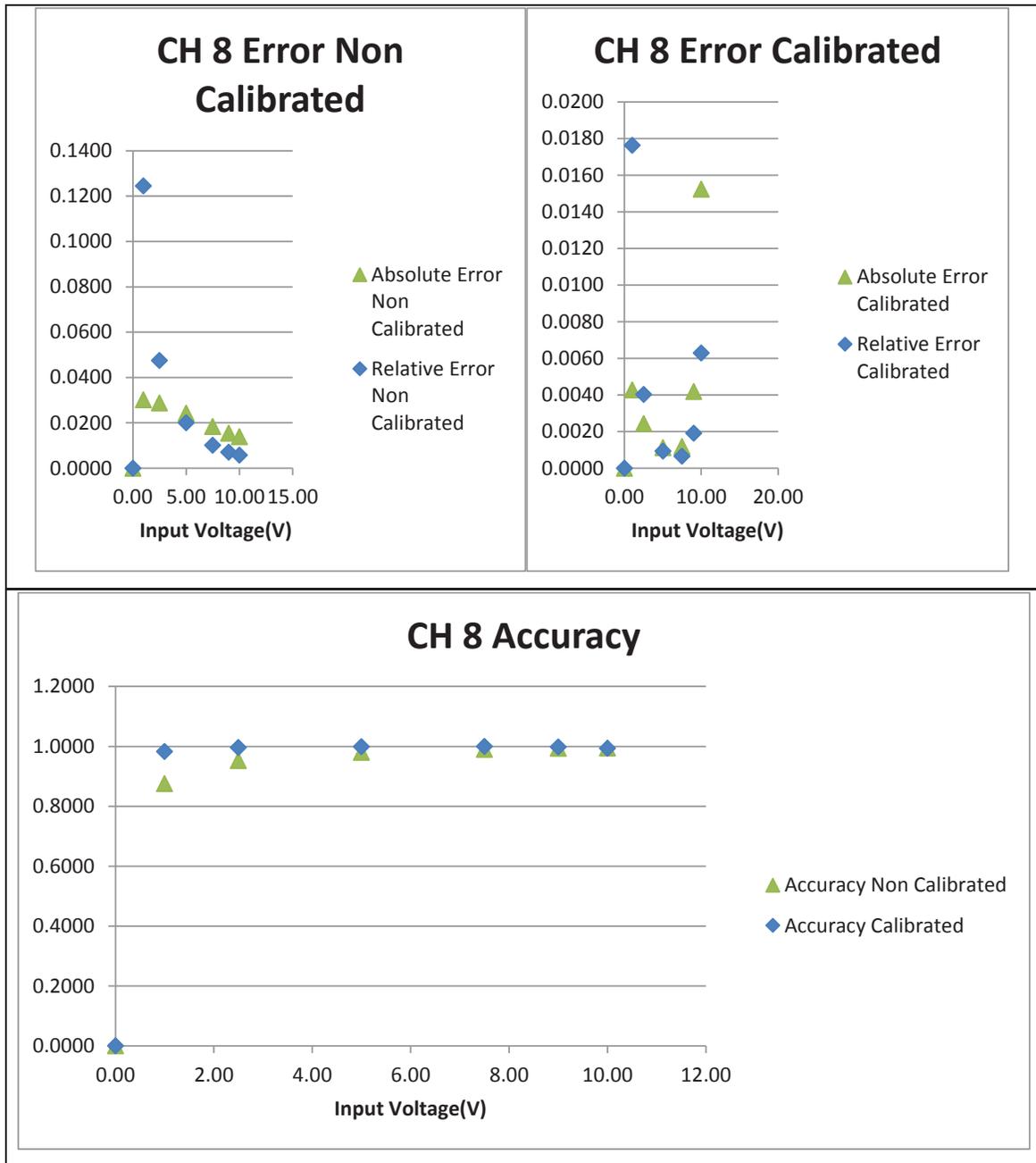


Figure 7.34 Channel 8 absolute, relative error and accuracy

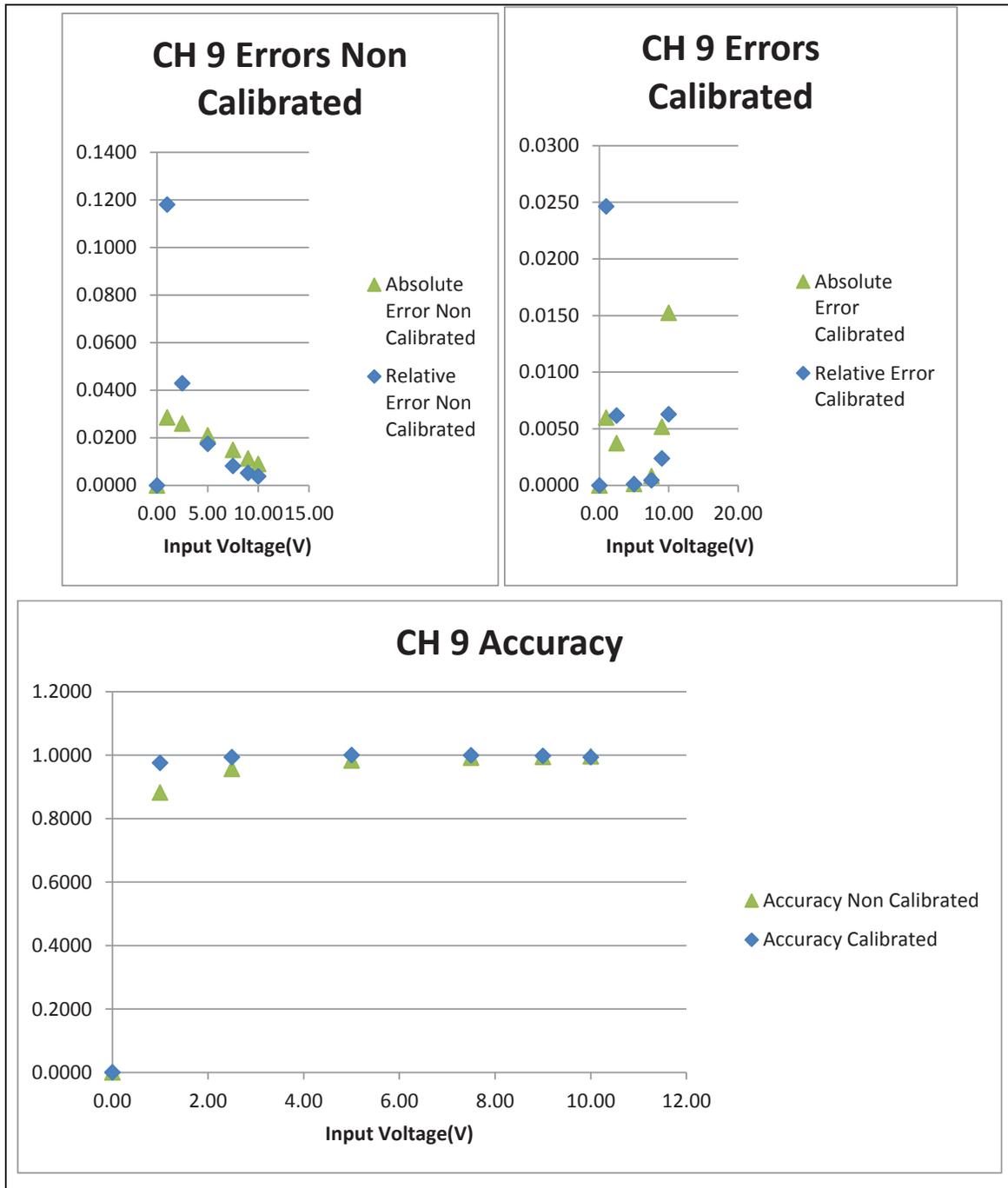


Figure 7.35 Channel 9 absolute, relative error and accuracy

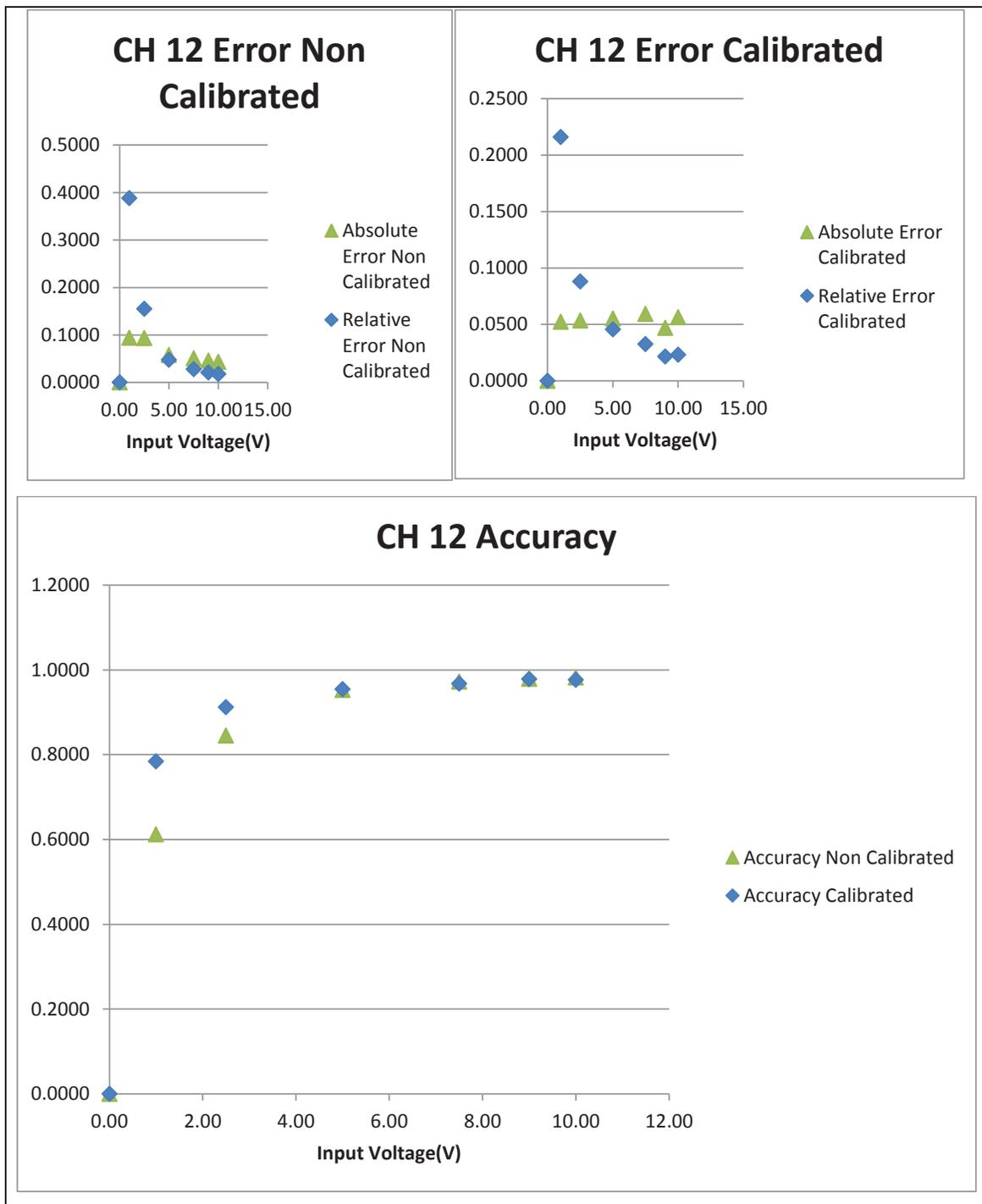


Figure 7.36 Channel 12 absolute, relative error and accuracy

7.7 Limitations

The required operating speed (96MHz) could not be achieved. This placed severe limitations on implementing the system successfully. The real-time sampling rate (20kSPS and 100kSPS) required could not be achieved. Moreover, due to these limits set by the ADC subsystem implementation, the input signal bandwidth could not exceed 2Hz. This also placed limits in plotting data. The data could not be plotted with respect to time. The system calibration was only limited to static and up-scale calibration. The platform and libraries in this current version were used to design and develop the DAS controller model, algorithms, verifying and testing the model. The library was limited, as it did not provide functions that allowed low-level access to configure and set processor and peripherals control registers. The foregoing results indicate that the library provided for both the Simulink® models and integrated development environment (IDE), which were limiting factors on the performance of the ADC and the processors. This affected overall performance of the Arduino® Due hardware platform in real time.

The platform and development tools were not suitable for demanding and complex systems, such as required for this project. It was proved that this platform and libraries are only limited for proof of concepts, testing of protocols, verification of model logics and, thus, not suitable for high-end system design and implementation. The DAS system operated as modelled and was calibrated. However, the system was limited to measure input signals from DC to 2Hz. The GUI application showed poor plotting performance, as the streamed data collected was becoming larger. The initial emphasis was to develop a GUI application that can be used to record the collected data in a file and use this data to verify and validate the data acquisition process. However, there was still much development work to be done to improve the performance of the application.

Chapter 8 Conclusion and Recommendations

8.1 Conclusion

The penultimate chapter focused on the experimental results and analysis for this study/project. This final chapter will discuss the conclusion and recommendations for this study. The main objective of this research is to develop and design a data acquisition system that measures, processes, stores and displays raw signals collected from sensors and transducers on a pilot mill

However, the aim of this project was to develop a data acquisition system that will collect raw data from sensors embedded in a pilot mill. These sensors will transmit measurand data via a 4-20mA and 0-10V analogue data communication protocol. The DAS will collect this data and digitise it and store it in a file; no digital signal processing is performed on the data. The data was collected accurately within the limits of the hardware. Furthermore, system engineering principles were applied successfully in the design, testing and verification of the entire system.

The development was done within the context of new technologies used in system development. It included using open-source rapid prototyping boards such as Arduino Due® hardware platform and developing a code using MATLAB® Simulink® library blocks, applying a model-based software development concept. This model-based development used a Simulink® environment and Arduino® library blocks to design and develop DAS controller algorithms used for raw data collection and the streaming of collected data to the host system for storage based on models. This was achieved successfully.

There were challenges encountered in the project due to the use of the above-mentioned rapid prototyping platform and model-based design. These included limitations such as hidden low-level implementations, inability to configure ADC sampling rate that corresponded to real-time requirements of the system, and failure of the development tools to map the configured

sampling time required for hardware operating speed and limited in library blocks. This severely limited the system sampling rate to 5Hz per channel, which is way below the capacity of the ADC (rated at 1MSPS) and system requirements. This also limited the ability to get accurate timing from the sampling process. This time could have been used when plotting data in real time, with respect to time. The test signals were limited to a maximum bandwidth of 2Hz, this assisted in reducing measurement errors due to the limited sampling rate.

The advantage of a model-based design was the ability to develop software algorithms within the shortest time and the ability to test and verify the model using hardware in loop concept.

The DAS Analogue signal conditioning card was successfully developed, and it was used to interface the sensors signals to the ADC on the Arduino® Due platform. The test results for the DAS signal conditioning card were very similar and comparable to theoretical designs and simulations.

A beta DAS GUI was developed successfully using Matlab® GUIDE programming environment. Data was requested and received correctly from the DAS controller and stored correctly into a text file and in CSV format. The GUI was limited in terms of processing speed, as data collected become large. This affected real-time plotting of data. The results were not affect, only the plotting of data was delayed, thus the plotted data was not in real-time. The buffer memory to hold data received from DAS controller must be increased, this will assist in improving the real-time plotting of data.

The system was calibrated successfully with the transfer function of each channel established from the calibration process (limited to up-scale calibration). The overall system accuracy was 2% (for low-end measurements) and approximately 0.3% (for mid to high-end measurements) on the 5kHz channels, and with worst-case accuracy of 22% (for low-end measurements) on the two 50kHz channels. This was due to poor low-level performance on op-amps used on the 50kHz channels implementations. The system can be used successfully to acquire sensor data, provided the sensor signal bandwidth does not exceed 2Hz. This places severe limitations,

especially for use in acquiring data on a pilot mill. It should be noted that sensors with signal bandwidth exceeding 2Hz will contain more measurand information (e.g. vibration sensor).

There is a number of platforms discussed that could have been used to implement the DAS controller board. However, this might require the purchase of expensive developed tools if a model-based design is to be used. The traditional approaches for code development could have been used, which would have required C/C++ compilers. The advantage of these is that the low-level implementation of the code is under the control of the developer and will be able to control, configure ADC and other peripheral registers. The main disadvantage is that it takes time to develop low-level software drivers, and these are prone to bugs. One of the project aims was to learn new trends, concepts and technologies used in embedded system software development and implement the system using such technology (i.e. model-based software development).

8.2 Recommendations

After carrying out this study, the following suggestions are recommended, due to limited time and lack of resources the recommendations were not implemented. The recommendations may implemented following the sequence list below:

- Research should be conducted to further understand the limitations of the Arduino Due® hardware platform and Simulink® model libraries in implementing complex and high-rate data acquisition system models. This will include developing or requesting improved Simulink® library blocks and C++/C libraries, which provide access to low-level registers for processor configuration. If this can be achieved successfully, it means a very low-cost platform can be used to implement a high-end data acquisition system.
- The DAS controller should be implemented on the STM32F4 Discovery platform. This platform meets all the processor and ADC peripheral requirements (Table 3.3). This will require following the traditional embedded code development route. This may include using a real-time operating system (RTOS) for embedded systems,

specifically for ARM core processors. The RTOS, such as the FreeRTOS^{TM37}, should be used in the development of the firmware.

- The DAS system meeting the system specifications should be tested and qualified according to the IEEE Standard for Digitizing Waveform Recorders (IEEE Std 1057TM-2007, 2008) requirements.
- The quantification of the systematic and random errors must be performed, and a DAS system measurement uncertainty analysis must be carried out.
- The DAS GUI Host software must be improved in terms of processing speed, especially during real-time plotting of the acquired data. Error collection functions must be implemented. The algorithm for recording and saving data must be improved. Calibration mode must be developed and implemented on the DAS host and DAS controller.
- The DAS Analogue signal conditioning card must be enhanced to protect against reverse polarity, negative signals and overvoltage in the input stages. The PCB must be miniaturised. This will reduce the PCB costs and assist in improving noise performance. Better-performing op-amps must be identified and used. The card must be tested, qualified and characterised against environmental effects (i.e. temperature, humidity, and EMI).

