Placement Report

Student Engineer, Cyber Team
ZF Friedrichshafen AG
01/09/25 – 31/08/26







Phone numbers, addresses and supervisors name/signatures etc. removed to put on my personal site. I doubt it matters but now it definitely doesn't.

Table of Contents

1. Technical Section 3
   1.1 The Company 3
   1.2 My Role 3
   1.3 My Work 3
      1.3.1 PyDownloader 3
      1.3.2 TARA RRR generation 4
   1.4 DataLogger Student Project 5
      1.4.1 Project brief 5
      1.4.2 Planning 5
      1.4.3 Implementation 7
      1.4.4 Documentation 10
      1.4.4.1 Future Improvements 11
      1.4.5 Presenting 11
   1.5 Other 11
   1.6 SFIA8 Skills 12
   1.6.1 Programming/Software Development Level 3 13
   1.6.2 Testing Level 3 13
   1.6.3 Project Management Level 4 13
2. Reflective Section 14
3. Acronyms 15
4. References 15

1. Technical Section

Relevant, usually automotive related acronyms are defined in Section 3.

1.1 The Company

For my placement year I worked at ZF Friedrichshafen AG [1], also known as ZF Group, at their technical centre in Birmingham. Founded by the inventor of the zeppelin airship, ZF is now a large manufacturer in the automotive sector supplying products such as powertrains and chassis for passenger and commercial vehicles.

1.2 My Role

As a ‘Student Engineer’ at ZF I worked within the cyber security team, writing code mainly for internal tooling alongside another placement student. We attended cyber team meetings and daily standups to talk about how our work was progressing with those on site and also those from cyber in India.

On Fridays, which were also a half day, I would work with all three other students on one long project throughout the year. This allowed me experience work at scale, with us going through all the stages that a real project at a company would. This involved extensive planning and meeting many people across the whole site to understand their requirements for the project, along with help we received at all stages, through to implementation and presenting our work at the end of the project stakeholders.

1.3 My Work

My main 2 ‘day job’ tasks within cyber were working on PyDownloader, a tool to allow the flashing of bootloaders onto test rig ECUs via UDS, and TARA RRR generation, a script which intends to reduce the time taken to create a Residual Risk Report from outputted XML files.

1.3.1 PyDownloader

PyDownloader is an internal ZF tool used for executing custom UDS sequences over a CAN bus, and for downloading firmware such as a bootloader onto ECUs. This is designed for integration with EXAM testing workflows, which allow for further automating the test pipelines for new ECU bootloader versions. On site, we have test rig racks full of mainly steering and braking ECUs connected to a computer. Throughout this work, I would remotely access these rigs, and a mock development rig, to test PyDownloader could send and receive UDS messages with the real hardware.

Coming into this work previous students had been working on an old version of PyDownloader, named ‘JamesBrownDownloader’ after a placement student from a few years ago who started work on it. Alongside this, work had begun on a fully new version which we would be working on from start through its to first release near the end of the placement. The goal behind the new version was, other than continuing feature development, to give a new structure to it tool inside a proper ZF tooling repository and DevOps [2] pipelines. Still written in Python with tkinter, we did get the opportunity to use newer tools like uv [3] for dependency management and moon [4] alongside a robust testing strategy.

After handover with last year’s placement students and reading the code and documentation, my real first work on PyDownloader was based around UDS commands. Using the udsoncan python

library and an internal addon for it to work best with PyDl, this would allow the tool to create a successful connection and send specific commands, such as disabling and enabling Diagnostic Session Control for flashing purposes. Challenges arose with this around supporting both Vector [5] and PCAN adapters [6], which needed to be handled differently to interface correctly with udsoncan. Learning for this system afforded me the pleasure to read 'ISO 14229-1, Road Vehicles - Unified Diagnostic Services (UDS), 2020-02 Third Edition' [7] to understand, among others, specific UDS message frames with their service and subfunction ids.