³⁷ <http://www.freertos.org/>

References

- Abdallah, M., Elkeelany, O., Alouani, A.T., 2011. A Low-Cost Stand-Alone Multichannel Data Acquisition, Monitoring, and Archival System With On-Chip Signal Preprocessing. *IEEE Trans. Instrum. Meas.* 60, 2813–2827. doi:10.1109/TIM.2009.2036402
- Alatalo, J., 2011. Charge Dynamics in Tumbling Mills. Luleå University of Technology.
- Al-Eryani, J., Nitzsche, D., Sattler, S., 2011. A System-Level Model for an Analog-to-Digital Converter. 2011 IEEE 17th Int. Mix. Sensors Syst. Test Work. 100–105. doi:10.1109/IMS3TW.2011.24
- Analog Devices Inc., E., 2002. Mixed-signal and DSP Design Techniques. Newnes, USA.
- Antonyuk, M., Lobur, M., Antonyuk, V., 2007. Design Digital Data Acquisition and Processing Systems for Embedded System, in: 2007 International Conference on Perspective Technologies and Methods in MEMS Design. IEEE, pp. 54–60. doi:10.1109/MEMSTECH.2007.4283423
- Armenante, P.M., n.d. Centrifugation.
- Austerlitz, H., 2003. Data Acquisition Techniques Using PCs, 2nd ed. Academic Press, San Diego, Calif.
- Aziz, P.M., Sorensen, H.V., van der Spiegel, J., 1996. An overview of sigma-delta converters. *IEEE Signal Process. Mag.* 13, 61–84. doi:10.1109/79.482138
- Bagajewicz, M.J., Chmielewski, D.J., 2010. Smart Process Plants Software and Hardware Solutions for Accurate Data and Profitable Operations. The McGraw-Hill Companies, Inc.
- Baker, B.C., 2011. A Glossary of Analog-to-Digital Specifications and Performance Characteristics (SBAA147A). Converter 1 – 33.
- Baker, B.C., 2003. Driving the Analog Inputs of a SAR A/D Converter. USA.
- Ball, S., 2004. Analog Interfacing to Embedded Microprocessor Systems, 2nd ed. Newnes,

- USA.
- Ball, S., 2002. *Embedded Microprocessor Systems:Real World Design*, 3rd ed. Newnes, USA.
- Bansal, L.K., Patel, P.J., Prahlad, V., Qureshi, K., Patel, V.B., Gupta, L.N., Thakkar, D.P., Sumod, C.B., Vadher, V., Parmar, S., Bharathi, P., Baruah, U.K., 2013. Signal conditioning & data acquisition system for neutral beam calorimeter for NBI SST-1, in: 2013 IEEE 25th Symposium on Fusion Engineering (SOFE). IEEE, pp. 1–4. doi:10.1109/SOFE.2013.6635318
- Baofeng, Z., Ya, W., Junchao, Z., 2010. Design of high speed data acquisition system based on FPGA and DSP. 2010 Int. Conf. Artif. Intell. Educ. 132–135. doi:10.1109/ICAIE.2010.5641136
- Barrett, S.F., Pack, D.J., 2006. *Microcontrollers Fundamentals for Engineers and Scientists, Synthesis Lectures on Digital Circuits and Systems*. doi:10.2200/S00025ED1V01Y200605DCS001
- Barwicz, A., Morawski, R.Z., 1999. Teaching measuring systems beyond the year 2000. IEEE Instrum. Meas. Mag. 2, 20–27. doi:10.1109/5289.754755
- Beaty, W.H., Fink, D.G., 2013. *Standard Handbook for Electrical Engineers*, Six. ed. The McGraw-Hill Companies, Inc., USA.
- Behera, B., Mishra, B.K., Murty, C.V.R., 2007. Experimental analysis of charge dynamics in tumbling mills by vibration signature technique. Miner. Eng. 20, 84–91. doi:10.1016/j.mineng.2006.05.007
- Bich, W., 2012. From Errors to Probability Density Functions. Evolution of the Concept of Measurement Uncertainty. IEEE Trans. Instrum. Meas. 61, 2153–2159. doi:10.1109/TIM.2012.2193696
- Bolton, W., 1992. *Electrical and Electronic Measurement and Testing*. Longman Scientific & Technical, England.
- Bordoni, F., D'amico, A., 1990. Noise in Sensors. Sensors and Actuators, 17–24. doi:0924-

4241/90/\$3 50

- Bowling, S., 2002. *Understanding A/D Converter Performance Specifications*. USA.
- Broesch, J.D., 2009. The Analog-Digital Interface 7–25. doi:10.1016/B978-0-7506-8976-2.00002-X
- Broesch, J.D., 2008. *Digital Signal Processing: Instant Access*. Newnes, San Diego, USA.
- Broesch, J.D., 1997. *Digital Signal Processing Demystified*. LLH Technology, Eagle Rock, Virginia, USA.
- Broggi, F., Paccalini, a., Rivoltella, G., 2006. Data Acquisition System for Large Superconducting Magnets. *IEEE Trans. Appl. Supercond.* 16, 545–548. doi:10.1109/TASC.2006.870835
- Cantley, K.D., Fernandes, P.G., Zhao, M., Stiegler, H.J., Chapman, R.A., Vogel, E.M., 2012. Noise effects in field-effect transistor biological sensor detection circuits, in: 2012 IEEE 55th International Midwest Symposium on Circuits and Systems (MWSCAS). IEEE, pp. 370–373. doi:10.1109/MWSCAS.2012.6292034
- Carter, B., Mancini, R., 2009. *Op Amps for Everyone*, 3rd ed. Newnes, MA, USA.
- Catunda, S.Y.C., Deep, G.S., Freire, R.C.S., Naviner, J.-F., 2002. Programming an analog signal conditioning circuit without loss in measurement range, in: *IMTC/2002. Proceedings of the 19th IEEE Instrumentation and Measurement Technology Conference (IEEE Cat. No.00CH37276)*. IEEE, pp. 529–534. doi:10.1109/IMTC.2002.1006898
- Chaipurimas, K., Rerktratn, A., Cheypoca, T., Riewruja, V., 2010. 4–20mA current transceiver, in: *International Conference on Control, Automation and Systems 2010*. IEEE, pp. 1631 – 1634. doi:978-89-93215-02-1
- Chen, W., 2004. *The Electrical Engineering Handbook*. Academic Press, MA, USA.
- Christiansen, D., Alexander, C.K., Jurgen, R.K., 2005a. *Standard Handbook of Electronic Engineering*, Fifth Edition, 5th ed. McGraw- Hill, New York.
- Christiansen, D., Alexander, C.K., Jurgen, R.K., 2005b. *Standard Handbook of Electronic*

- Engineering, 5th ed. The McGraw-Hill Companies, Inc., New York.
- Clayton, G.B., Winder, S., 2003. Operational Amplifiers, 5th ed. Newnes, Great Britain.
- Coombs Jr, C.F., 2000. Electronic Instrument Handbook, 3rd ed. McGraw-Hill, New York.
- Cooper, W.D., Helfrick, A.D., 1985. Electronic Instrumentation and Measurement Techniques. Prentice Hall, New Jersey.
- Curtis, K.E., 2006. Embedded Multitasking. Newnes, MA, USA.
- Cvetkovic, Z., Vetterli, M., 1998. Error-rate characteristics of oversampled analog-to-digital conversion. *IEEE Trans. Inf. Theory* 44, 1961–1964. doi:10.1109/18.705574
- D’Amico, A., Di Natale, C., 2001. A contribution on some basic definitions of sensors properties. *IEEE Sens. J.* 1, 183–190. doi:10.1109/JSEN.2001.954831
- D’Apuzzo, M., Liguori, C., 1999. Electric Sensing Devices, in: *Wiley Encyclopedia of Electrical and Electronics Engineering*. John Wiley & Sons, Inc. doi:10.1002/047134608X.W4005
- Davis, J., 2006. High-Speed Digital System Design. *Synth. Lect. Digit. Circuits Syst.* 1, 1–96. doi:10.2200/S00044ED1V01Y200609DCS005
- de Barros Soldera, J.D., Saldana, J.C., Penteado, C.G., Hernandez, H.D., Acosta, R., Porras, F.C., Valerio, M.A., dos Anjos, A., Trevisan, P.H., 2012. On-chip 4to20mA reconfigurable current loop transmitter for smart sensor applications, in: 2012 25th Symposium on Integrated Circuits and Systems Design (SBCCI). IEEE, pp. 1–6. doi:10.1109/SBCCI.2012.6344434
- Dempster, J., 2001. Signal Conditioning 74–100. doi:10.1016/B978-012209551-1/50037-4
- DeNatale, J., Borwick, R., Stupar, P., Anderson, R., Garrett, K., Morris, W., Yao, J.J., 2003. MEMS high resolution 4-20 mA current sensor for industrial I/O applications, in: *TRANSDUCERS ’03. 12th International Conference on Solid-State Sensors, Actuators and Microsystems. Digest of Technical Papers (Cat. No.03TH8664)*. IEEE, pp. 1598–1601. doi:10.1109/SENSOR.2003.1217086

- Dias Pereira, J.M., Postolache, O., Silva Girao, P.M.B., 2007. A Digitally Programmable A/D Converter for Smart Sensors Applications. *IEEE Trans. Instrum. Meas.* 56, 158–163. doi:10.1109/TIM.2006.887771
- Dictionary, F., 2014. Active [WWW Document]. URL <http://www.thefreedictionary.com/active> (accessed 1.17.15).
- Doboli, A., Currie, E.H., 2011. *Introduction to Mixed-Signal, Embedded Design*. Springer New York, New York, NY. doi:10.1007/978-1-4419-7446-4
- Dos Reis Filho, C.A., 1989. An integrated 4-20 mA two-wire transmitter with intrinsic temperature sensing capability. *IEEE J. Solid-State Circuits* 24, 1136–1142. doi:10.1109/4.34102
- Dudojc, B., 2008. Proprieties of transmitters in function of dynamic changes of operation point of two wires measurement line in 4-20mA standard, in: 16th IMEKO TC4 Int. Symp.: Exploring New Frontiers of Instrum. and Methods for Electrical and Electronic Measurements; 13th TC21 Int. Workshop on ADC Modelling and Testing - Joint Session, Proc. pp. 573–577.
- Dunn, P.F., 2014. *Measurement and Data Analysis for Engineering and Science*, 3rd ed. CRC Press, Indiana.
- Ebookaktiv, D., 2013. *Model Based Software Development with Simulink and Arduino*.
- Eccles, W., 2011. Pragmatic Electrical Engineering: Systems and Instruments. *Synth. Lect. Digit. Circuits Syst.* 6, 1–143. doi:10.2200/S00353ED1V01Y201105DCS034
- Electronics, T., 2004. Type RN73 Series High Precision Resistors.
- Engelberg, S., 2007. Sigma-Delta Converters: Theory and Simulations. *IEEE Instrum. Meas. Mag.* 10, 49–53. doi:10.1109/MIM.2007.4428582
- Felinger, A., Kilár, A., Boros, B., 2015. The myth of data acquisition rate. *Anal. Chim. Acta* 854, 178–182. doi:10.1016/j.aca.2014.11.014
- Ferrero Martín, F.J., Valledor Llopis, M., Campo Rodríguez, J.C., Blanco González, J.R., Menéndez Blanco, J., 2014. Low-cost open-source multifunction data acquisition system

- for accurate measurements. *Meas. J. Int. Meas. Confed.* 55, 265–271.
doi:10.1016/j.measurement.2014.05.010
- Feucht, D., 2010. *Designing High-Performance Amplifiers (Analog Circuit Design Series: Volume 3)*. Institution of Engineering and Technology. doi:10.1049/SBCS015E
- Finkelstein, P.L., 2014. *Instrument Science [WWW Document]*. AccessScience (McGraw-Hill Educ. 2014). URL <http://www.accessscience.com/content/instrument-science/346850> (accessed 12.15.14).
- Fischer-Cripps, A.C., 2002. *Newnes Interfacing Companion*. Newnes, Oxford.
- Floyd, T.L., 1999. *Electronic Devices, 5th ed.* Prentice Hall, New Jersey.
- Fowler, K., 2006. Tried and true - Selecting an ADC. *IEEE Instrum. Meas. Mag.* 9, 62–65.
doi:10.1109/MIM.2006.1634993
- Fowler, K., 2003. Part 7: analog-to-digital conversion in real-time systems. *IEEE Instrum. Meas. Mag.* 6, 58–64. doi:10.1109/MIM.2003.1238355
- Fowler, K., 2001. Giving meaning to measurement. *IEEE Instrum. Meas. Mag.* 4, 41–45.
doi:10.1109/5289.953458
- Fowler, K.R., 1996. *Electronic Instrument Design: Architecting for the Life Cycle*. Oxford University Press, New York.
- Fowler, K.R., Schmalzel, J.L., 2004. Sensors: the first stage in the measurement chain. 2. In a series of tutorials in instrumentation and measurement. *IEEE Instrum. Meas. Mag.* 7, 60–66. doi:10.1109/MIM.2004.1337915
- Fraden, J., 2010. *Handbook of Modern Sensors Physics, Designs, and Applications, 4th ed.* Springer New York, New York.
- Gao, F., 2000. *Firmware development in a data acquisition system*. Texas A&M University.
- Gaydecki, P., 2004. *Foundations of Digital Signal Processing: theory, algorithms and hardware design*. IET, The Institution of Engineering and Technology, Michael Faraday House, Six Hills Way, Stevenage SG1 2AY, UK. doi:10.1049/PBCS015E

- Gill, G., Hexter, P., 1973. Some Instrumentation Definitions for Use by Meteorologists and Engineers. *IEEE Trans. Geosci. Electron.* 11, 83–89. doi:10.1109/TGE.1973.294291
- Glisson, T.H., 2011. *Introduction to Circuit Analysis and Design*. Springer Netherlands, Dordrecht. doi:10.1007/978-90-481-9443-8
- Goldie, D.J., Audley, M.D., Glowacka, D.M., Tsaneva, V.N., Withington, S., 2009. Thermal models and noise in transition edge sensors. *J. Appl. Phys.* 105, 074512. doi:10.1063/1.3097396
- Gray, N., 2006. *ABCs of ADCs Analog-to-Digital Converter Basics*. *Natl. Semicond. Signal Sound Inf.* 64.
- Gray, R.M., 1990. Quantization noise spectra. *IEEE Trans. Inf. Theory* 36, 1220–1244. doi:10.1109/18.59924
- Grout, I., 2008. *Digital Systems Design with FPGAs and CPLDs*. Newnes, Ireland.
- Guan, K., 2012. *Opportunistic Sampling By Level-Crossing*. University of Illinois.
- Gupta, R., Mitra, M., Bera, J., 2009. Development of a State-of-the-Art ECG DAS for Storing, Processing and Analysis Using MATLAB-Based GUI and Microprocessor, in: 2009 International Conference on Advances in Computing, Control, and Telecommunication Technologies. IEEE, pp. 570–572. doi:10.1109/ACT.2009.144
- Hamid, U., Qamar, R.A., Shahzad, M., 2013. PC based data acquisition and signal processing for underwater sensor arrays, in: *Proceedings of 2013 10th International Bhurban Conference on Applied Sciences & Technology (IBCAST)*. Ieee, pp. 321–327. doi:10.1109/IBCAST.2013.6512172
- Harnett, C., 2011. Open source hardware for instrumentation and measurement. *IEEE Instrum. Meas. Mag.* 14, 34–38. doi:10.1109/MIM.2011.5773535
- Harrison, R.G., Nicoll, K. a, Lomas, a G., 2012. Note: Programmable data acquisition system for research measurements from meteorological radiosondes. *Rev. Sci. Instrum.* 83, 036106. doi:10.1063/1.3697717
- Heath, S., 2002. *Embedded Systems Design*, 2nd ed. Newnes, Burlington, MA.

- Herniter, M.E., 2014. Instrumentation and Microcontrollers using Automatic Code Generation (Arduino Mega Version), Rose-Hulman Institute of Technology.
- Hosticka, B.J., 2007. Analog circuits for sensors, in: ESSCIRC 2007 - 33rd European Solid-State Circuits Conference. Ieee, pp. 97–102. doi:10.1109/ESSCIRC.2007.4430255
- Huijsing, J., 2011. Operational Amplifiers. Springer Netherlands, Dordrecht. doi:10.1007/978-94-007-0596-8
- Ida, N., 2013. Sensors, Actuators, and their Interfaces: A Multidisciplinary Introduction. Institution of Engineering and Technology. doi:10.1049/SBCS502E
- IEEE Std 1057TM-2007, 2008. IEEE Std 1057TM-2007, IEEE Standard for Digitizing Waveform Recorders.
- IEEE Std 1241, 2011. IEEE Standard for Terminology and Test Methods for Analog-to-Digital Converters. IEEE Std 1241-2010 (Revision IEEE Std 1241-2000 139. doi:10.1109/IEEESTD.2011.5692956
- Ifeachor, C.E., Jervis, W.B., 2002. Digital Signal Processing: A Practical Approach, 2nd ed. Pearson Education, USA.
- Instruments, T., 2014. TL08xx JFET-Input Operational Amplifiers.
- Instruments, T., 2010. THS4032 100-MHz Low-Noise High-Speed Amplifiers.
- Integrated, M., 2002. The ABCs of ADCs: Understanding How ADC Errors Affect System Performance [WWW Document]. A/D D/A Conversion/Sampling Circuits. URL <http://www.maximintegrated.com/en/app-notes/index.mvp/id/748> (accessed 9.20.14).
- International Society of Automation (ISA), 2012. ANSI/ISA-50.00.01-1975 (R2012) Compatibility of Analog Signals for Electronic Industrial Process Instruments. AMERICAN NATIONAL STANDARD. doi:978-1-937560-54-6
- James, K., 2000. PC Interfacing and Data Acquisition Techniques for Measurement, Instrumentation and Control. Newnes, Boston.
- Jaspan, R.K., Young, G.J.C., Mellor, M.S., 1986. ROM mill power draft control using

- multiple microphones to determine mill load, in: International Conference on Gold. SAIMM, Johannesburg, pp. 483–392.
- Jenq, Y.-C., 2003. A/D converters with midpoint correction, in: Proceedings of the 20th IEEE Instrumentation Technology Conference (Cat. No.03CH37412). IEEE, pp. 1203–1205. doi:10.1109/IMTC.2003.1207943
- Johnson, C.D., 2000. Process Control Instrumentation Technology, 6th ed. Prentice Hall, New Jersey.
- Joint Committee for Guides in Metrology (JCGM/WG 2), 2012. Basic and general concepts and associated terms (VIM).
- Jones, N.B., Watson, J.D.M. (Eds.), 1990. Digital Signal Processing: principles, devices and applications. IET, The Institution of Engineering and Technology, Michael Faraday House, Six Hills Way, Stevenage SG1 2AY, UK. doi:10.1049/PBCE042E
- Jung, W., 2005. Op Amp Applications Handbook. Newnes, USA.
- Kamrul Islam, S., Haider, M.R., 2010. Sensors and Low Power Signal Processing. Springer US, Boston, MA. doi:10.1007/978-0-387-79392-4
- Kehtarnavaz, N., 2008. Digital Signal Processing System Design: LabVIEW-Based Hybrid Programming, 2nd ed. Academic Press, USA.
- Kester, B.W., 2006. ADC Input Noise : The Good , The Bad , and The Ugly . Is No Noise Good Noise ? Reading 40, 1–5.
- Kester, W., 2005a. Data Conversion Handbook. Newnes, Amsterdam, Boston.
- Kester, W., 2005b. Instrumentation Amplifiers Op Amp ~ In Amp Functionality Differences 123–149.
- Kiangi, K.K., Moys, M.H., 2006. Measurement of the load behaviour in a dry pilot mill using an inductive proximity probe. Miner. Eng. 19, 1348–1356. doi:10.1016/j.mineng.2006.01.007
- Kim, Y.J., Kunkler, B., Liu, C.-Y., Visser, G., 2012. A high dynamic range data acquisition

- system for a solid-state electron electric dipole moment experiment. *Rev. Sci. Instrum.* 83, 013502. doi:10.1063/1.3676163
- Kolacz, J., 1998. Control of the Mill Charge Behavior in Dry Tumbling Mills. *Miner. Eng.* 12, 51–64. doi:0892-6875(98)00119-8
- Kolacz, J., 1997. Measurement System of the Mill Charge in Grinding Ball Mill Circuits. *Miner. Eng.* 10, 1329–1338. doi:0892--6875(97)00124-6
- Kook, S.H., 2011. *Low-Cost Testing of High-Precision Analog-to-Digital Converters.* Georgia Institute of Technology.
- Kularatna, N., 2003a. *Digital and analogue instrumentation : testing and measurement.* Institution of Electrical Engineers, London.
- Kularatna, N., 2003b. *Digital and Analogue Instrumentation: testing and measurement.* IET, The Institution of Engineering and Technology, Michael Faraday House, Six Hills Way, Stevenage SG1 2AY, UK. doi:10.1049/PBEL011E
- Kularatna, N., 2000. *Modern Component Families and Circuit Block Design.* Newnes, New Zealand.
- Lanstiak, B., 1973. Method for measurement of ball-mill loading., in: *Tenth International Mineral Processing Congress.* Institution of Mining and Metallurgy, London, pp. 63–71.
- Leach, M. J., 1999. Circuit Noise, in: *Wiley Encyclopedia of Electrical and Electronics Engineering.* John Wiley & Sons, Inc. doi:10.1002/047134608X.W2220
- Leis, J.W., 2011. *Digital Signal Processing Using MATLAB for Students and Researchers.* John Wiley & Sons, Hoboken, NJ.
- Li, T., Liu, Y., Sun, L., 2009. Effect of Sensor Resolution on the Accuracy of Positioning Stage, in: *2009 International Conference on Measuring Technology and Mechatronics Automation.* IEEE, pp. 89–92. doi:10.1109/ICMTMA.2009.580
- Liddell, K., Moys, M.H., 1988. The effects of mill speed and filling on the behaviour of the load in a rotary grinding mill. *J. South African Inst. Min. Metall.* 88.

- Lita, I., Sofron, E., Serban, G., Visan, D., 2001. A method of noise reduction in data acquisition systems from sensors, in: 24th International Spring Seminar on Electronics Technology. Concurrent Engineering in Electronic Packaging. ISSE 2001. Conference Proceedings (Cat. No.01EX492). IEEE, pp. 253–257. doi:10.1109/ISSE.2001.931073
- Lohiya, G.B., Talbar, S.N., 1998. Design of profibus compatible smart transmitter, in: Proceedings of IEEE TENCON '98. IEEE Region 10 International Conference on Global Connectivity in Energy, Computer, Communication and Control (Cat. No.98CH36229). IEEE, pp. 365–368. doi:10.1109/TENCON.1998.798161
- Lowdermilk, R.W., Harris, F.J., 2003. Signal conditioning and data collection in synthetic instruments [ATE systems], in: Proceedings AUTOTESTCON 2003. IEEE Systems Readiness Technology Conference. IEEE, pp. 426–431. doi:10.1109/AUTEST.2003.1243608
- MathWorks, 2014a. Simulink® Reference. MathWorks.
- MathWorks, 2014b. DSP System Toolbox™ Reference.
- McGhee, J., Henderson, I.A., Sydenham, P.H., 1999. Sensor science – essentials for instrumentation and measurement technology. *Measurement* 25, 89–113.
- McGrath, M., Ni Scanail, C., 2013a. Sensing and Sensor Fundamentals. Intel 14.
- McGrath, M., Ni Scanail, C., 2013b. Sensor Characteristics Demystified 7.
- McMillan, G.K., 1999. *Process/Industrial Instruments and Controls Handbook*, 5th ed. The McGraw-Hill Companies, Inc., USA.
- Meddins, R., 2000. *Introduction to Digital Signal Processing*. Newnes, Great Britain.
- Meyer, T., Johanson, R.E., Kasap, S., 2011. Effect of 1/f noise in integrating sensors and detectors. *IET Circuits, Devices Syst.* 5, 177. doi:10.1049/iet-cds.2010.0220
- Michalski, A., Makowski, L., 2012. Measurement systems as foundations for reliable decision support systems [Instrumentation Notes]. *IEEE Instrum. Meas. Mag.* 15, 48–52. doi:10.1109/MIM.2012.6204874

- Mishra, D.K., Gamad, R.S., 2011. Dynamic testing of an ADC for real application input, in: 2011 IEEE AUTOTESTCON. IEEE, pp. 354–359. doi:10.1109/AUTEST.2011.6058732
- Montini, A., Moys, M.H., 1988. The measurement of rheological properties inside a grinding mill. *J. South African Inst. Min. Metall.* 88.
- Morris, A.S., 2001. *Measurement and Instrumentation Principles*. Butterworth Heinemann, USA.
- Morris, A.S., Langari, R., 2012. *Measurement and instrumentation : theory and application*, US Patent 4,039,260. Academic Press, USA. doi:10.1016/B978-0-12-381960-4.00020-6
- Moshayedi, A.J., Toudeshki, A., Gharpure, D.C., 2013. Mathematical modeling for SnO₂ gas sensor based on second-order response, in: 2013 IEEE Symposium on Industrial Electronics & Applications. IEEE, pp. 33–38. doi:10.1109/ISIEA.2013.6738963
- Moys, M.H., 1984. The measurement of parameters describing the dynamic behaviour of the load in a grinding mill., in: *MINTEK 50*,. Sandton.
- Moys, M.H., Skorupa, J., 1992. Measurement of the radial and tangential forces exerted by the load on a liner in a ball mill, as a function of load volume and mill speed. 37.
- Moys, M.H., Smit, I., Stange, W., 1996a. The measurement of forces exerted by the load on liners in rotary mills (wet and dry). *Miner. Process.* 44, 383–393. doi:0301-7516(95)00034-8
- Moys, M.H., Van Nierop, M.A., Smit, I., 1996b. Progress in Measuring and Modelling Load Behaviour in Pilot and Industrial Mills. *Miner. Eng.* 9.
- Mulenga, F., Moys, M., 2013. Effects of slurry filling and mill speed on the net power draw of a tumbling ball mill. *Miner. Eng.* 56, 45–56.
- Najarian, S., Javad Dargahi, P.D., Darbemamieh, G., Farkoush, S.H., 2004. Processing and Control Systems, in: *Mechatronics in Medicine: A Biomedical Engineering Approach*. McGraw Hill Professional, Access Engineering, USA.
- Niu, T., 2012. A research on the design of smart pressure transmitter, in: 2012 International Conference on Computer Science and Information Processing (CSIP). IEEE, pp. 803–

806. doi:10.1109/CSIP.2012.6308975
- Norsworthy, S.R., Schreier, R., Temes, G.C., 1997. Quantization Errors and Dithering in Sigma Delta Modulators, in: Delta-Sigma Data Converters: Theory, Design, and Simulation. IEEE Press, New York, USA, p. 476.
- Nuccio, S., Spataro, C., 2002. Approaches to evaluate the virtual instrumentation measurement uncertainties. *IEEE Trans. Instrum. Meas.* 51, 1347–1352.
doi:10.1109/TIM.2002.808036
- O’Shea, P., Sadik, A.Z., Hussain, Z.M., 2011. Digital Signal Processing. Springer Berlin Heidelberg, Berlin, Heidelberg. doi:10.1007/978-3-642-15591-8
- Ogata, K., 2002. Modern Control Engineering, 4th ed. Prentice Hall, New Jersey.
- Oshana, R., 2012a. DSP for Embedded and Real-Time Systems. Newnes.
- Oshana, R., 2012b. DSP for Embedded and Real-Time Systems, Expert Guide DSP for Embedded and Real-Time Systems. Elsevier, USA. doi:10.1016/B978-0-12-386535-9.00001-9
- Oshana, R., 2007. Overview of digital signal processing algorithms, part II - Part 10 in a series of tutorials in instrumentation and measurement. *IEEE Instrum. Meas. Mag.* 10, 53–58. doi:10.1109/MIM.2007.364962
- Park, J., Mackay, S., 2003. Practical data acquisition for instrumentation and control systems. Newnes, Oxford.
- Pease, R., 2008. Analog Circuits World Class Designs. Newnes, USA.
- Pelgrom, M.J.M., 2010. Analog-to-Digital Conversion. Publisher, Netherlands.
doi:10.1007/978-90-481-8888-8
- Pereira, J.M.D., 2004. A fieldbus prototype for educational purposes. *IEEE Instrum. Meas. Mag.* 7, 24–31. doi:10.1109/MIM.2004.1288735
- Perez, R., 2002. Design of Medical Electronic Devices. Academic Press, USA.
- Peterson, A.F., Durgin, G.D., 2008. Transient Signals on Transmission Lines: An Introduction

- to Non-Ideal Effects and Signal Integrity Issues in Electrical Systems. Synth. Lect. Comput. Electromagn. 3, 1–144. doi:10.2200/S00155ED1V01Y200812CEM024
- Phillips, L.C., Nagle, H.T., 1995. Digital Control System Analysis and Design, 3rd ed. Prentice Hall, USA.
- Pottie, G.J., Kaiser, W.J., 2005. Principles of Embedded Networked Systems Design. Cambridge University Press, Cambridge. doi:10.1017/CBO9780511541049
- Power, T., 2014. THL 25WI Series DC/DC Converters.
- Qaisar, S.M., Yahiaoui, R., Gharbi, T., 2013. An efficient signal acquisition with an adaptive rate A/D conversion, in: 2013 IEEE International Conference on Circuits and Systems (ICCAS). IEEE, pp. 124–129. doi:10.1109/CircuitsAndSystems.2013.6671611
- Rabinovich, S.G., 2010. Evaluating measurement accuracy: A practical approach, Evaluating Measurement Accuracy: A Practical Approach. doi:10.1007/978-1-4419-1456-9
- Rapuano, S., Daponte, P., Balestrieri, E., De Vito, L., Tilden, S.J., Max, S., Blair, J., 2005. ADC parameters and characteristics. IEEE Instrum. Meas. Mag. 8, 44–54. doi:10.1109/MIM.2005.1578617
- Rauth, D.A., Randal, V.T., 2005. Analog-to-digital conversion. part 5. IEEE Instrum. Meas. Mag. 8, 44–54. doi:10.1109/MIM.2005.1518622
- Regtien, P., 2012. Sensors for Mechatronics. Elsevier. doi:10.1016/B978-075066379-3/50009-4
- Regtien, P., 2004. Measurement Errors and Uncertainty 43–85. doi:10.1016/B978-190399658-4/50004-6
- Regtien, P., van der Heijden, F., Korsten, M., Olthius, W., 2004. Measurement Science for Engineers. Butterworth Heinemann, London.
- Regtien, P.P.L., 1992. Instrumentation electronics. Prentice Hall, New York.
- Rieger, R., Demosthenous, A., 2008. A DC coupled signal acquisition system with ultra-wide input range, in: 2008 IEEE International Symposium on Circuits and Systems. IEEE, pp.

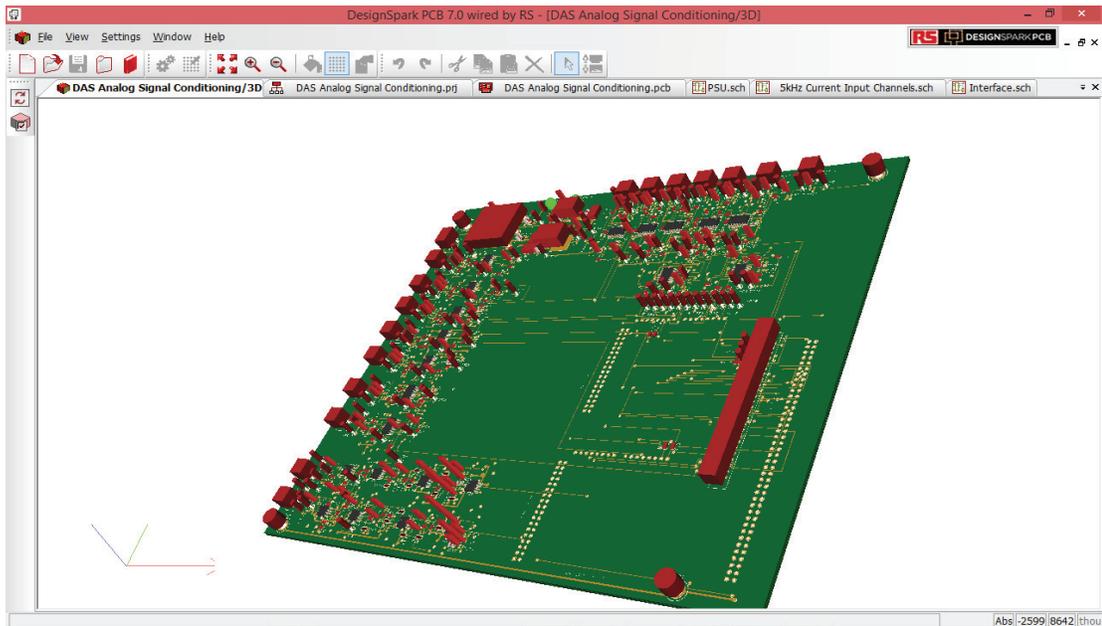
- 2729–2732. doi:10.1109/ISCAS.2008.4542021
- Robert Pease (Ed.), 2008. *Analog Circuits*. Newnes, USA.
- Russell, L., Steele, A.L., Goubran, R., 2012. Low-cost, rapid prototyping of IMU and pressure monitoring system using an open source hardware design, in: 2012 IEEE International Instrumentation and Measurement Technology Conference Proceedings. IEEE, pp. 2695–2699. doi:10.1109/I2MTC.2012.6229719
- Salicone, S., 2013. The theory of evidence: a new promising approach to the evaluation and expression of measurement uncertainty. *IEEE Instrum. Meas. Mag.* 16, 18–23. doi:10.1109/MIM.2013.6417052
- Schmalzel, J.L., Rauth, D.A., 2005. Sensors and signal conditioning. *IEEE Instrum. Meas. Mag.* 8, 48–53. doi:10.1109/MIM.2005.1438844
- Sclater, N., 1999. *Electronic Technology Handbook*. The McGraw-Hill Companies, Inc, New York.
- Sen, S., Bhaumik, A.K., 2013. Design of Intelligent Control System Using Acoustic Parameters for Grinding Mill Operation, in: *Computer Science & Information Technology (CS & IT)*. Academy & Industry Research Collaboration Center (AIRCC), pp. 261–268. doi:10.5121/csit.2013.3224
- Si, G., Cao, H., Zhang, Y., Jia, L., 2009. Experimental investigation of load behaviour of an industrial scale tumbling mill using noise and vibration signature techniques. *Miner. Eng.* 22, 1289–1298. doi:10.1016/j.mineng.2009.07.010
- Sinclair, I., 2011. *Electronics Simplified*, 3rd ed. Newnes, Great Britain.
- Sinclair, I.R., 2001. *Sensors and Transducers*, 3rd ed. Newnes, Great Britain.
- Siskind, E.J., 2007. Data acquisition system issues for large experiments. *Nucl. Instruments Methods Phys. Res. Sect. A Accel. Spectrometers, Detect. Assoc. Equip.* 579, 839–843. doi:10.1016/j.nima.2007.05.303
- Smit, I., 2000. *The effect of slurry viscosity and mill speed on the behaviour of a rotary grinding mill*. University of the Witwatersrand.

- Sobot, R., 2012. *Wireless Communication Electronics*. Springer US, Boston, MA.
doi:10.1007/978-1-4614-1117-8
- Soloman, S., 2010. *Sensors Handbook*, 2nd ed. The McGraw-Hill Companies, Inc., USA.
- Stringham, G., 2009. *Hardware/Firmware Interface Design Best Practices for Improving Embedded Systems Development*. Newnes, USA.
- Suranthiran, S., Jayasuriya, S., 2003. Signal recovery and noise removal with memoryless sensors, in: *Proceedings of the 2003 American Control Conference*, 2003. IEEE, pp. 4155–4160. doi:10.1109/ACC.2003.1240487
- Tang, J., Zhao, L., Zhou, J., Yue, H., Chai, T., 2010. Experimental analysis of wet mill load based on vibration signals of laboratory-scale ball mill shell. *Miner. Eng.* 23, 720–730. doi:10.1016/j.mineng.2010.05.001
- Tano, K., 2005. *Continuous Monitoring of Mineral Processes with Special Focus on Tumbling Mills – A Multivariate Approach*. Luleå University of Technology. doi:1402-1544
- Terrell, D., 1996. *Op Amps: Design, Application, and Troubleshooting*. Newnes, USA.
- Thompson, M., 2006. *Intuitive Analog Circuit Design*. Newnes, USA.
- TI, 2011. *FilterPro™ User's Guide*.
- Van Nierop, M.A., Moys, M.H., 1998. Premature Centrifuging, Oscillation and Axial Mixing of an Industrial Grinding Mill Load. *Miner. Eng.* 11.
- Vermeulen, L., Ohlson De Fine, M.J., Schakowski, F., 1984. Physical information from the inside of a rotary mill. *J. South African Inst. Min. Metall.* 84.
- Walls, C., 2012. *Embedded Software The Works*, 2nd ed. Newnes, USA.
- Wang, M., 2010. *Understandable Electric Circuits*. IET, The Institution of Engineering and Technology, Michael Faraday House, Six Hills Way, Stevenage SG1 2AY, UK.
doi:10.1049/PBCS023E
- Wang, W., Zhang, D., Zhang, Z., 2010. Control, measurement and data acquisition systems of

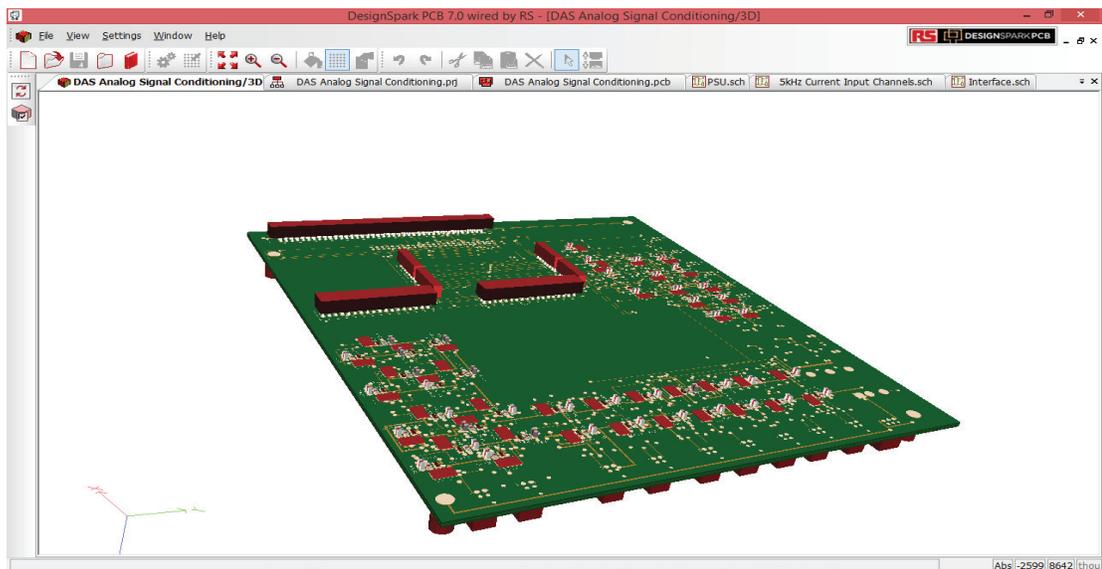
- the fretting apparatus, in: 2010 International Conference on Mechanic Automation and Control Engineering. IEEE, pp. 2680–2683. doi:10.1109/MACE.2010.5535405
- Watson, J., 1985. An analysis of mill grinding noise. *Powder Technol.* 41.
- Wilson, J.S., 2005. *Sensor Technology Handbook*. Newnes, Amsterdam, Boston.
- Wolf, M., 2012. *Computers as Components: Principles of Embedded Computing System Design*, 3rd ed. Morgan Kaufmann, USA.
- Xue, D., Chen, Y., 2013. *System Simulation Techniques with MATLAB and Simulink*. John Wiley & Sons, Ltd, United Kingdom.
- Zeng, Y., Forssberg, E., 1994. Monitoring Grinding Parameters by Vibration Signal Measurement - A Primary Application. *Miner. Eng.* 7, 495–501. doi:0892-6875(93)E0035-V
- Zeng, Y., Forssberg, E., 1992. Effects of operating parameters on vibration signal under laboratory scale ball grinding conditions. *Int. J. Miner. Process.* 273–290. doi:0301-7516/92/
- Zhang, M., Yin, Y., Deng, H., Chen, H., 2012. Design of low-jitter clock duty cycle stabilizer in high-performance pipelined ADC, in: *Anti-Counterfeiting, Security, and Identification*. IEEE, pp. 1–5. doi:10.1109/ICASID.2012.6325310
- Zhang, Y., Sun, G., Yang, Y., 2012. 12-Lead ECG Data Acquisition System Based on ADS1298. *Procedia Eng.* 29, 2103–2108. doi:10.1016/j.proeng.2012.01.270
- Zhu, J., Dutt, N., 2009. *Electronic Design Automation, Electronic Design Automation: Synthesis, Verification, and Test*. Elsevier. doi:10.1016/B978-0-12-374364-0.50012-6
- Zumbahlen, H., 2008. *Linear Circuit Design Handbook*. Newnes, USA.