Another feature I worked on was config handling for the application, which grew in scope as PyDl feature implementation did too. This stored ordered UDS commands for specific sequences, and all parameters needed for setting up CAN and ISOTP connections. Json was chosen to format this data due to the number of variables changing and for readability during testing the system, with an overall config manager to interface with this, and a json schema for enforcing the format.

PyDownloader followed a Behaviour-Driven Development approach, using Gherkin feature definitions and pytest fixtures to implement these. This was a new experience compared to simpler TDD unit testing I had done previously at university and gave a much deeper look into testing and validation in the long-term compared to a personal project or short coursework. Getting sufficient coverage while being concise with writing tests is very important with Gherkin features as if miswritten can cause many redundant tests to be written. With BDD being new to the whole team it became a very collaborative experience all learning together.

After PyDownloader's first release, it was able to be installed on all our test rigs for use with the Cyber team. Mentioned before, these rigs are composed of various steering and braking ECUs for test purposes, setup with either a Vector or PCAN connection and a debugger like a MiniWiggler [8] to a mini-PC for the team to remote into for their work. This grew throughout the year with more systems being setup and then moved into newer server rack type setups (which was a simple job given I created a spreadsheet with all our rigs and which ECUs they contain).

1.3.2 TARA RRR generation

Coming again with a relevant standards document 'ISO 21434, Road Vehicles — Cybersecurity engineering, 2021-08 First Edition', [9] I worked on a long pre-existing script to parse ITEMis Secure XSAM into an RRR excel spreadsheet. Currently a very manual task, this aims to bridge the gap for automatic RRR generation working upon code created by the TARA team and previous placement students.

TARA is a risk management framework followed by these processes at ZF to look at handling intentional attacks on systems. After understanding the current system, my work centred on calculating residual risk scores directly from .mps files. instead of relying upon the full TARA library. This expands the current, also Python, script to calculate scores and assign residual treatments of 'accepting', 'sharing' or reducing' to specific threat scenarios.

I found this very difficult initially, with understanding the current codebase alongside TARA as a new process to me. However, this did provide many opportunities to speak about and patch holes in my understanding with those more experienced than me on this task. I would be able to ask questions, or go about it the other way around and try to explain my current understanding to get corrections and confidence in what I was working on.

1.4 DataLogger Student Project

On Fridays I would work with all the other placement students on a 'Multi-Lane High Throughput Data Logger' for logging live data from ECUs, lasting throughout the whole year.

Although Fridays are a half day at ZF, for the final month of the placement we had all our time dedicated to this project. Due to time constraints with holidays around the summer and attending our UKESF professional development course among other distractions, this was seen as more important than our day job tasks. This allowed us time to finish the project up to where we could present to teams on site and have documentation complete for handover once we left.

1.4.1 Project brief

To begin this project, a very wide brief was given to ‘develop a data logging solution that supports system development and condition monitoring, with a particular focus on multi-lane systems. The logger will be built on existing ZF electronics hardware and will unify data at a high acquisition rate into a common, standardised format for streamlined analysis and diagnostics.’. This, alongside a longer but equally broad scope document, started the project off with meeting with many teams across the site to determine which requirements should be our focus to be of most use.

1.4.2 Planning

Working on a project like this gave a novel experience for work all the way start to finish which isn't common in academic coursework usually. Lots of time was spent on the scope, planning, requirements tables and many flowcharts before implementation. We presented these back and forth with teams on site, and mainly our project supervisors for feedback and for experience sharing our work in this manner. Organising meeting times and agendas around getting this project complete, and the availabilities of everyone else was something that improved massively as the project progressed.

These documents include: our complete scope document, management tracking sheet, requirements table alongside linked flowcharts, impact analysis, state and interaction diagrams, wiring schematic, 3d box designs and redundancy sheet for available hardware.

Key documents are highlighted below:

1.4.2.1. Scope Document

After speaking to potential stakeholders on site, this list of project objectives was arrived upon, focused on features such as: multi lane support, with 4 being required for Steer by Wire systems; handling high throughput of data even if lanes must be limited to allow this; and a mode to handle data at high speed, up to sub 200µs to match a VX1000 (a pre-existing but expensive solution to this task) to identify signals which may last < 1ms. The scope document goes into detail for these and was one of the first main deliverables near the beginning of the year and project.