Appendix B Signal Conditioning Card 3D Views

Signal Conditioning Card 3D Top View

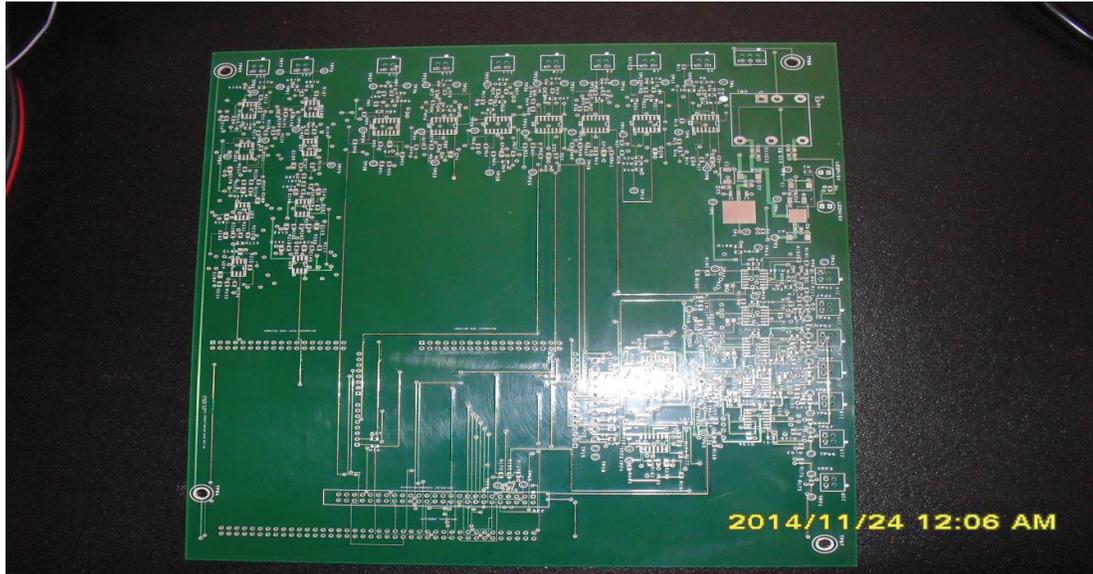


Signal Conditioning Card 3D Bottom View

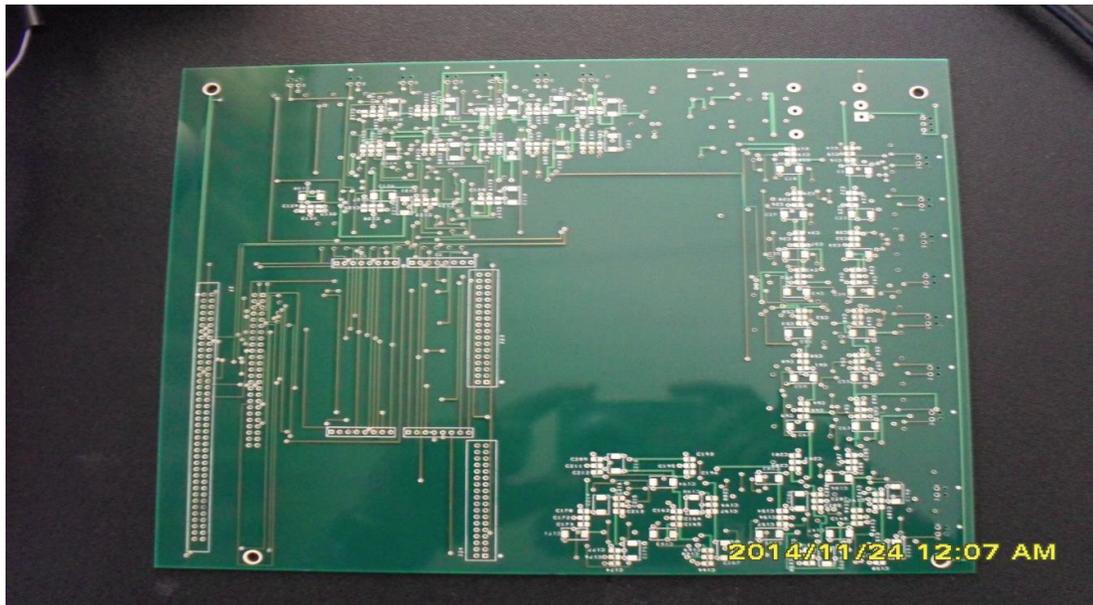


Appendix C Bare Signal Conditioning PCB

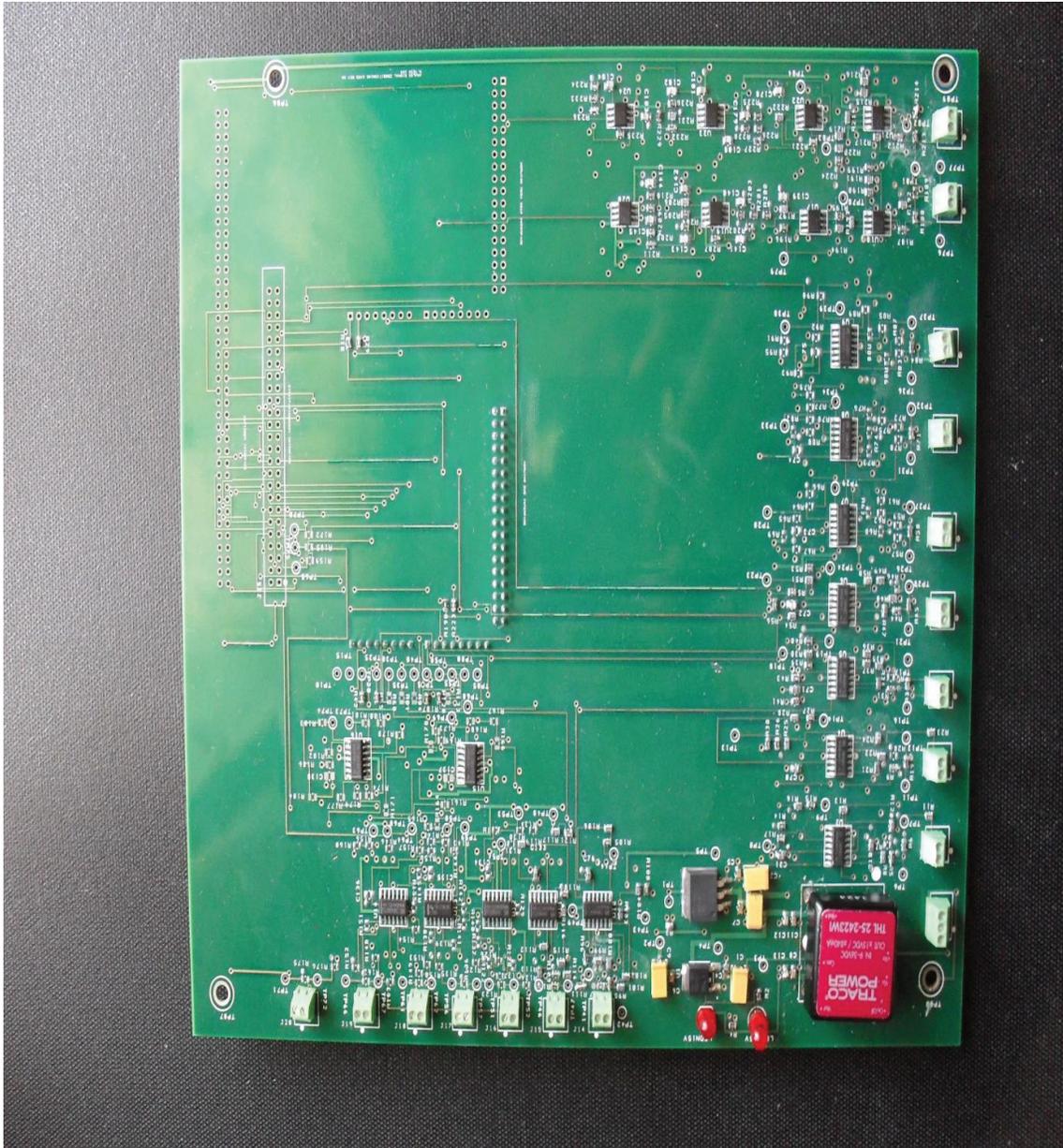
Bare Signal Conditioning PCB Top Side



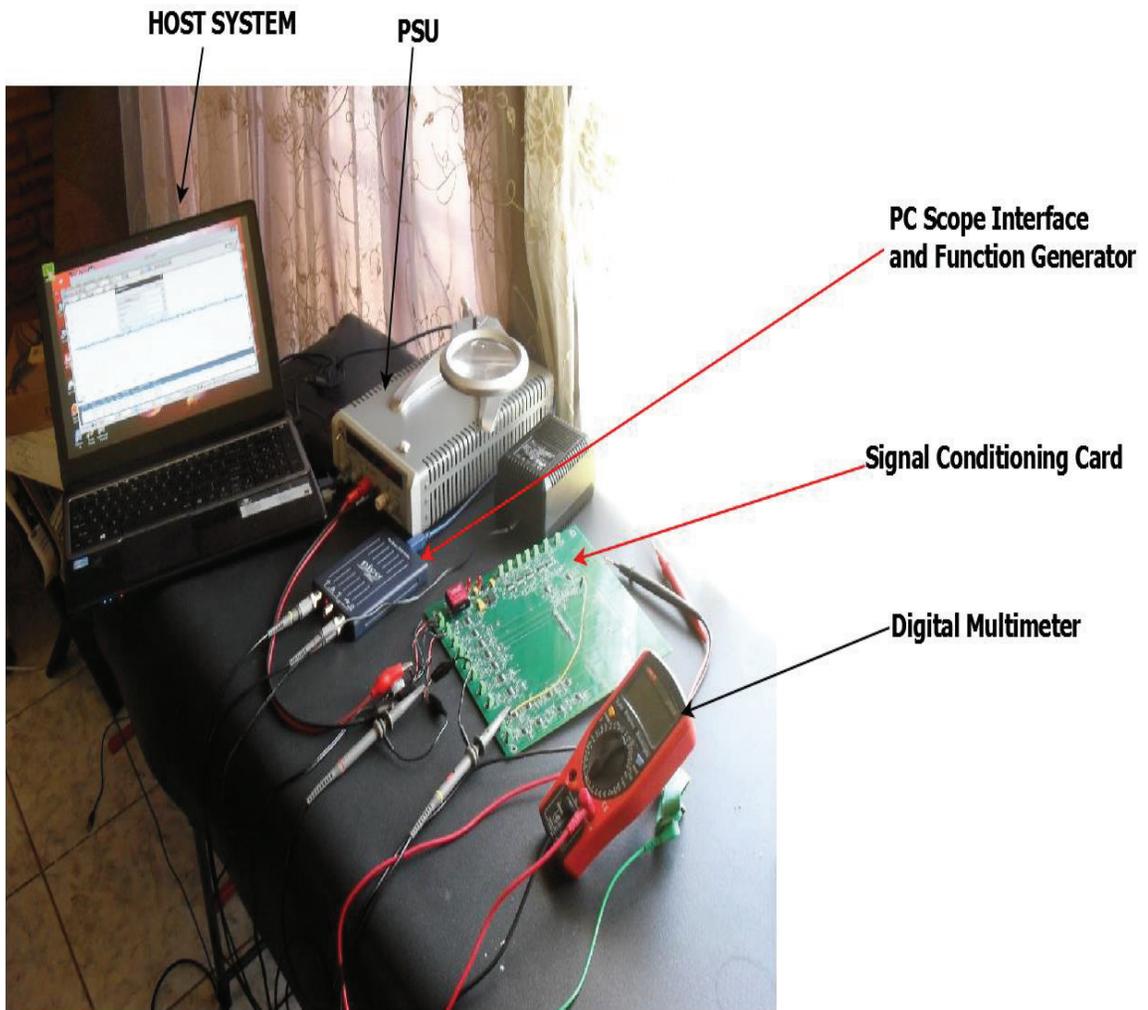
Bare Signal Conditioning PCB Bottom Side



Appendix D Assembled Signal Conditioning Card



Appendix E Signal Conditioning Card Test Set-up



Appendix F Control Signals Subsystem Matlab Code

Control Signals Subsystem Code

```
function [En_System, Ch0_En, Ch1_En, Ch2_En, Ch3_En, Ch4_En, Ch5_En,
Ch6_En, Ch7_En, Ch8_En, Ch9_En, Ch10_En, Ch11_En] = System_Control(Data)
%#reset the control signals
%*****
Ch0_En = 0;
Ch1_En = 0;
Ch2_En = 0;
Ch3_En = 0;
Ch4_En = 0;
Ch5_En = 0;
Ch6_En = 0;
Ch7_En = 0;
Ch8_En = 0;
Ch9_En = 0;
Ch10_En = 0;
Ch11_En = 0;
En_System = 0;
%*****
Txt = Data;
%process the commands
if Txt(15) == '&' %if control character is received

if Txt(1) == 'S'; %start data acquisition
    En_System = 1; %enable the system
end

if Txt(2) == '1'
    Ch0_En = 1;
end
if Txt(3) == '1'
    Ch1_En = 1;
end
if Txt(4) == '1'
    Ch2_En = 1;
end
if Txt(5) == '1'
    Ch3_En = 1;
end
if Txt(6) == '1'
    Ch4_En = 1;
end
if Txt(7) == '1'
    Ch5_En = 1;
end
if Txt(8) == '1'
    Ch6_En = 1;
end
end
```

```

    if Txt(9) == '1'
        Ch7_En = 1;
    end
    if Txt(10) == '1'
        Ch8_En = 1;
    end
    if Txt(11) == '1'
        Ch9_En = 1;
    end
    if Txt(12) == '1'
        Ch10_En = 1;
    end
    if Txt(13) == '1'
        Ch11_En = 1;
    end
    end
    %if channel enabled

if Txt(2) == 'Z' %all channels selected
    Ch0_En = 1;
    Ch1_En = 1;
    Ch2_En = 1;
    Ch3_En = 1;
    Ch4_En = 1;
    Ch5_En = 1;
    Ch6_En = 1;
    Ch7_En = 1;
    Ch8_En = 1;
    Ch9_En = 1;
    Ch10_En = 1;
    Ch11_En = 1;

    end

end

if and(Txt(1) == 'X', Txt(2) == '+'); %stop data acquisition
    En_System = 0; %disable the system
end

```

Appendix G DAS Host GUI Matlab Code

DAS Host GUI Code

```
function varargout = HostGUI_AA(varargin)
% HOSTGUI_AA MATLAB code for HostGUI_AA.fig
%   HOSTGUI_AA, by itself, creates a new HOSTGUI_AA or raises the
existing
%   singleton*.
%
%   H = HOSTGUI_AA returns the handle to a new HOSTGUI_AA or the handle
to
%   the existing singleton*.
%
%   HOSTGUI_AA('CALLBACK',hObject,eventData,handles,...) calls the local
%   function named CALLBACK in HOSTGUI_AA.M with the given input
arguments.
%
%   HOSTGUI_AA('Property','Value',...) creates a new HOSTGUI_AA or
raises the
%   existing singleton*. Starting from the left, property value pairs
are
%   applied to the GUI before HostGUI_AA_OpeningFcn gets called. An
%   unrecognized property name or invalid value makes property
application
%   stop. All inputs are passed to HostGUI_AA_OpeningFcn via varargin.
%
%   *See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one
%   instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help HostGUI_AA

% Last Modified by GUIDE v2.5 29-Jan-2015 21:49:13

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',  gui_Singleton, ...
                  'gui_OpeningFcn', @HostGUI_AA_OpeningFcn, ...
                  'gui_OutputFcn',  @HostGUI_AA_OutputFcn, ...
                  'gui_LayoutFcn',  [], ...
                  'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
```

```

end
% End initialization code - DO NOT EDIT

% --- Executes just before HostGUI_AA is made visible.
function HostGUI_AA_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to HostGUI_AA (see VARARGIN)

% Choose default command line output for HostGUI_AA
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);
%Global variables for tracking status
global CheckBox_Status;
CheckBox_Status = 0;
global CheckBox1_Status;
CheckBox1_Status = 0;
global CheckBox2_Status;
CheckBox2_Status = 0;
global CheckBox3_Status;
CheckBox3_Status = 0;
global CheckBox4_Status;
CheckBox4_Status = 0;
global CheckBox5_Status;
CheckBox5_Status = 0;
global CheckBox6_Status;
CheckBox6_Status = 0;
global CheckBox7_Status;
CheckBox7_Status = 0;
global CheckBox8_Status;
CheckBox8_Status = 0;
global CheckBox9_Status;
CheckBox9_Status = 0;
global CheckBox10_Status;
CheckBox10_Status = 0;
global CheckBox11_Status;
CheckBox11_Status = 0;
global CheckBox12_Status;
CheckBox12_Status = 0;
global StartCommand_Status
StartCommand_Status = 0;
global Port_Status
Port_Status = 0;
global filecounter
filecounter = 0;
global RxData
RxData = 0;
global ScrabPad
ScrabPad = 0;
% UIWAIT makes HostGUI_AA wait for user response (see UIRESUME)

```

```

% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = HostGUI_AA_OutputFcn(hObject, eventdata, handles)
% varargout cell array for returning output args (see VARARGOUT);
% hObject handle to figure
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

% --- Executes on button press in channel1.
function channel1_Callback(hObject, eventdata, handles)
% hObject handle to channel1 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of channel1
%check the status of check box and perform appr action
global CheckBox1_Status
global CheckBox_Status
if CheckBox_Status == 'Z';
    set(hObject,'Value',1.0);
elseif (get(hObject,'Value') == get(hObject,'Max'))
display('Selected');
    CheckBox1_Status = 1;
else
display('Not selected');
CheckBox1_Status = 0;
end

% --- Executes on button press in channel2.
function channel2_Callback(hObject, eventdata, handles)
% hObject handle to channel2 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of channel2
%check the status of check box and perform appropriate action
global CheckBox_Status
global CheckBox2_Status
if CheckBox_Status == 'Z';
    set(hObject,'Value',1.0);
elseif (get(hObject,'Value') == get(hObject,'Max'))
display('Selected');
CheckBox2_Status = 1;
else
display('Not selected');
CheckBox2_Status = 0;
end

```

```

% --- Executes on button press in channel3.
function channel3_Callback(hObject, eventdata, handles)
% hObject    handle to channel3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of channel3
%check the status of check box and perform appropriate action
global CheckBox_Status
global CheckBox3_Status
if CheckBox_Status == 'Z';
    set(hObject,'Value','Max');
elseif (get(hObject,'Value') == get(hObject,'Max'))
display('Selected');
CheckBox3_Status = 1;
else
display('Not selected');
CheckBox3_Status = 0;
end

% --- Executes on button press in channel4.
function channel4_Callback(hObject, eventdata, handles)
% hObject    handle to channel4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of channel4
%check the status of check box and perform appropriate action
global CheckBox_Status
global CheckBox4_Status
if CheckBox_Status == 'Z';
    set(hObject,'Value','Max');
elseif (get(hObject,'Value') == get(hObject,'Max'))
display('Selected');
CheckBox4_Status = 1;
else
display('Not selected');
CheckBox4_Status = 0;
end

% --- Executes on button press in channel5.
function channel5_Callback(hObject, eventdata, handles)
% hObject    handle to channel5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of channel5
%check the status of check box and perform appropriate action
global CheckBox_Status
global CheckBox5_Status
if CheckBox_Status == 'Z';
    set(hObject,'Value','Max');
elseif (get(hObject,'Value') == get(hObject,'Max'))
display('Selected');
CheckBox5_Status = 1;

```

```

else
display('Not selected');
CheckBox5_Status = 0;
end

% --- Executes on button press in channel6.
function channel6_Callback(hObject, eventdata, handles)
% hObject    handle to channel6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of channel6
%check the status of check box and perform appropriate action
global CheckBox_Status
global CheckBox6_Status
if CheckBox_Status == 'Z';
    set(hObject,'Value','Max');
elseif (get(hObject,'Value') == get(hObject,'Max'))
display('Selected');
CheckBox6_Status = 1;
else
display('Not selected');
CheckBox6_Status = 0;
end

% --- Executes on button press in channel7.
function channel7_Callback(hObject, ~, handles)
% hObject    handle to channel7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of channel7
%check the status of check box and perform appropriate action
global CheckBox_Status
global CheckBox7_Status
if CheckBox_Status == 'Z';
    set(hObject,'Value','Max');
elseif (get(hObject,'Value') == get(hObject,'Max'))
display('Selected');
CheckBox7_Status = 1;
else
display('Not selected');
CheckBox7_Status = 0;
end

% --- Executes on button press in channel8.
function channel8_Callback(hObject, eventdata, handles)
% hObject    handle to channel8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of channel8
%check the status of check box and perform appropriate action
global CheckBox_Status
global CheckBox8_Status

```

```

if CheckBox_Status == 'Z';
    set(hObject,'Value','Max');
elseif (get(hObject,'Value') == get(hObject,'Max'))
display('Selected');
CheckBox8_Status = 1;
else
display('Not selected');
CheckBox8_Status = 0;
end

% --- Executes on button press in channel9.
function channel9_Callback(hObject, eventdata, handles)
% hObject    handle to channel9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of channel9
%check the status of check box and perform appropriate action
global CheckBox_Status
global CheckBox9_Status
if CheckBox_Status == 'Z';
    set(hObject,'Value','Max');
elseif (get(hObject,'Value') == get(hObject,'Max'))
display('Selected');
CheckBox9_Status = 1;
else
display('Not selected');
CheckBox9_Status = 0;
end

% --- Executes on button press in channel10.
function channel10_Callback(hObject, eventdata, handles)
% hObject    handle to channel10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of channel10
%check the status of check box and perform appropriate action
global CheckBox_Status
global CheckBox10_Status
if CheckBox_Status == 'Z';
    set(hObject,'Value','Max');
elseif (get(hObject,'Value') == get(hObject,'Max'))
display('Selected');
CheckBox10_Status = 1;
else
display('Not selected');
CheckBox10_Status = 0;
end

% --- Executes on button press in channel11.
function channel11_Callback(hObject, eventdata, handles)
% hObject    handle to channel11 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

```

```

% Hint: get(hObject,'Value') returns toggle state of channel11
%check the status of check box and perform appropriate action
global CheckBox_Status
global CheckBox11_Status
if CheckBox_Status == 'Z';
    set(hObject,'Value','Max');
elseif (get(hObject,'Value') == get(hObject,'Max'))
display('Selected');
CheckBox11_Status = 1;
else
display('Not selected');
CheckBox11_Status = 0;
end

% --- Executes on button press in channel12.
function channel12_Callback(hObject, eventdata, handles)
% hObject    handle to channel12 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of channel12
%check the status of check box and perform appropriate action
global CheckBox_Status
global CheckBox12_Status
if CheckBox_Status == 'Z';
    set(hObject,'Value',1.0);
elseif (get(hObject,'Value') == get(hObject,'Max'))
display('Selected');
CheckBox12_Status = 1;
else
display('Not selected');
CheckBox12_Status = 0;
%set(hObject,'Value','Min');
end

%-----Function to read Serial Data
function y = ReadSerialPort()
global Port1
global RxData
global ScrabPad
RxData = fread(Port1,270); %fread(Port1,27);    %read 270 bytes at a time to
RxData until stopped works but too slow
ScrabPad = RxData;

% --- Executes on button press in Startbutton1.
function Startbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to Startbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%1. Open Port
% create serial port object only once
global filecounter
%filecounter = filecounter + 1;

```

```

set(hObject,'Enable','off'); %disable start button
global Port_Status
global Port1
if Port_Status == 0
    Port1 = serial('COM6','BaudRate',115200,'DataBits',8);
    set(Port1,'InputBufferSize',2000000) %set the buffer size to 10000
bytes
    set(Port1,'ReadAsyncMode','continuous');%Continuously query the device
to determine if data is available to be read
    set(Port1,'Timeout',10); %Waiting time to complete a read or write
operation in seconds

    if strcmp(Port1.status,'open')
        errordlg('Serial Port Already Open'); %display message is open
        Port_Status = 1;
    elseif strcmp(Port1.status,'closed')
        fopen(Port1); %open port if available
        Port_Status = 1;
        display('Port Open')
    end
end
% 2. Send command to serial port
global CheckBox_Status
global CheckBox1_Status
global CheckBox2_Status
global CheckBox3_Status
global CheckBox4_Status
global CheckBox5_Status
global CheckBox6_Status
global CheckBox7_Status
global CheckBox8_Status
global CheckBox9_Status
global CheckBox10_Status
global CheckBox11_Status
global CheckBox12_Status
global StartCommand_Status

selected_channel = '0';
if StartCommand_Status == 0 %command not sent
    if
        CheckBox1_Status||CheckBox2_Status||CheckBox3_Status||CheckBox4_Status||Che
        ckBox5_Status||CheckBox6_Status||CheckBox7_Status||CheckBox8_Status||CheckB
        ox9_Status||CheckBox10_Status||CheckBox11_Status||CheckBox12_Status %== 1
        %channel selected

        channel_selected_array =
        [CheckBox1_Status,CheckBox2_Status,CheckBox3_Status,CheckBox4_Status,CheckB
        ox5_Status,CheckBox6_Status,CheckBox7_Status,CheckBox8_Status,CheckBox9_Sta
        tus,CheckBox10_Status,CheckBox11_Status,CheckBox12_Status];

        for i=1:12
            if channel_selected_array(i) == 1
                selected_channel(i) = '1';
            else
                selected_channel(i) = '0';
            end
        end
    end
end

```

```

        end
    end
    %selected_channel = uint8(channel_selected_array);
    selected_channel_str = selected_channel;
    TxData = ['S',selected_channel_str,'0','&','0']; %prepare message
    CommandData = TxData;
    %write command to serial port object only one time
    fprintf(Port1,CommandData,'async'); %send message
    StartCommand_Status = 1; %command to start signal
acquisition send once
end
    if CheckBox_Status == 'Z' %all channels selected
        TxData = ['S',CheckBox_Status,'0','&','0']; %prepare message
        CommandData = TxData;
        %write command to serial port object only one time
        fprintf(Port1,CommandData,'async'); %send message
        StartCommand_Status = 1; %command to start signal
acquisition send once
    end

end
%2. File Management
    if StartCommand_Status == 1; %command send
        filecounter = filecounter + 1; % increment everytime a new data
acq session starts
        filecounter_str = int2str(filecounter);

        if CheckBox1_Status == 1 %channel selected
open file to save it
            s = datestr(date);
            filename = ['Ch1Data ',filecounter_str,' ',s,'.txt'];
            ch1_fileId = fopen(filename, 'a'); %openfile
        end

        if CheckBox2_Status == 1 %channel
selected open file to save it
            s = datestr(date);
            filename = ['Ch2Data ',filecounter_str,' ',s,'.txt'];
            ch2_fileId = fopen(filename, 'a'); %openfile
        end

        if CheckBox3_Status == 1 %channel
selected open file to save it
            s = datestr(date);
            filename = ['Ch3Data ',filecounter_str,' ',s,'.txt'];
            ch3_fileId = fopen(filename, 'a'); %openfile
        end

        if CheckBox4_Status == 1 %channel
selected open file to save it
            s = datestr(date);
            filename = ['Ch4Data ',filecounter_str,' ',s,'.txt'];
            ch4_fileId = fopen(filename, 'a'); %openfile
        end
end

```

```

        if CheckBox5_Status == 1                                %channel
selected open file to save it
        s = datestr(date);
        filename = ['Ch5Data ',filecounter_str,' ',s,'.txt'];
        ch5_fileId = fopen(filename, 'a');                    %openfile
    end

        if CheckBox6_Status == 1                                %channel
selected open file to save it
        s = datestr(date);
        filename = ['Ch6Data ',filecounter_str,' ',s,'.txt'];
        ch6_fileId = fopen(filename, 'a');                    %openfile
    end

        if CheckBox7_Status == 1                                %channel
selected open file to save it
        s = datestr(date);
        filename = ['Ch7Data ',filecounter_str,' ',s,'.txt'];
        ch7_fileId = fopen(filename, 'a');                    %openfile
    end

        if CheckBox8_Status == 1                                %channel
selected open file to save it
        s = datestr(date);
        filename = ['Ch8Data ',filecounter_str,' ',s,'.txt'];
        ch8_fileId = fopen(filename, 'a');                    %openfile
    end

        if CheckBox9_Status == 1                                %channel selected
open file to save it
        s = datestr(date);
        filename = ['Ch9Data ',filecounter_str,' ',s,'.txt'];
        ch9_fileId = fopen(filename, 'a');                    %openfile
    end

        if CheckBox10_Status == 1                               %channel
selected open file to save it
        s = datestr(date);
        filename = ['Ch10Data ',filecounter_str,' ',s,'.txt'];
        ch10_fileId = fopen(filename, 'a');                   %openfile
    end

        if CheckBox11_Status == 1                               %channel
selected open file to save it
        s = datestr(date);
        filename = ['Ch11Data ',filecounter_str,' ',s,'.txt'];
        ch11_fileId = fopen(filename, 'a');                   %openfile
    end

        if CheckBox12_Status == 1                               %channel
selected open file to save it
        s = datestr(date);
        filename = ['Ch12Data ',filecounter_str,' ',s,'.txt'];
        ch12_fileId = fopen(filename, 'a');                   %openfile
    end

```

```

        end

        if CheckBox_Status == 'Z'                                %all channels
selected
            s = datestr(date);
            filename = ['Ch1Data ',filecounter_str,' ',s,'.txt'];
            ch1_fileId = fopen(filename, 'a');                    %openfile

            filename = ['Ch2Data ',filecounter_str,' ',s,'.txt'];
            ch2_fileId = fopen(filename, 'a');                    %openfile

            filename = ['Ch3Data ',filecounter_str,' ',s,'.txt'];
            ch3_fileId = fopen(filename, 'a');                    %openfile

            filename = ['Ch4Data ',filecounter_str,' ',s,'.txt'];
            ch4_fileId = fopen(filename, 'a');                    %openfile

            filename = ['Ch5Data ',filecounter_str,' ',s,'.txt'];
            ch5_fileId = fopen(filename, 'a');                    %openfile

            filename = ['Ch6Data ',filecounter_str,' ',s,'.txt'];
            ch6_fileId = fopen(filename, 'a');                    %openfile

            filename = ['Ch7Data ',filecounter_str,' ',s,'.txt'];
            ch7_fileId = fopen(filename, 'a');                    %openfile

            filename = ['Ch8Data ',filecounter_str,' ',s,'.txt'];
            ch8_fileId = fopen(filename, 'a');                    %openfile

            filename = ['Ch9Data ',filecounter_str,' ',s,'.txt'];
            ch9_fileId = fopen(filename, 'a');                    %openfile

            filename = ['Ch10Data ',filecounter_str,' ',s,'.txt'];
            ch10_fileId = fopen(filename, 'a');                    %openfile

            filename = ['Ch11Data ',filecounter_str,' ',s,'.txt'];
            ch11_fileId = fopen(filename, 'a');                    %openfile

            filename = ['Ch12Data ',filecounter_str,' ',s,'.txt'];
            ch12_fileId = fopen(filename, 'a');

        end

    end

    %3.read responds from DAS controller and start collecting data
    global RxData
    global ScrabPad
    global StopDataAcqStatus
    StopDataAcqStatus = 0;                                       %start data collection
    %RxData = fread(Port1);
    N = 10;
    Vref = 3.3; %Arduino Due is 3.3;                               %theoretical reference
    FS = (2^N) - 1;
    ScaleFactor1 = 1;%(1/0.219359875); %4-20mA channels scale factor

```

```

ScaleFactor2 = 1;%(1/0.2424242424);%0-10V channels scale factor
counter = 0;
index = 0;
indexcounter = 0;
myH = gca; %Get current Axes handle
title('Collected Data');
xlabel('Sample Number'); %label x axis
ylabel('Volt');%('Pa'); %use temporary to be set by
units in listbox.
axis([0 inf 0 3.5]); %set axis limits
%catch communication failure
if (get(Port1, 'BytesAvailable') == 0)
    disp('Data not avail yet. Try again or check connection.')
    StopDataAcqStatus = 1; %stop data collection
    set(hObject, 'Enable', 'on'); %enable start button
    return
end
hold(handles.axes1, 'on') %retains plots in the axes specified by
ax so that new plots added to the axes do not delete existing plots
%
%

while counter ~= 5001 %StopDataAcqStatus ~= 1 %(get(Port1,
'BytesAvailable') >= 27) %counter ~= 401 %StopDataAcqStatus ~= 1
%collect,store and display data until stopped
counter = counter + 1; % data points for x axis i.e counting number of data
points
indexcounter = indexcounter + 1;

    if get(Port1, 'BytesAvailable') ~= 0 % if data not avialable do not
read port
        ReadSerialPort();
    end

%debugging &testing
if numel(RxData) == 0
    disp('Memory Error')
    get(Port1, 'BytesAvailable')
    return
end
%index to access data on 270 byte array
if indexcounter < 10
    index = indexcounter * 27;
else
    indexcounter = 0;
end
%check if the correct header was received
if RxData(1 + index) == 'R' %RxData(1) == 'R' %ScrabPad(1) == 'R'
%Process Data for plotting
%data for channel 1
    if CheckBox1_Status == 1 %channel selected plot
it
        ch1_DigData = ScrabPad(4 + index)*256 + ScrabPad(3 +
index);%ScrabPad(4)*256 + ScrabPad(3);%ScrabPad(4+27*(counter-1))*256 +

```

```

ScrabPad(3+27*(counter-1));    %signal digital value (convert uint8 to
uint16)
    ch1_PlotData = ((Vref*ch1_DigData)/FS)*ScaleFactor1;    % data to
plot
    % Create plot
                                %
        subplot(3,3,2);
        plot(counter,ch1_PlotData,'bx') ;%plot(counter,ch1_PlotData,'gx') ;
%plot(x,y) 2D

        title('Channel 1')
        axis([0 inf 0 1.2])
        xlabel('Samples');
        ylabel('V');                                %use temporary to be set by
units list.
        grid on;
        drawnow;
        hold on;
    end
    %data for channel 2
    if CheckBox2_Status == 1                                %channel selected plot
it
        ch2_DigData = ScrabPad(6 + index)*256 + ScrabPad(5 +
index);%ScrabPad(4+27*(counter-1))*256 + ScrabPad(3+27*(counter-1));
%signal digital value (convert uint8 to uint16)
        ch2_PlotData = ((Vref*ch2_DigData)/FS)*ScaleFactor1;    % data to
plot
        % Create plot
        %hold on;
        subplot(3,3,3);%
        plot(counter,ch2_PlotData,'ro') ;                %plot(x,y) 2D changed
from plot to subplot 20 January

        title('Channel 2')
        axis([0 inf 0 1.2])
        xlabel('Samples');
        ylabel('Rev/s');                                %use temporary to be
set by units list.
        grid on;
        drawnow;
        hold on;
    end

    %data for channel 3
    if CheckBox3_Status == 1                                %channel selected plot
it
        ch3_DigData = ScrabPad(8 + index)*256 + ScrabPad(7 +
index);%ScrabPad(4+27*(counter-1))*256 + ScrabPad(3+27*(counter-1));
%signal digital value (convert uint8 to uint16)
        ch3_PlotData = ((Vref*ch3_DigData)/FS)*ScaleFactor1;    % data to
plot
        %to draw plot code here
        subplot(3,3,5);%
        plot(counter,ch3_PlotData,'yo') ;                %plot(x,y) 2D changed
from plot to subplot 20 January

```

```

        title('Channel 3')
        axis([0 inf 0 2])
        xlabel('Samples');
        ylabel('J');
units list.
        grid on;
        drawnow;
        hold on;
    end

    %data for channel 4
    if CheckBox4_Status == 1
it
        ch4_DigData = ScrabPad(10 + index)*256 + ScrabPad(9 +
index);%ScrabPad(4+27*(counter-1))*256 + ScrabPad(3+27*(counter-1));
        %signal digital value (convert uint8 to uint16)
        ch4_PlotData = ((Vref*ch4_DigData)/FS)*ScaleFactor1;
plot
        %to draw plot code here
        subplot(3,3,6);%
        plot(counter,ch4_PlotData,'m+') ;
from plot to subplot_20 January
        %plot(x,y) 2D changed

        title('Channel 4')
        axis([0 inf 0 2])
        xlabel('Samples');
        ylabel('rad');
by units list.
        grid on;
        drawnow;
        hold on;
    end
    %data for channel 5
    if CheckBox5_Status == 1
it
        ch5_DigData = ScrabPad(12 + index)*256 + ScrabPad(11 +
index);%ScrabPad(4+27*(counter-1))*256 + ScrabPad(3+27*(counter-1));
        %signal digital value (convert uint8 to uint16)
        ch5_PlotData = ((Vref*ch5_DigData)/FS)*ScaleFactor1;
plot
        %to draw plot code here
        subplot(3,3,8);%
        plot(counter,ch5_PlotData,'cs') ;
from plot to subplot 20 January
        %plot(x,y) 2D changed

        title('Channel 5')
        axis([0 inf 0 2])
        xlabel('Samples');
        ylabel('N');
units list.
list.
        grid on;
        drawnow;
        hold on;
        %use temporary to be set by
        %use temporary to be set by units

```

```

end

%data for channel 6
if CheckBox6_Status == 1                                %channel selected plot
it
    ch6_DigData = ScrabPad(14 + index)*256 + ScrabPad(13 +
index);%ScrabPad(4+27*(counter-1))*256 + ScrabPad(3+27*(counter-1));
%signal digital value (convert uint8 to uint16)
    ch6_PlotData = ((Vref*ch6_DigData)/FS)*ScaleFactor1;      % data to
plot
    %to draw plot code here
    subplot(3,3,9);%
    plot(counter,ch6_PlotData,'bd') ;                        %plot(x,y) 2D changed
from plot to subplot 20 January

    title('Channel 6')
    axis([0 inf 0 2])
    xlabel('Samples');
    ylabel('K');                                           %use temporary to be set by
units list.
    grid on;
    drawnow;
    hold on;
end

%data for channel 7
if CheckBox7_Status == 1                                %channel selected plot
it
    ch7_DigData = ScrabPad(16 + index)*256 + ScrabPad(15 +
index);%ScrabPad(4+27*(counter-1))*256 + ScrabPad(3+27*(counter-1));
%signal digital value (convert uint8 to uint16)
    ch7_PlotData = ((Vref*ch7_DigData)/FS)*ScaleFactor1;      % data to
plot
    %to draw plot code here
    subplot(3,3,2);%
    plot(counter,ch7_PlotData,'kv') ;                        %plot(x,y) 2D changed
from plot to subplot 20 January

    title('Channel 7')
    axis([0 inf 0 2])
    xlabel('Samples');
    ylabel('Ohm');                                         %use temporary to be set
by units list.
    grid on;
    drawnow;
    hold on;
end

%data for channel 8
if CheckBox8_Status == 1                                %channel selected plot
it
    ch8_DigData = ScrabPad(18 + index)*256 + ScrabPad(17 +
index);%ScrabPad(4+27*(counter-1))*256 + ScrabPad(3+27*(counter-1));
%signal digital value (convert uint8 to uint16)

```

```

    ch8_PlotData = ((Vref*ch8_DigData)/FS)*ScaleFactor2;           % data to
plot
    %to draw plot code here
    subplot(3,3,3);%
    plot(counter,ch8_PlotData,'k^') ;                             %plot(x,y) 2D changed
from plot to subplot 20 January

    title('Channel 8')
    axis([0 inf 0 2.5])
    xlabel('Samples');
    ylabel('mm');                                                 %use temporary to be set
by units list.
    grid on;
    drawnow;
    hold on;
end

    %data for channel 9
    if CheckBox9_Status == 1                                     %channel selected plot
it
        ch9_DigData = ScrabPad(20 + index)*256 + ScrabPad(19 +
index);%ScrabPad(4+27*(counter-1))*256 + ScrabPad(3+27*(counter-1));
%signal digital value (convert uint8 to uint16)
        ch9_PlotData = ((Vref*ch9_DigData)/FS)*ScaleFactor2;           % data to
plot
        %to draw plot code here
        subplot(3,3,5);%
        plot(counter,ch9_PlotData,'k<') ;                         %plot(x,y) 2D changed
from plot to subplot 20 January

        title('Channel 9')
        axis([0 inf 0 2.5])
        xlabel('Samples');
        ylabel('m');                                             %use temporary to be set by
units list.
        grid on;
        drawnow;
        hold on;
    end

    %data for channel 10
    if CheckBox10_Status == 1                                   %channel selected plot
it
        ch10_DigData = ScrabPad(22 + index)*256 + ScrabPad(21 +
index);%ScrabPad(4+27*(counter-1))*256 + ScrabPad(3+27*(counter-1));
%signal digital value (convert uint8 to uint16)
        ch10_PlotData = ((Vref*ch10_DigData)/FS)*ScaleFactor2;           % data to
plot
        %to draw plot code here
        subplot(3,3,6);%
        plot(counter,ch10_PlotData,'k>') ;                       %plot(x,y) 2D changed
from plot to subplot 20 January

        title('Channel 10')
        axis([0 inf 0 2])