1.4.2.2. Management and Tracking

A Gantt chart was used for keeping track of key milestones throughout this project, such as presentation days and supervisor meetings alongside our own deadlines set to keep us on track. Although we had no official budget, prices of pre-existing hardware used from site and cost for custom cable components were tracked too.

1.4.2.3. Requirements and Flowcharts

Software development progress and hardware implementation was matched with flowcharts using ID numbers, which then contain fulfilment linking with their respective high and lower-level

requirements. This gave traceability to what was planned and allowed implementation to continue without creating conflicts between how systems were implemented on each end eg. PC GUI <-> intermediary board expecting/sending differing data formats. A high-level flowchart shows the full system, with secondary flowcharts existing for each one to expand upon them. 71 total requirements were created, with 32 hardware and 49 software (some being relevant to both)

1.4.2.4. Interaction Diagram and Custom Cable

The datalogger system was designed around using UART over ASCLIN with a custom-made cable which connects to the Tx and Rx of the board to a JTAG breakout board, then connecting over DAP (being the best available EPP connector we had access to) but also using the ASCLIN protocol instead.

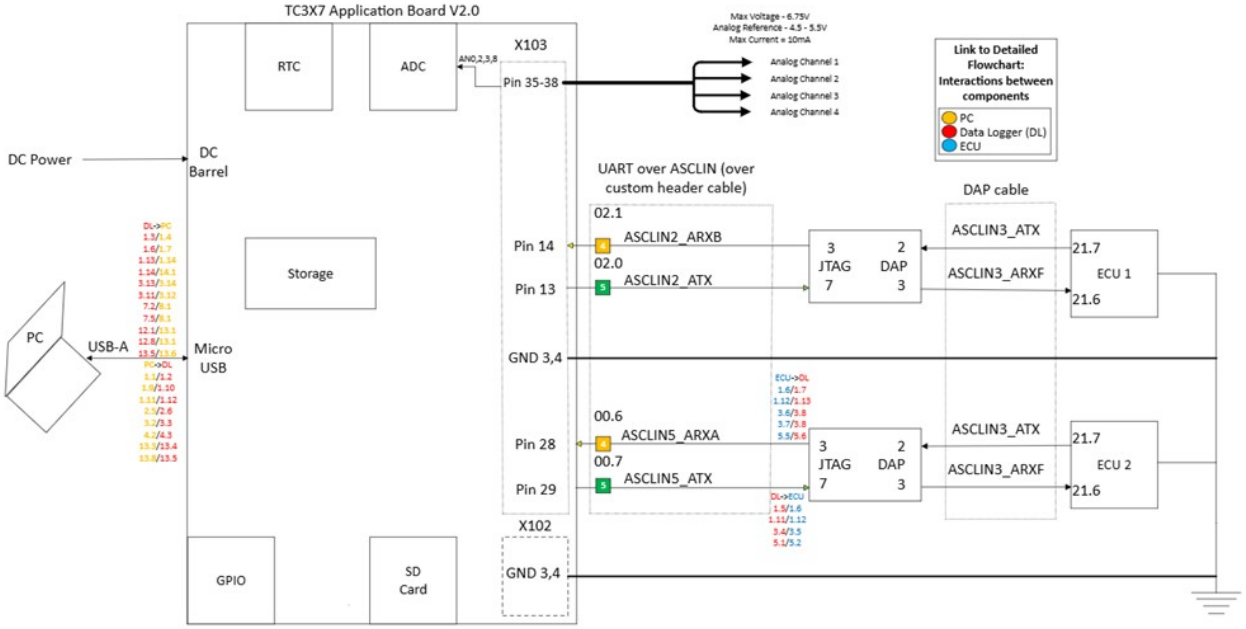


Figure 1: Hardware interaction diagram.