```

```

        xlabel('Samples');
        ylabel('m/s');
        %use temporary to be set
by units list.
        grid on;
        drawnow;
        hold on;
    end

    if CheckBox11_Status == 1
        %channel selected plot
    it
        ch11_DigData = ScrabPad(24 + index)*256 + ScrabPad(23 +
index);%ScrabPad(4+27*(counter-1))*256 + ScrabPad(3+27*(counter-1));
        %signal digital value (convert uint8 to uint16)
        ch11_PlotData = ((Vref*ch11_DigData)/FS)*ScaleFactor1;
        % data to
    plot
        %to draw plot code here
        subplot(3,3,8);%
        plot(counter,ch11_PlotData,'kp') ;
        %plot(x,y) 2D changed
    from plot to subplot 20 January

        title('Channel 11')
        axis([0 inf 0 1.2])
        xlabel('Samples');
        ylabel('Kg');
        %use temporary to be set
by units list.
        %use temporary to be set by
    units list.
        grid on;
        drawnow;
        hold on;
    end

    %data for channel 12
    if CheckBox12_Status == 1
        %channel selected plot
    it
        ch12_DigData = ScrabPad(26 + index)*256 + ScrabPad(25 +
index);%ScrabPad(4+27*(counter-1))*256 + ScrabPad(3+27*(counter-1));
        %signal digital value (convert uint8 to uint16)
        ch12_PlotData = ((Vref*ch12_DigData)/FS)*ScaleFactor2;
        % data to
    plot
        %to draw plot code here
        subplot(3,3,9);%
        plot(counter,ch12_PlotData,'kh');
        %plot(x,y) 2D changed
    from plot to subplot 20 January

        title('Channel 12')
        axis([0 inf 0 2.5])
        xlabel('Samples');
        ylabel('Pa');
        %use temporary to be set
by units list.
        grid on;
        drawnow;
        hold on;
    end
else
    StopDataAcqStatus = 1;
        %stop data collection

```

```

set(hObject,'Enable','on');           %enable start button
display('no header found')
%return                               %no header found
if get(Port1, 'BytesAvailable') ~= 0 % if data not available do not read
port
    ReadSerialPort();
end;%try this
end

%save data to disk
if CheckBox1_Status == 1               %channel selected save
data
    fprintf(ch1_fileId,'%u,           %f\n',counter,ch1_PlotData);
%StoreData);
end

if CheckBox2_Status == 1               %channel selected save
data
    fprintf(ch2_fileId,'%u,           %f\n',counter,ch2_PlotData);
%StoreData
end

if CheckBox3_Status == 1               %channel selected save
data
    fprintf(ch3_fileId,'%u,           %f\n',counter,ch3_PlotData);
%StoreData
end

if CheckBox4_Status == 1               %channel selected save
data
    fprintf(ch4_fileId,'%u,           %f\n',counter,ch4_PlotData);
%StoreData
end

if CheckBox5_Status == 1               %channel selected save
data
    fprintf(ch5_fileId,'%u,           %f\n',counter,ch5_PlotData);
%StoreData);
end

if CheckBox6_Status == 1               %channel selected save
data
    fprintf(ch6_fileId,'%u,           %f\n',counter,ch6_PlotData);
%StoreData
end

if CheckBox7_Status == 1               %channel selected save
data
    fprintf(ch7_fileId,'%u,           %f\n',counter,ch7_PlotData);
%StoreData
end

if CheckBox8_Status == 1               %channel selected save
data

```

```

        fprintf(ch8_fileId,'%u,      %f\n',counter,ch8_PlotData);
%StoreData
    end
    if CheckBox9_Status == 1          %channel selected save
data
        fprintf(ch9_fileId,'%u,      %f\n',counter,ch9_PlotData);
%StoreData);
    end

    if CheckBox10_Status == 1         %channel selected save
data
        fprintf(ch10_fileId,'%u,      %f\n',counter,ch10_PlotData);
%StoreData
    end

    if CheckBox11_Status == 1         %channel selected save
data
        fprintf(ch11_fileId,'%u,      %f\n',counter,ch11_PlotData);
%StoreData
    end

    if CheckBox12_Status == 1         %channel selected save
data
        fprintf(ch12_fileId,'%u,      %f\n',counter,ch12_PlotData);
%StoreData
    end

    if CheckBox_Status == 'Z'        %all channels
selected
        fprintf(ch1_fileId,'%u,      %f\n',counter,ch1_PlotData);
%StoreData
        fprintf(ch2_fileId,'%u,      %f\n',counter,ch2_PlotData);
%StoreData
        fprintf(ch3_fileId,'%u,      %f\n',counter,ch3_PlotData);
%StoreData
        fprintf(ch4_fileId,'%u,      %f\n',counter,ch4_PlotData);
%StoreData
        fprintf(ch5_fileId,'%u,      %f\n',counter,ch5_PlotData);
%StoreData
        fprintf(ch6_fileId,'%u,      %f\n',counter,ch6_PlotData);
%StoreData
        fprintf(ch7_fileId,'%u,      %f\n',counter,ch7_PlotData);
%StoreData
        fprintf(ch8_fileId,'%u,      %f\n',counter,ch8_PlotData);
%StoreData
        fprintf(ch9_fileId,'%u,      %f\n',counter,ch9_PlotData);
%StoreData
        fprintf(ch10_fileId,'%u,      %f\n',counter,ch10_PlotData);
%StoreData
        fprintf(ch11_fileId,'%u,      %f\n',counter,ch11_PlotData);
%StoreData
        fprintf(ch12_fileId,'%u,      %f\n',counter,ch12_PlotData);
%StoreData
    end %end of all channel selected

```

```

%for debugging & testing
if counter == 50
    %fclose(fileId);    %close file
    if CheckBox1_Status == 1
        fclose(ch1_fileId);
    end
    if CheckBox2_Status == 1
        fclose(ch2_fileId);
    end
    if CheckBox3_Status == 1
        fclose(ch3_fileId);
    end
    if CheckBox4_Status == 1
        fclose(ch4_fileId);
    end
    if CheckBox5_Status == 1
        fclose(ch5_fileId);
    end
    if CheckBox6_Status == 1
        fclose(ch6_fileId);
    end
    if CheckBox7_Status == 1
        fclose(ch7_fileId);
    end
    if CheckBox8_Status == 1
        fclose(ch8_fileId);
    end
    if CheckBox9_Status == 1
        fclose(ch9_fileId);
    end
    if CheckBox10_Status == 1
        fclose(ch10_fileId);
    end
    if CheckBox11_Status == 1
        fclose(ch11_fileId);
    end
    if CheckBox12_Status == 1
        fclose(ch12_fileId);
    end
    get(Port1, 'BytesAvailable')
return
end

end %end of while

% --- Executes on button press in Stopbutton2.
function Stopbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to Stopbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global StopDataAcqStatus
StopDataAcqStatus = 1;          %stop data collection
global Port1
%write stop command to serial port object only one time

```

```

CommandData =
['X','+', '0','0','0','0','0','0','0','0','0','0','0','0','0','0'];
%stop command
fprintf(Port1,CommandData,'async'); %send message
set(handles.Startbutton1,'Enable','on'); %enable start button

% --- Executes on button press in SelectAll14.
function SelectAll14_Callback(hObject, eventdata, handles)
% hObject    handle to SelectAll14 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of SelectAll14
% get status of checkbox
global CheckBox_Status;
if (get(hObject,'Value') == get(hObject,'Max'))
display('Selected');
CheckBox_Status = 'Z';
else
display('Not selected');
CheckBox_Status = '0';
%set(hObject,'Value','Min');
end

% --- Executes on selection change in chlunits.
function chlunits_Callback(hObject, eventdata, handles)
% hObject    handle to chlunits (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns chlunits
%         contents as cell array
%         contents{get(hObject,'Value')} returns selected item from chlunits

% --- Executes during object creation, after setting all properties.
function chlunits_CreateFcn(hObject, eventdata, handles)
% hObject    handle to chlunits (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: listbox controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor','white');
end

% --- Executes on selection change in listbox2.
function listbox2_Callback(hObject, eventdata, handles)
% hObject    handle to listbox2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

```

```

% Hints: contents = cellstr(get(hObject,'String')) returns listBox2
contents as cell array
%         contents{get(hObject,'Value')} returns selected item from listBox2

% --- Executes during object creation, after setting all properties.
function listBox2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to listBox2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: listBox controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on selection change in ch3units.
function ch3units_Callback(hObject, eventdata, handles)
% hObject    handle to ch3units (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns ch3units
contents as cell array
%         contents{get(hObject,'Value')} returns selected item from ch3units

% --- Executes during object creation, after setting all properties.
function ch3units_CreateFcn(hObject, eventdata, handles)
% hObject    handle to ch3units (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: listBox controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on selection change in ch4units.
function ch4units_Callback(hObject, eventdata, handles)
% hObject    handle to ch4units (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns ch4units
contents as cell array

```

```

%         contents(get(hObject,'Value')) returns selected item from ch4units

% --- Executes during object creation, after setting all properties.
function ch4units_CreateFcn(hObject, eventdata, handles)
% hObject    handle to ch4units (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: listbox controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on selection change in ch5units.
function ch5units_Callback(hObject, eventdata, handles)
% hObject    handle to ch5units (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns ch5units
%         contents as cell array
%         contents(get(hObject,'Value')) returns selected item from ch5units

% --- Executes during object creation, after setting all properties.
function ch5units_CreateFcn(hObject, eventdata, handles)
% hObject    handle to ch5units (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: listbox controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on selection change in ch6units.
function ch6units_Callback(hObject, eventdata, handles)
% hObject    handle to ch6units (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns ch6units
%         contents as cell array
%         contents(get(hObject,'Value')) returns selected item from ch6units

```

```

% --- Executes during object creation, after setting all properties.
function ch6units_CreateFcn(hObject, eventdata, handles)
% hObject    handle to ch6units (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: listbox controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on selection change in ch7units.
function ch7units_Callback(hObject, eventdata, handles)
% hObject    handle to ch7units (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns ch7units
%         contents as cell array
%         contents{get(hObject,'Value')} returns selected item from ch7units

% --- Executes during object creation, after setting all properties.
function ch7units_CreateFcn(hObject, eventdata, handles)
% hObject    handle to ch7units (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: listbox controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on selection change in ch8units.
function ch8units_Callback(hObject, eventdata, handles)
% hObject    handle to ch8units (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns ch8units
%         contents as cell array
%         contents{get(hObject,'Value')} returns selected item from ch8units

% --- Executes during object creation, after setting all properties.
function ch8units_CreateFcn(hObject, eventdata, handles)
% hObject    handle to ch8units (see GCBO)

```

```

% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns called

% Hint: listbox controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on selection change in ch9units.
function ch9units_Callback(hObject, eventdata, handles)
% hObject handle to ch9units (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns ch9units
contents as cell array
% contents{get(hObject,'Value')} returns selected item from ch9units

% --- Executes during object creation, after setting all properties.
function ch9units_CreateFcn(hObject, ~, ~)
% hObject handle to ch9units (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns called

% Hint: listbox controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on selection change in ch10units.
function ch10units_Callback(hObject, eventdata, handles)
% hObject handle to ch10units (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns ch10units
contents as cell array
% contents{get(hObject,'Value')} returns selected item from
ch10units

% --- Executes during object creation, after setting all properties.
function ch10units_CreateFcn(hObject, eventdata, handles)
% hObject handle to ch10units (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns called

```

```

% Hint: listbox controls usually have a white background on Windows.
%     See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on selection change in chl1units.
function chl1units_Callback(hObject, eventdata, handles)
% hObject     handle to chl1units (see GCBO)
% eventdata   reserved - to be defined in a future version of MATLAB
% handles     structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns chl1units
contents as cell array
%     contents{get(hObject,'Value')} returns selected item from
chl1units

% --- Executes during object creation, after setting all properties.
function chl1units_CreateFcn(hObject, ~, handles)
% hObject     handle to chl1units (see GCBO)
% eventdata   reserved - to be defined in a future version of MATLAB
% handles     empty - handles not created until after all CreateFcns called

% Hint: listbox controls usually have a white background on Windows.
%     See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on selection change in chl2units.
function chl2units_Callback(hObject, eventdata, handles)
% hObject     handle to chl2units (see GCBO)
% eventdata   reserved - to be defined in a future version of MATLAB
% handles     structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns chl2units
contents as cell array
%     contents{get(hObject,'Value')} returns selected item from
chl2units

% --- Executes during object creation, after setting all properties.
function chl2units_CreateFcn(hObject, eventdata, handles)
% hObject     handle to chl2units (see GCBO)
% eventdata   reserved - to be defined in a future version of MATLAB
% handles     empty - handles not created until after all CreateFcns called

```

```

% Hint: listbox controls usually have a white background on Windows.
%     See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

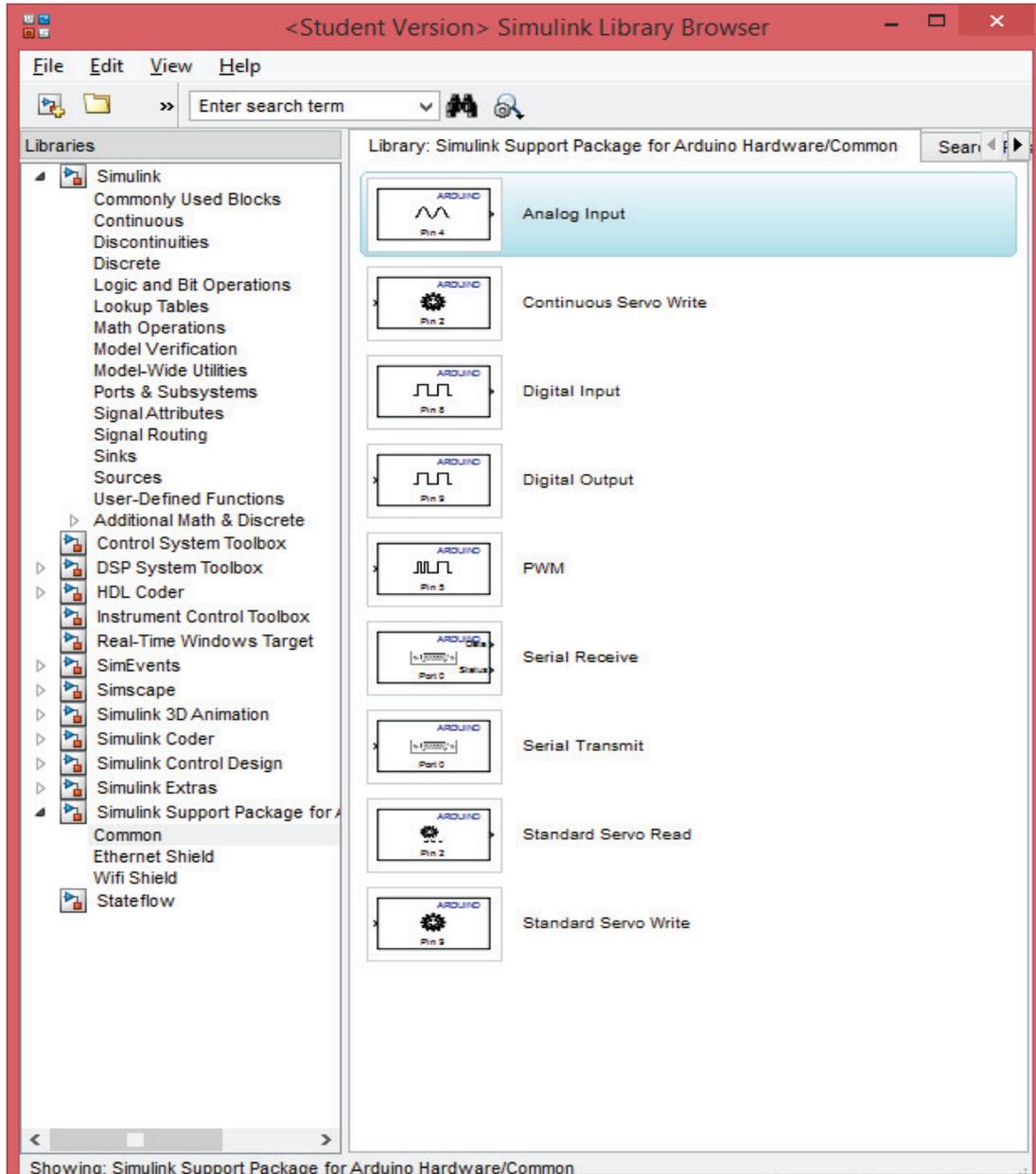
% --- Executes when user attempts to close figure1.
function figure1_CloseRequestFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: delete(hObject) closes the figure
global Port1
if strcmp(Port1.status,'open') %close the serial port if opened and delete
the object
fclose(Port1); %close the serial port
delete(Port1); %delete serial port object
end
delete(hObject);

```

Appendix H Arduino Library

Arduino® Block Library



Appendix I Relay Data Subsystem Matlab Code

Relay Data Subsystem Code

```
function Channel_Data =  
RelayData(CH0,CH1,CH2,CH3,CH4,CH5,CH6,CH7,CH8,CH9,CH10,CH11,u)  
%#codegen  
Max_Storage = length(u); %set the maximum size of each 1D vector  
Channel_Data = zeros(Max_Storage,1); %initialize to zero  
%*****  
%Check if channel is enabled , pass data of the channel to next stage  
if CH0 == 1  
    Channel_Data(1,1) = u(1,1); %copy only channel 1 data  
end  
  
if CH1 == 1  
    Channel_Data(2,1) = u(2,1); %copy only channel 2 data  
end  
  
if CH2 == 1  
    Channel_Data(3,1) = u(3,1); %copy only channel 3 data  
end  
  
if CH3 == 1  
    Channel_Data(4,1) = u(4,1); %copy only channel 4 data  
end  
  
if CH4 == 1  
    Channel_Data(5,1) = u(5,1); %copy only channel 5 data  
end  
  
if CH5 == 1  
    Channel_Data(6,1) = u(6,1); %%copy only channel 6 data  
end  
  
if CH6 == 1  
    Channel_Data(7,1) = u(7,1); %copy only channel 7 data  
end  
  
if CH7 == 1  
    Channel_Data(8,1) = u(8,1); %copy only channel 8 data  
end  
  
if CH8 == 1  
    Channel_Data(9,1) = u(9,1); %copy only channel 9 data  
end  
  
if CH9 == 1  
    Channel_Data(10,1) = u(10,1); %copy only channel 10 data  
end
```

```
if CH10 == 1
    Channel_Data(11,1) = u(11,1);    %%copy only channel 11 data
end

if CH11 == 1
    Channel_Data(12,1) = u(12,1);    %%copy only channel 12 data
end
```

Appendix J Arduino C/C++ DAS Test Code

Arduino C/C++ DAS Test Code

```
//DAS Data Acquisition Code
//constant
const int analogPin0 = 0;
const int analogPin1 = 1;
const int analogPin2 = 2;
const int analogPin3 = 3;
const int analogPin4 = 4;
const int analogPin5 = 5;
const int analogPin6 = 6;
const int analogPin7 = 7;
const int analogPin8 = 8;
const int analogPin9 = 9;
const int analogPin10 = 10;
const int analogPin11 = 11;
const int SystemAliveLed = 51;
boolean SystemEnable = false;
boolean Channel1Enabled = false;
boolean Channel2Enabled = false;
boolean Channel3Enabled = false;
boolean Channel4Enabled = false;
boolean Channel5Enabled = false;
boolean Channel6Enabled = false;
boolean Channel7Enabled = false;
boolean Channel8Enabled = false;
boolean Channel9Enabled = false;
boolean Channel10Enabled = false;
boolean Channel11Enabled = false;
```



```
}  
if (dataArray [2] == '1')  
{  
    Channel2Enabled = true;  
    //Serial1.write ('2'); //debugging  
}  
if (dataArray [3] == '1')  
Channel3Enabled = true;  
  
if (dataArray [4] == '1')  
Channel4Enabled = true;  
  
if (dataArray [5] == '1')  
Channel5Enabled = true;  
  
if (dataArray [6] == '1')  
Channel6Enabled = true;  
  
if (dataArray [7] == '1')  
Channel7Enabled = true;  
  
if (dataArray [8] == '1')  
Channel8Enabled = true;  
  
if (dataArray [9] == '1')  
Channel9Enabled = true;  
  
if (dataArray [10] == '1')  
Channel10Enabled = true;  
  
if (dataArray [11] == '1')
```

```

Channel11Enabled = true;

if (DataArray [12] == '1')
{
    Channel12Enabled = true;
    //Serial1.write ('C'); //debugging
}
}

else if (DataArray [0] == 'X' && DataArray [1] == '+' && ReadSerial)
{
    ReadSerial = false;
    SystemEnable = false;
    Channel1Enabled = false;
    Channel2Enabled = false;
    Channel3Enabled = false;
    Channel4Enabled = false;
    Channel5Enabled = false;
    Channel6Enabled = false;
    Channel7Enabled = false;
    Channel8Enabled = false;
    Channel9Enabled = false;
    Channel10Enabled = false;
    Channel11Enabled = false;
    Channel12Enabled = false;
    //Serial1.write ('S'); //debugging
}
}

```

```

void set-up()
{
  // set-up code , to run once:
  Serial1.begin(115200);    //open com at baud rate of 115200
  pinMode(SystemAliveLed, OUTPUT);
}

void loop()
{
  // main code to run repeatedly:
  digitalWrite(SystemAliveLed, LOW);    //system alive

  for(byte i=2; i < 26; i++)
  {
    ADCdataArray [i] = 0; //reset buffer so no chunk is sent to the DAS Host System
  }

  ReadSerialPort();    //read commnds

  if (SystemEnable)
  {
    if (Channel1Enabled)    //if channel is selected start data collection of data
    {
      ADCData = analogRead(analogPin0);
      ADCdataArray [2] = char(lowByte(ADCData));
      ADCdataArray [3] = char(highByte(ADCData));
    }

    if (Channel2Enabled)
    {

```

```

    ADCData = analogRead(analogPin1);
    ADCdataArray [4] = char(lowByte(ADCData));
    ADCdataArray [5] = char(highByte(ADCData));
}

if (Channel3Enabled)
{
    ADCData = analogRead(analogPin2);
    ADCdataArray [6] = char(lowByte(ADCData));
    ADCdataArray [7] = char(highByte(ADCData));
}

if (Channel4Enabled)
{
    ADCData = analogRead(analogPin3);
    ADCdataArray [8] = char(lowByte(ADCData));
    ADCdataArray [9] = char(highByte(ADCData));
}

if (Channel5Enabled)
{
    ADCData = analogRead(analogPin4);
    ADCdataArray [10] = char(lowByte(ADCData));
    ADCdataArray [11] = char(highByte(ADCData));
}

if (Channel6Enabled)
{
    ADCData = analogRead(analogPin5);
    ADCdataArray [12] = char(lowByte(ADCData));
    ADCdataArray [13] = char(highByte(ADCData));
}

```

```

}

if (Channel7Enabled)
{
    ADCData = analogRead(analogPin6);
    ADCdataArray [14] = char(lowByte(ADCData));
    ADCdataArray [15] = char(highByte(ADCData));
}

if (Channel8Enabled)
{
    ADCData = analogRead(analogPin7);
    ADCdataArray [16] = char(lowByte(ADCData));
    ADCdataArray [17] = char(highByte(ADCData));
}

if (Channel9Enabled)
{
    ADCData = analogRead(analogPin8);
    ADCdataArray [18] = char(lowByte(ADCData));
    ADCdataArray [19] = char(highByte(ADCData));
}

if (Channel10Enabled)
{
    ADCData = analogRead(analogPin9);
    ADCdataArray [20] = char(lowByte(ADCData));
    ADCdataArray [21] = char(highByte(ADCData));
}

if (Channel11Enabled)

```

```

    {
        ADCData = analogRead(analogPin10);
        ADCdataArray [22] = char(lowByte(ADCData));
        ADCdataArray [23] = char(highByte(ADCData));

    }

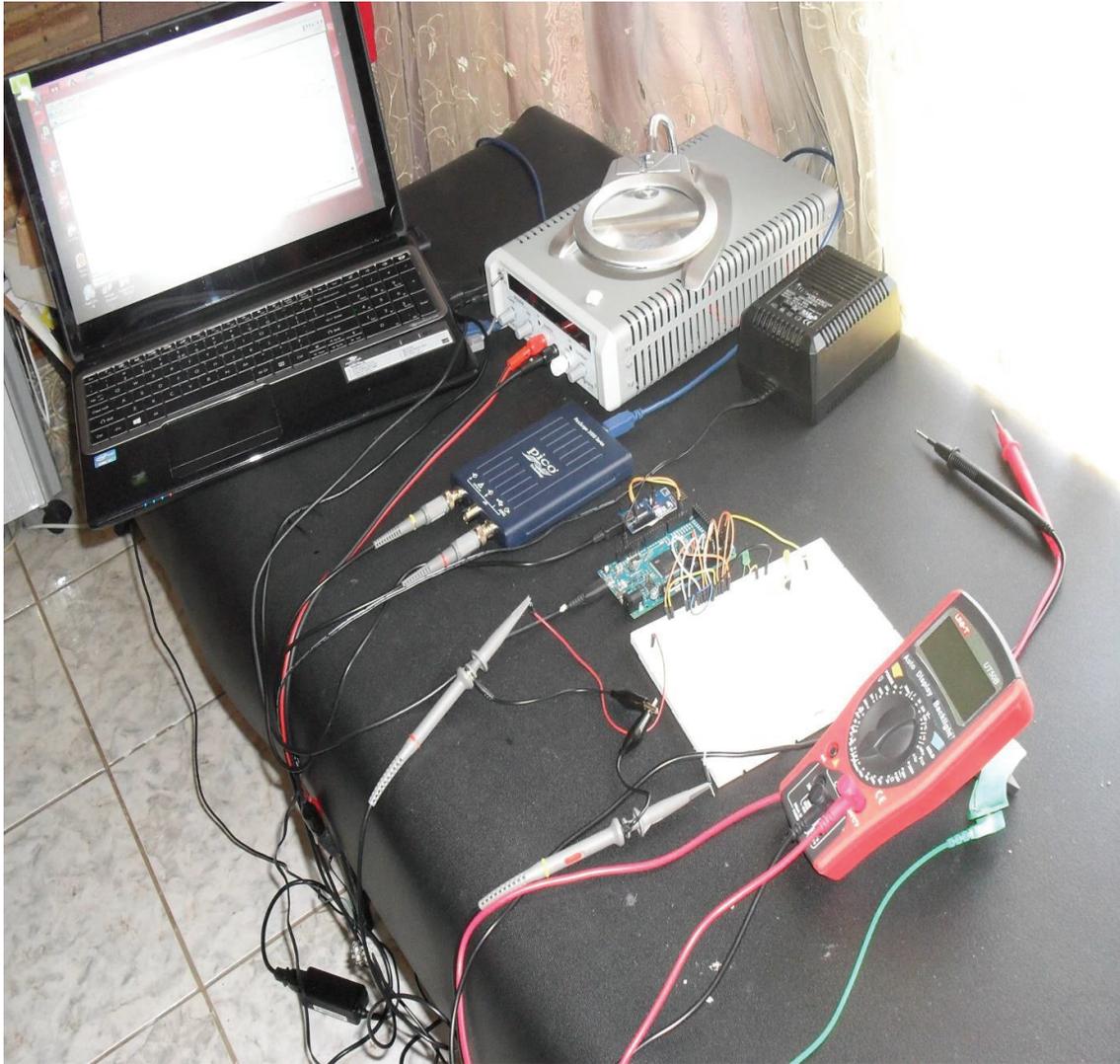
    if (Channel12Enabled)
    {
        ADCData = analogRead(analogPin11);
        ADCdataArray [24] = char(lowByte(ADCData));
        ADCdataArray [25] = char(highByte(ADCData));
    }
    //stream data
    Serial1.write(ADCdataArray, 27);
}

digitalWrite(SystemAliveLed, HIGH); //system alive
}

```

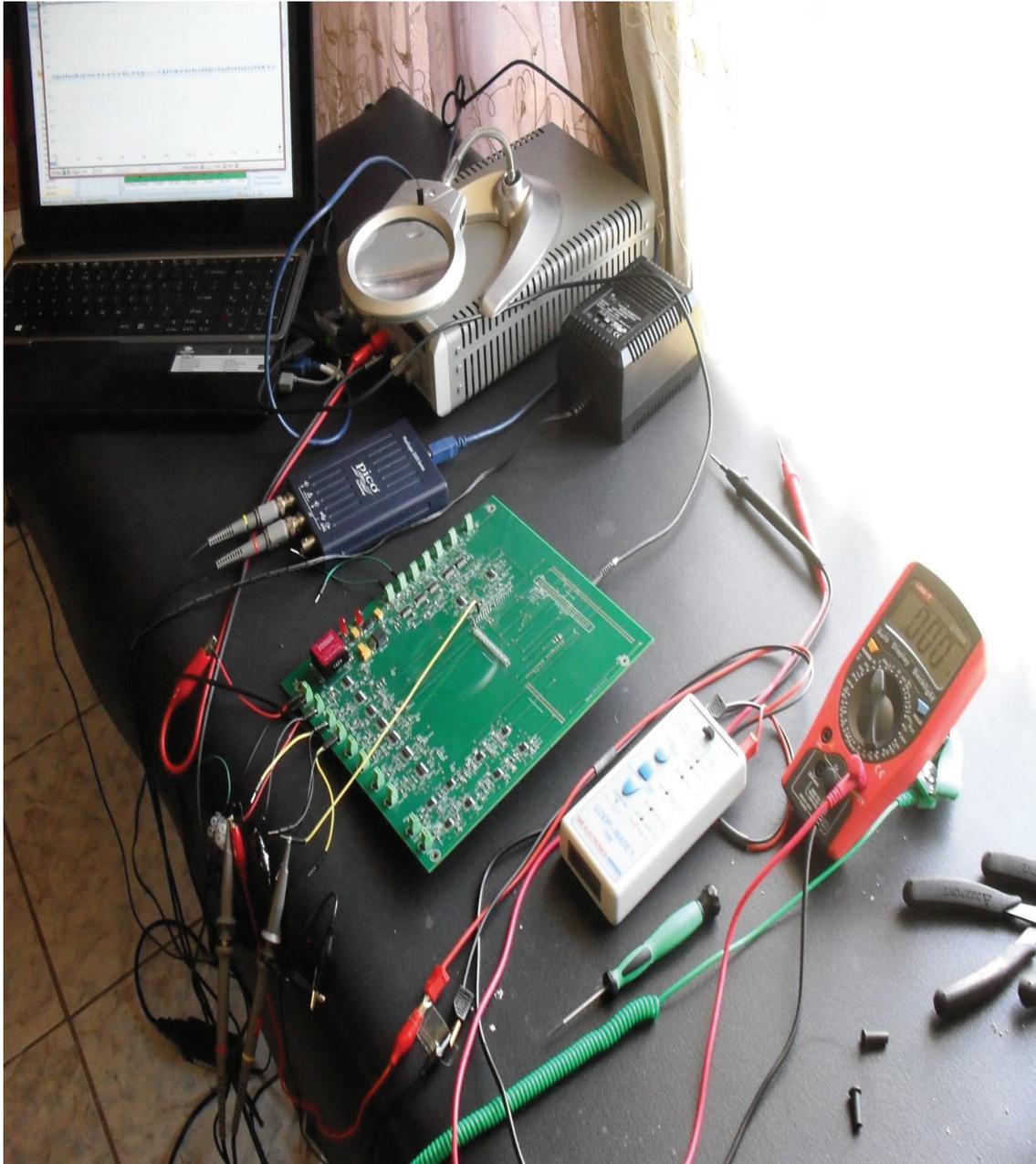
Appendix K Arduino Due adc test set-up

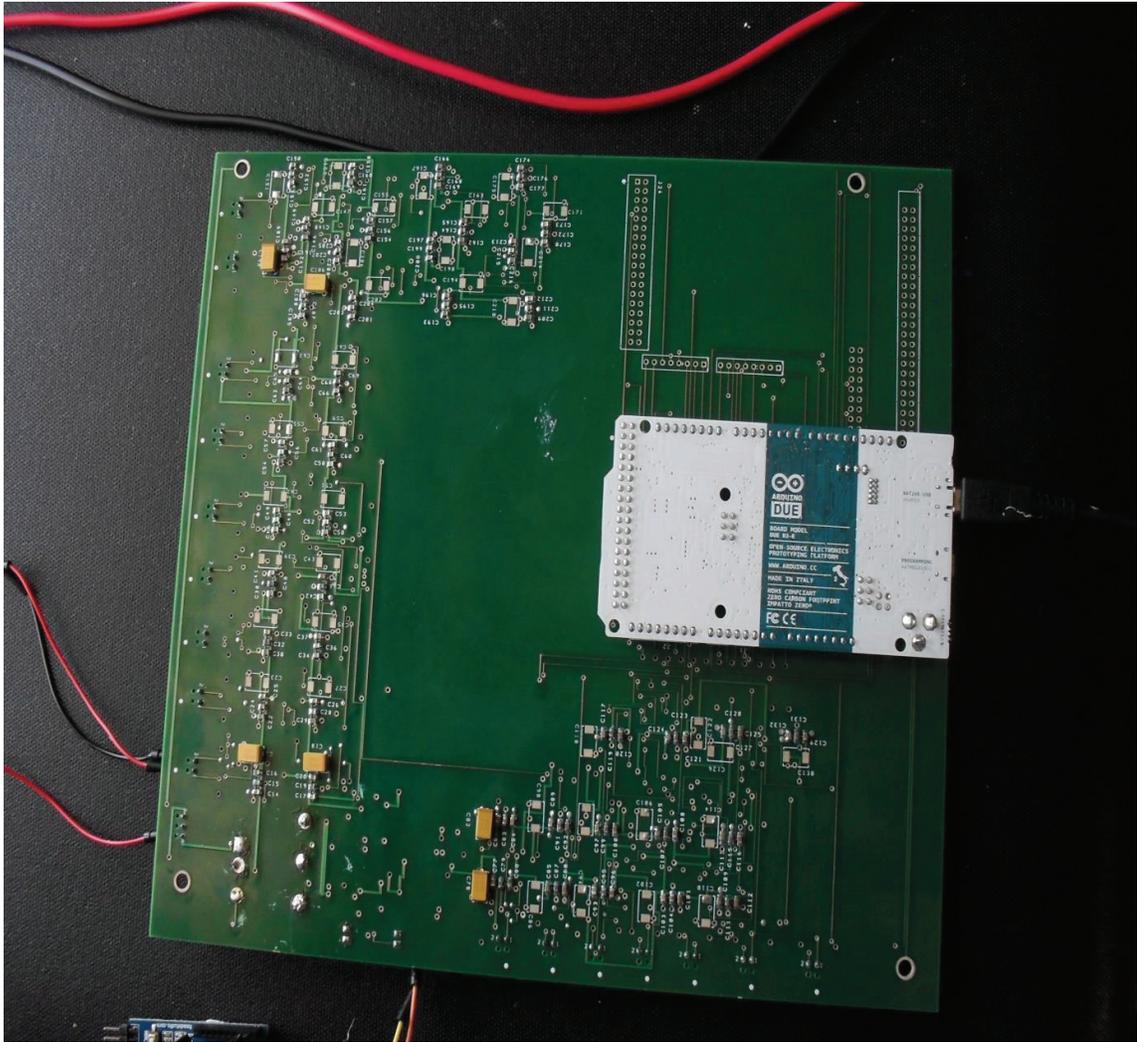
Arduino Due ADC Subsystem Test Set-up



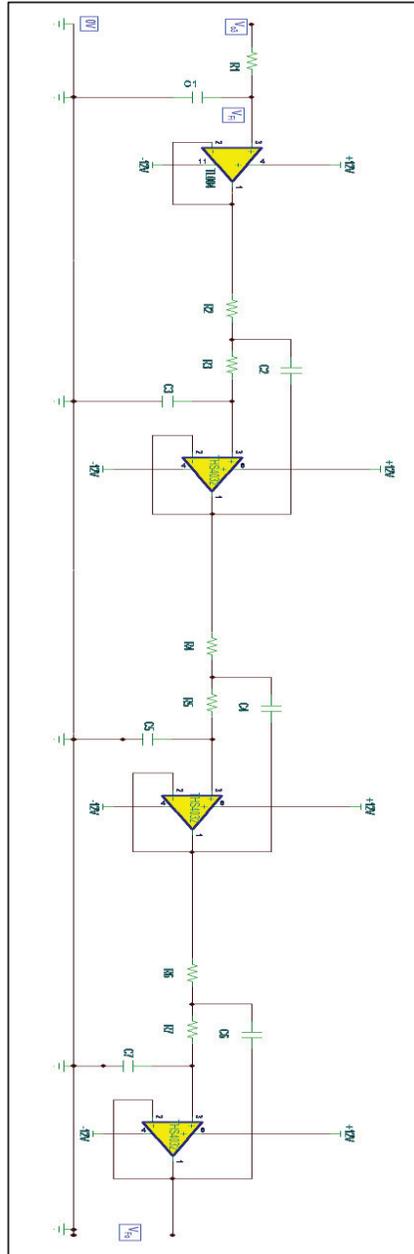
Appendix L DAS System Calibration Set-up

DAS System Calibration Set-up



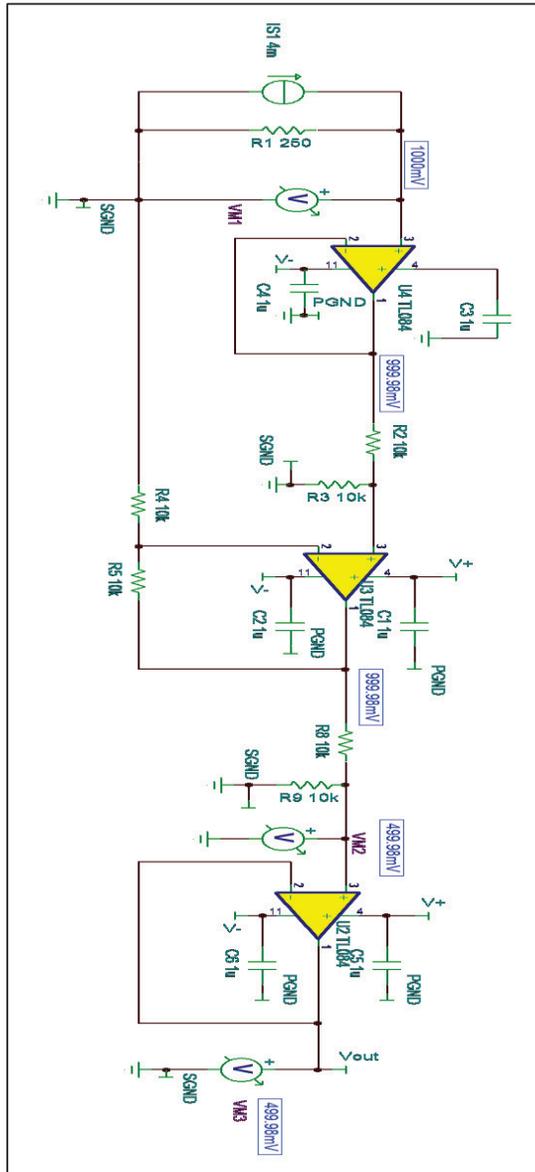


Appendix M Seven-order Lowpass Filter for 50kHz Channels



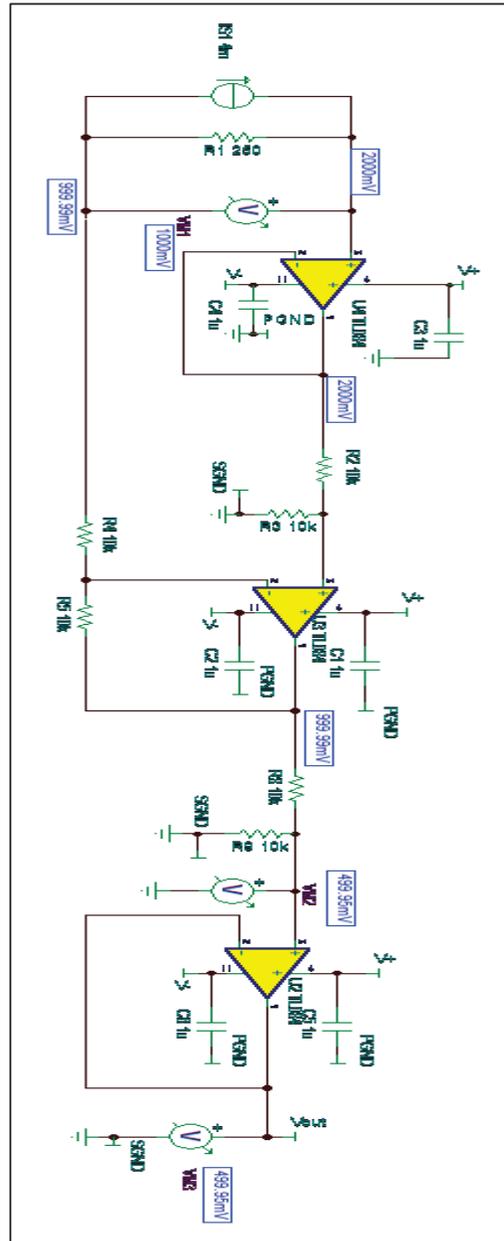
Appendix N 4-20mA Input, Differential and Scaling Stages Cascaded