This image shows how the physical components interact with each other, and which pins and ports data is transmitted across. This also shows the theoretical implementation for multiple ECU lanes, spoken more about below in ‘Future Improvements’.

1.4.2.5. 3D Model

For our prototype system, I designed and had a 3D modelled box printed to the size for the intermediary board + JTAG board, with enough height for cables alongside screw holes, and holes for the DAP cable + micro-USB to connect through. The box can be improved in multiple ways (due to the fact I learnt Blender [10] specifically for this) such as including the screw holes in the model itself rather than being done manually and moving the hole for the DAP cable to reach the ECU from the lid to the box allowing for the lid to be removed without unplugging the system.

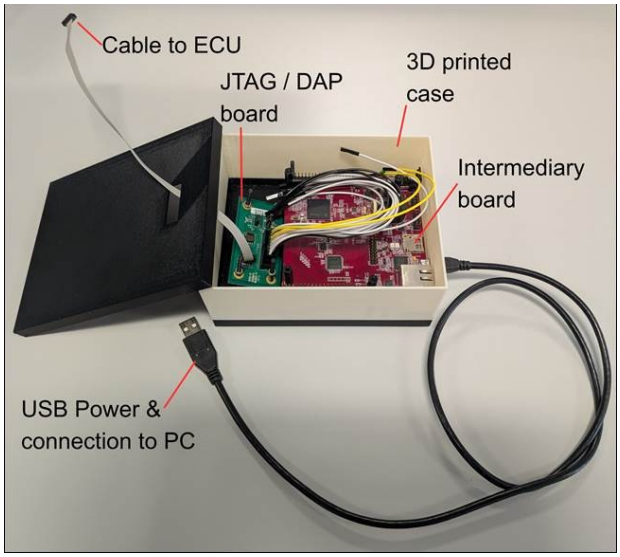


Figure 2: Full prototype DataLogger system setup inside the box.

1.4.3 Implementation

Group work on our project allowed looking at all features of the system while still having more ownership over our personal expertise. With other students studying EEE, my focus was on the PC side GUI code and its connection to our intermediary board. This was written in Python using PyQt library components to build the interface alongside custom made, more complex components. I chose PyQt6, instead of tkinter as used in PyDownloader, because it provided a more modern look for the interface. More importantly, I found PyQt easier to work with and simpler to create my own custom components with. One example of these is a collapsible panel to allow for showing and hiding individual signal graphs, which was implemented with simply a base 'ToolButton' and 'VBoxlayout' widgets with a toggle for showing and hiding content + correcting spacing for other surrounding panels. Components like a searchable checklist (seen in figure 5) allow for displaying, selecting specific items and confirming required more base PyQt widgets but were still built up in the same way.

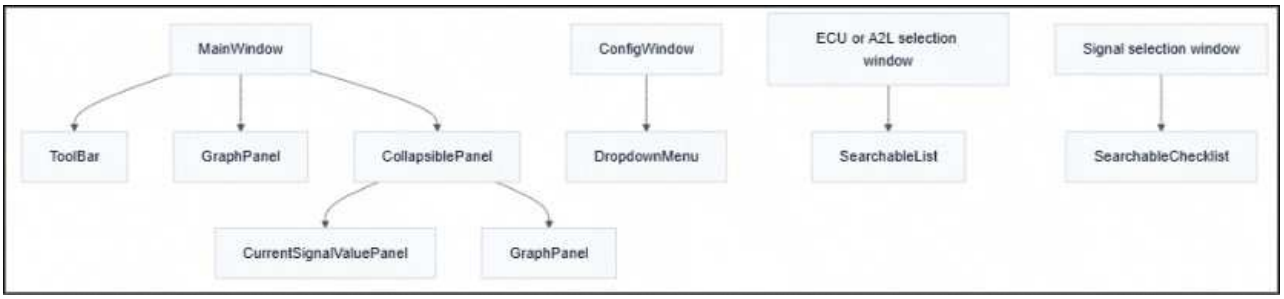


Figure 3: GUI windows and main components involved.