5kHz Channels (MINUS Input Grounded)



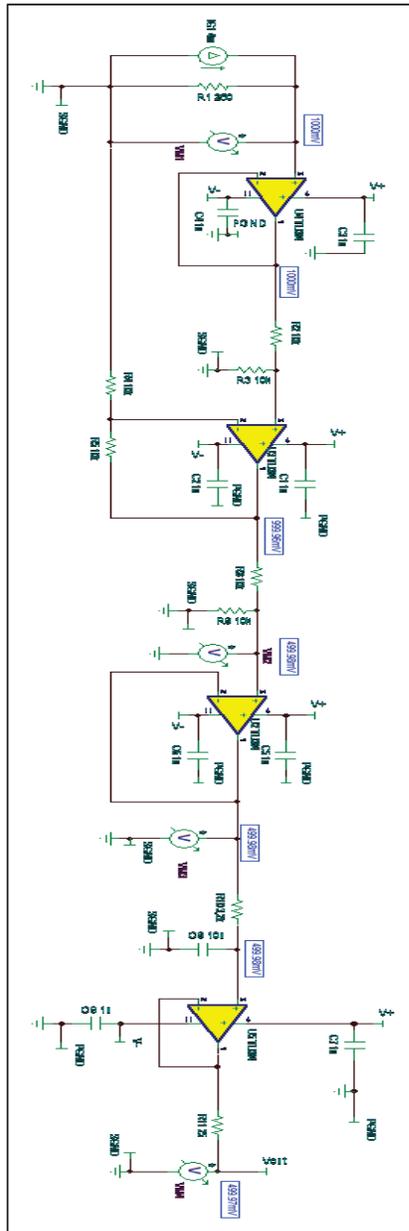
Appendix O 4-20mA Input, Differential and Scaling Stages Cascaded

5kHz Channels (differential input)



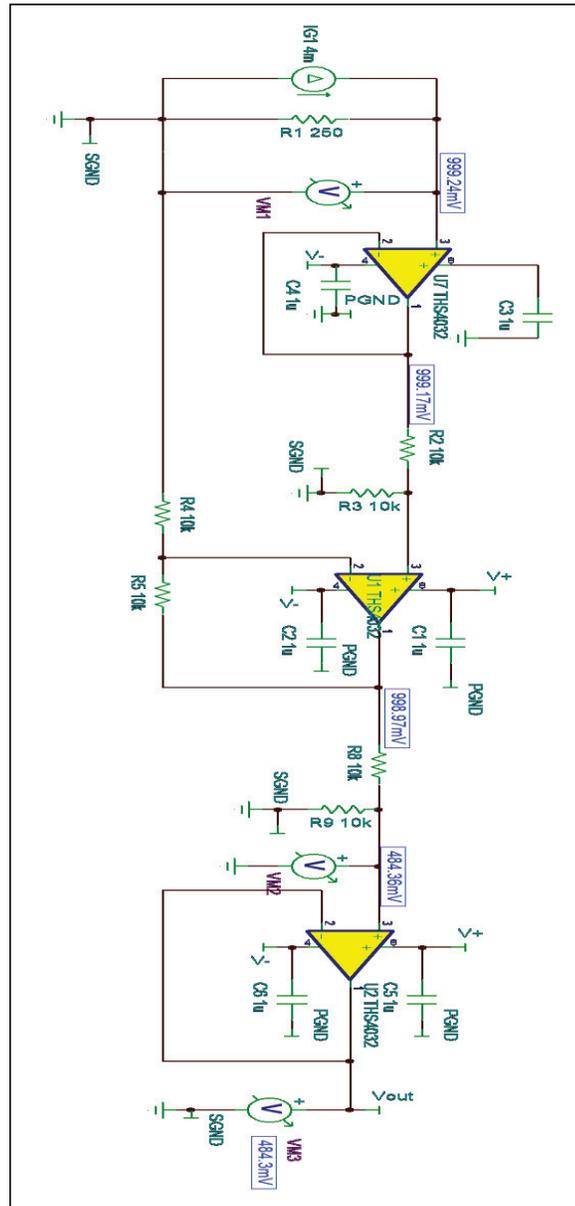
Appendix P Input, Differential, Scaling and Filter Stages Cascaded

5kHz Channels



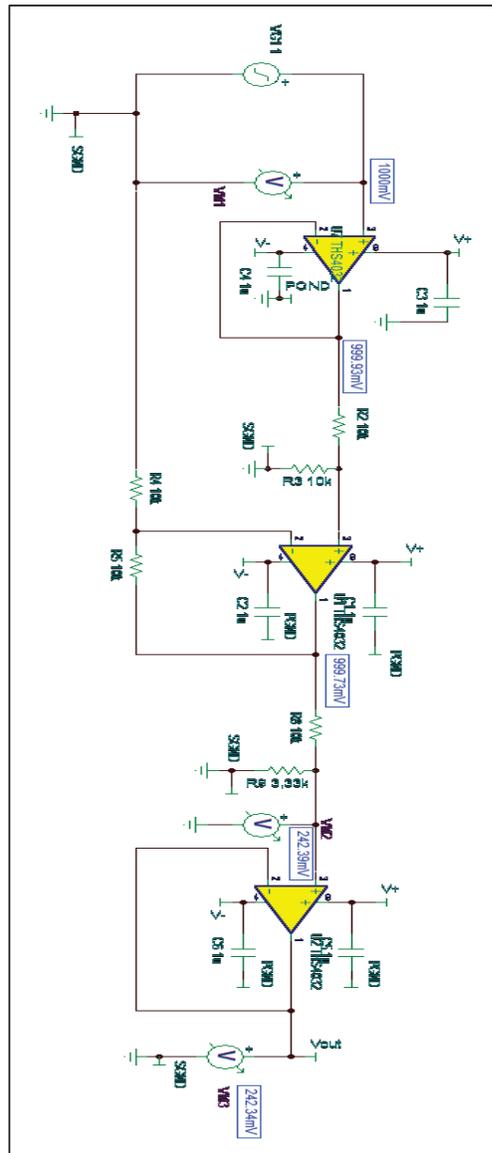
Appendix Q Input, Differential and Scaling Stages Cascaded 50kHz

Channel with 4-20mA Input Signal (Minus input grounded)



Appendix R Input, Differential and Scaling Stages Cascaded 50kHz

Channel with 0-10V Input Signal (minus input grounded)



Appendix T Some of the popular microprocessors and microcontrollers

Some of the popular microprocessors and microcontrollers

Adapted from: Dobioli and Currie (2011)

| Model | Speed | On-chip memory (Flash and RAM) | Interfacing | Data width | Manufacture | Notes |
|------------|------------------|--------------------------------|---|------------|-------------------|--|
| ARM 7 | 90 MIPS @ 100MHz | 4G addressable | - | 32 bits | ARM | Core used by vendors to implement their own μ C or μ P. |
| DSP56800 | 35 MIPS @ 70MHz | 28k-152k | Serial, CAN | 16 bits | Freescale | www.freescale.com |
| PIC 24FJXX | 16 MIPS @ 32MHz | 64k-128k Flash, 8192 bytes RAM | SPI, UART, I2C, parallel | 16 bits | Microchip | www.microchip.com |
| MAXQ3120 | 8 MIPS @ 8MHz | 16k Flash, 256 words RAM | I/Os, LCD driver, UART, ADCs | 16 bits | Maxim | www.maximintegrated.com |
| MSP430 | 8 MIPS @ 16MHz | 2k Flash, 128 bytes RAM | Universal Serial Interface, SPI, I2C, ADC, I/Os | 16 bits | Texas Instruments | www.ti.com |

| Model | Speed | On-chip memory (Flash and RAM) | Interfacing | Data width | Manufacture | Notes |
|--------------|--------------------------------------|--------------------------------|--------------------------------|------------|-------------|-----------------|
| MCS51 (8051) | 1 MIPS @ 12MHZ | 4k bytes ROM, 128 bytes RAM | Full-duplex serial | 8 bits | Intel | www.intel.com |
| 80C552 | 1.36 MIPS @ 24MHz, 0.91 MIPS @ 16MHz | 8k bytes ROM, 256 bytes RAM | UART, I2C, parallel I/Os | 8 bits | Philips | www.nxp.com |
| PSoC | 24MHz | 64k bytes Flash, 2k RAM | SPI, UART, I2C, PWM, I/Os, ADC | 8 bits | Cypress | www.cypress.com |

Appendix U Design Notes

Resistor Values Calculations

The output impedance of the buffer input stage (Figure 5.2) is estimated to be $Z_{oB} = 0.213\Omega$, which is calculated using equations (5.2) and (5.3). This value is only applicable when using TL084BC op-amp. This will require the next stage to have a minimum input impedance equal to $Z_{oB} * 1000 = 213\Omega$. With reference to the differential amplifier stage (Figure 5.3), the value of R_1 , R_2 , R_3 and R_4 is chosen to be $10k\Omega$. The output of this stage is estimated to be $Z_{oDS} = 0.427\Omega$, calculated using equations (5.3) and (5.7). The next stage, which is the signal-scaling stage (Figure 5.4), must have a minimum input impedance equal to $Z_{oDS} * 1000 = 427\Omega$. With reference to Figure 5.4 and assuming the signal is scaled by a factor of 0.5, choosing $R_1 = 10k\Omega$ and calculating for R_2 results in $R_2 = 10k\Omega$, using equation (5.8). This assumption is only applicable to 4-20mA signals. However, for 0-10V signals, it is assumed that the signal is scaled by a factor of 0.25, choosing $R_1 = 10k\Omega$ and calculating for R_2 , which is equal to $R_2 = 3.333k\Omega$.

The output impedance of the signal-scaling stage is the same as the buffer stage, thus equal to $Z_{oB} = 0.213\Omega$. This stage is connected to the lowpass filter stage, with an input impedance of $R_1 = 3.2 k\Omega$ (Figure 5.5). It is the last stage on the 4-20mA and 0-10V channels signal chain and has the same output impedance as the buffer stage ($Z_{oB} = 0.213\Omega$).

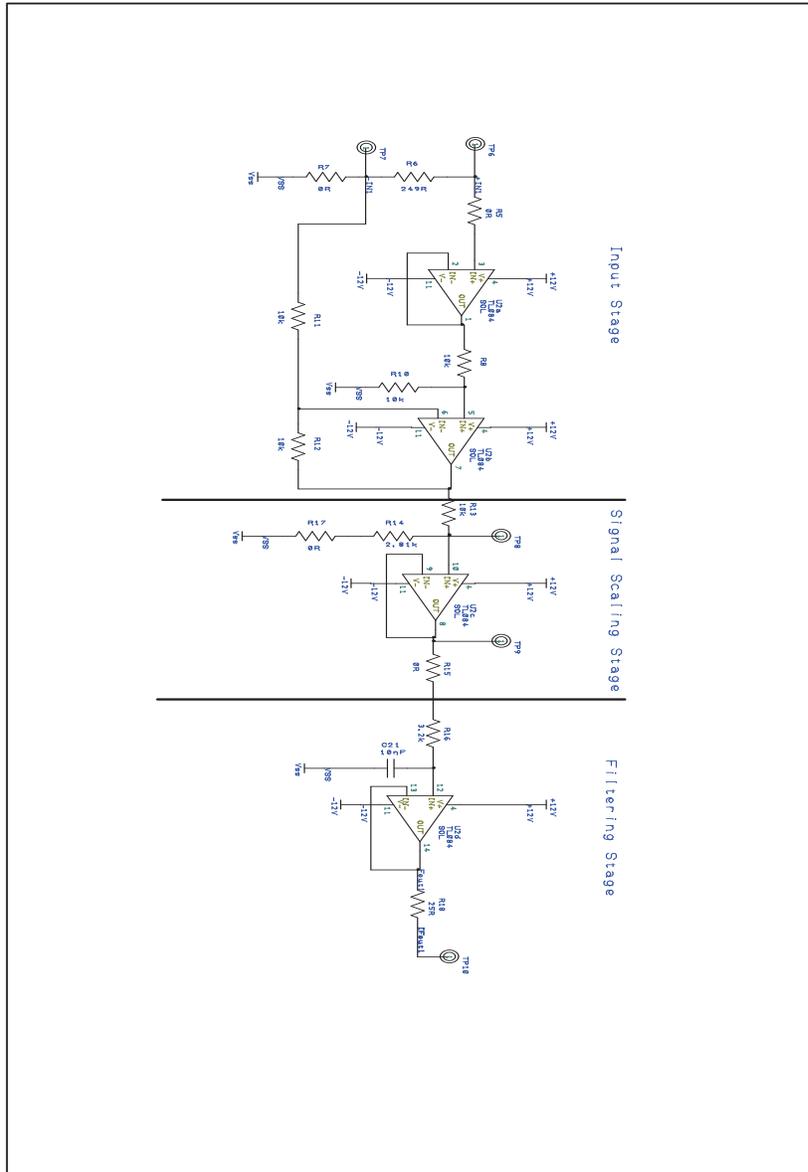
The 50kHz channel will use the THS4032 op-amp instead of the TL084BC, due to its high-gain bandwidth product (100MHz). In this case, output source impedance of the buffer input stage (Figure 5.2) is estimated to be $Z_{oB} = 0.064\Omega$. This is quite low; hence, it is good for the next stage. The component values obtained above for Figure 5.2, Figure 5.3 and Figure 5.4 are also applicable to the 50kHz signal channel. However, for the filter implementation, the lowest resistor value obtained using FilterPro application is $R_6 = 619 \Omega$ (see Appendix M). This value is approximately nine thousand six hundred (9 600) more than the source impedances ($Z_{oB} = 0.064\Omega$) of the buffer stage. Therefore, it is high enough not to load the previous stages.

Appendix V ADC Resolution

ADC Resolution

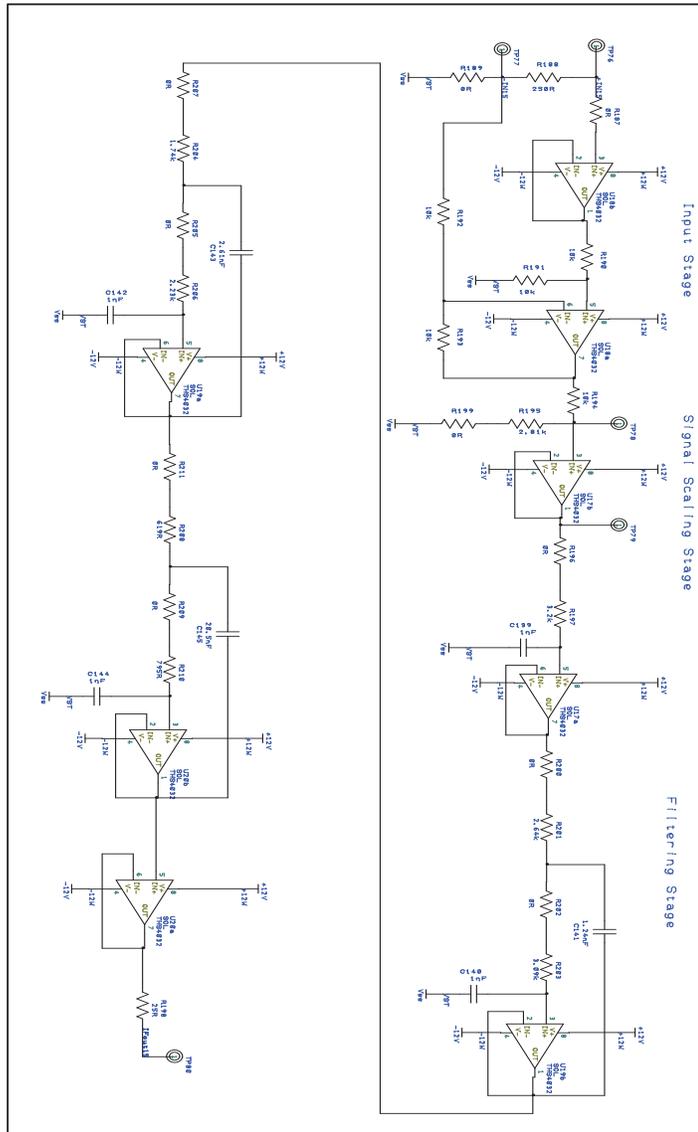
| Analogue Signal Range | Scale Factor | ADC Input Signal Range | NO Bits | Reference Voltage | Resolution |
|-----------------------|--------------|------------------------|---------|-------------------|------------------|
| 1V – 5V | 0.5 | 500mV – 2.5V | 10 | 2.56V | 2.5mV |
| 1V – 5V | 0.6 | 600mV – 3V | 10 | 3V | 2.9297mV |
| 1V – 5V | 0.66 | 660mV – 3.3V | 10 | 3.3V | 3.2227mV |
| 1V – 5V | 1 | 1V – 5V | 10 | 5V | 4.8828mV |
| 1V – 5V | 0.5 | 500mV – 2.5V | 12 | 2.5V | 625 μ V |
| 1V – 5V | 0.6 | 600mV – 3V | 12 | 3V | 732.4219 μ V |
| 1V – 5V | 0.66 | 660mV – 3.3V | 12 | 3.3V | 805.6641 μ V |
| 1V – 5V | 1 | 1V – 5V | 12 | 5 | 1.2207mV |
| 0V – 10V | 0.25 | 0V – 2.5V | 10 | 2.56V | 2.5mV |
| 0V – 10V | 0.3 | 0V – 3V | 10 | 3V | 2.9297mV |
| 0V – 10V | 0.33 | 0V – 3.3V | 10 | 3.3V | 3.2227mV |
| 0V – 10V | 0.5 | 0V – 5V | 10 | 5V | 4.8828mV |
| 0V – 10V | 0.25 | 0V – 2.5V | 12 | 2.5V | 625 μ V |
| 0V – 10V | 0.3 | 0V – 3V | 12 | 3V | 732.4219 μ V |
| 0V – 10V | 0.33 | 0V – 3.3V | 12 | 3.3V | 805.6641 μ V |
| 0V – 10V | 0.5 | 0V – 5V | 12 | 5 | 1.2207mV |

Appendix W 5kHz Channel Implementation



Appendix X 50kHz Channel Implementation

50kHz Channel Implementation



Appendix Y PSU Implementation

PSU Implementation