The user interface starts from the main window, showing connection and settings buttons held in the toolbar. Once connected to the DataLogger board, these link to open other windows such as the config window. This allows for baud rate etc. to be controlled and for ECU addresses for logging to be searched through and selected from a provided testbuild A2L file. Once saved this config is stored on the user's computer in AppData for future use of the DataLogger and then sent to the board.

This table below shows all the components for the main areas of the interface and what they do.

Component	Description	Inputs	Outputs

Main Window main.py	Base of the application, handles the interface, connections and DataLogger state changes	Button events, saved config, incoming samples	Commands, panels, graphs, log files
Config Window config_window.py	Logger config and selected signals are here, sending them to the connected logger and reports a success to MainWindow	User selections and existing config	Updated config and config_applied(baudrate) signal
Data Logger datalogger.py	Implements real serial communication with the board	Config and control commands	Parsed data dictionaries or text lines
Mock Data Logger mock_datalogger.py	Implements the same Data Logger interface using a TCP socket	Local mock stream	Mock data values
Base Data Logger base.py	Defines the interface shared by the real and mock loggers	N/A	Logger interface methods
Stream Reader stream_reader.py	Reads samples on a background thread and parses them	Logger stream	Values in SignalStore and optional CSV rows
Signal Store signal_store.py	Holds the latest value and metadata for each signal safely across threads	Parsed signal values	Values read by GUI components
Graph Panel graph_panel.py	Samples the store and plots values to a graph	Signal sources	Live graph plotting
Signal Value Panel current_signal_value_panel.py	Displays the latest signal value	Signal source	Value to be displayed
CSV Log Writer csv_log_writer.py	Writes received sample dictionaries to a timestamped file	Parsed samples	CSV file

These are my custom made PyQt widgets with capabilities not found with PyQt by default as mentioned above. Being their own files allows for importing and reuse as if they were any other widget.

Component	**Description**	**Used by**
Toolbar tool_bar.py	Holds buttons and connection status to the board and ECU	MainWindow
Graph Panel	Plots signals on a graph	The master graph and

graph_panel.py		each signal panel in MainWindow
Signal Display current_signal_value_panel.py	Displays current signal value	Each individual signal panel
Collapsible Panel collapsible_panel.py	Click to hide or show individual signal graphs	MainWindow
Dropdown Menu dropdown_menu.py	Custom dropdown menu component	ConfigWindow
Searchable List searchable_list.py	Displays a list which can be searched to filter	A2L file selection window
Searchable Checklist searchable_checklist.py	Displays a list which can be searched to filter. Items can be selected and then submitted	Signal selection window

When put together The GUI can be used to change config settings and also parse A2L files from the user to select ECU addresses.

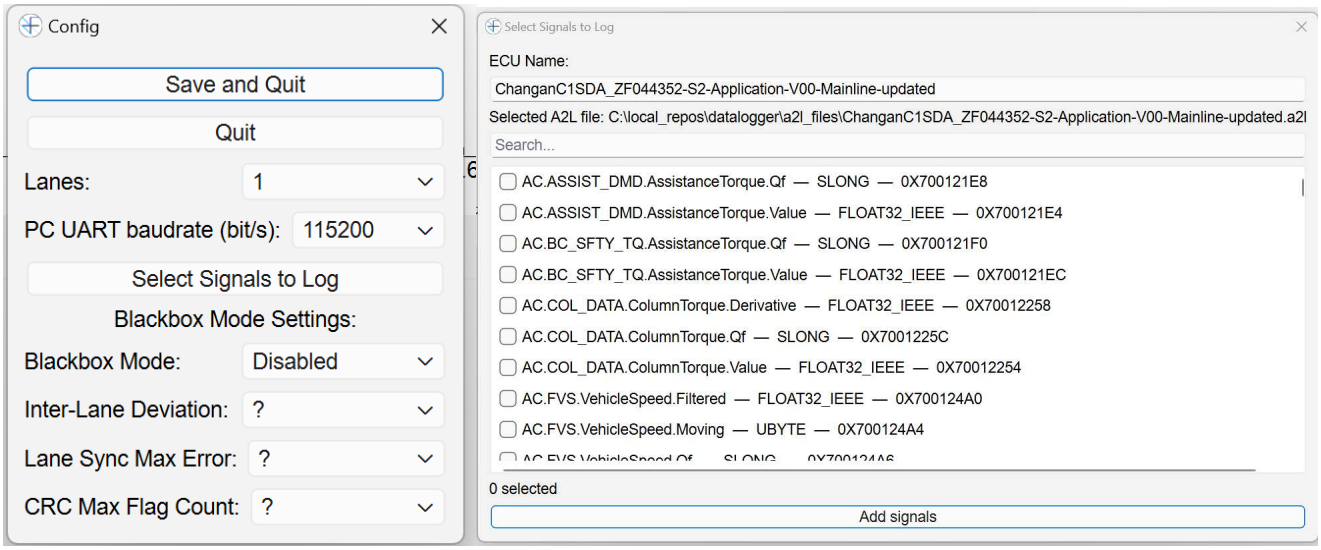


Figure 4: Config settings window.
Figure 5: ECU address selection window, with searchable and scrollable checklist.

Once everything is connected, logging can start. Graph cursors are available to measure exact values along with time width and value change in between. Once logging is stopped, a csv file can be exported for future analysis.

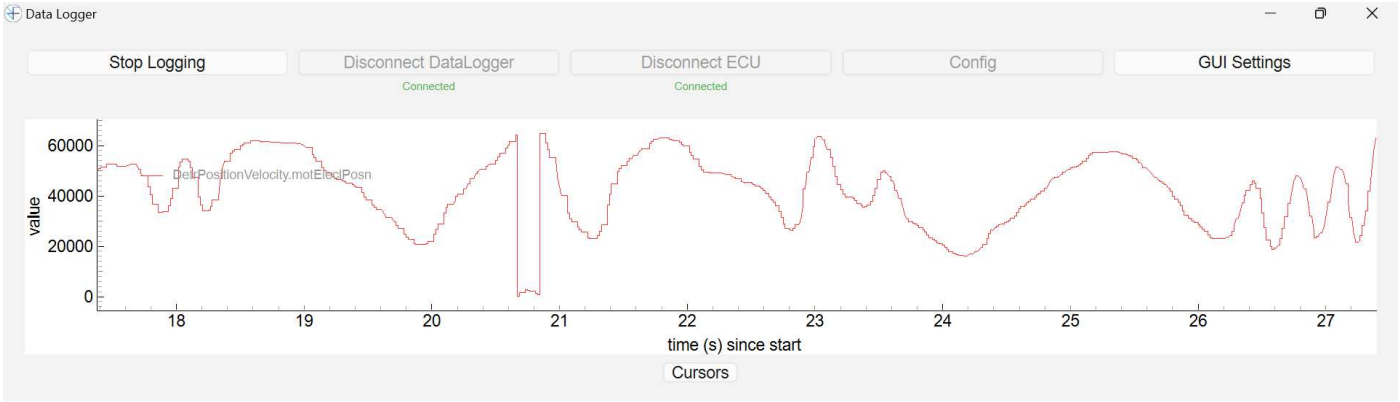


Figure 6: Graph showing motor position data, as it rotates, being plotted live onto the GUI.

While EPP [13] side was untouched by me, I did write some intermediary board C code for sending a mock data stream for testing the GUI connection, and also the handling of correct states with the GUI. To change config settings like baud rate, or to select which signals to log, a ‘signal <address>’ command would be sent to the board while not currently logging to make these config changes.

1.4.4 Documentation

After my placement ended, this project has been handed over to ZF’s Base Software team in India to continue work. I created detailed documentation for both usage and development guidance for the GUI side codebase. This shows how the GUI user flow works through to development setup guide, explanations of the custom components like in this report and notes on future development features with common errors and fixes found in the development of this system.

For code architecture documentation, I created mermaid [11] diagrams to show data flow throughout the interface, and for connection with hardware.

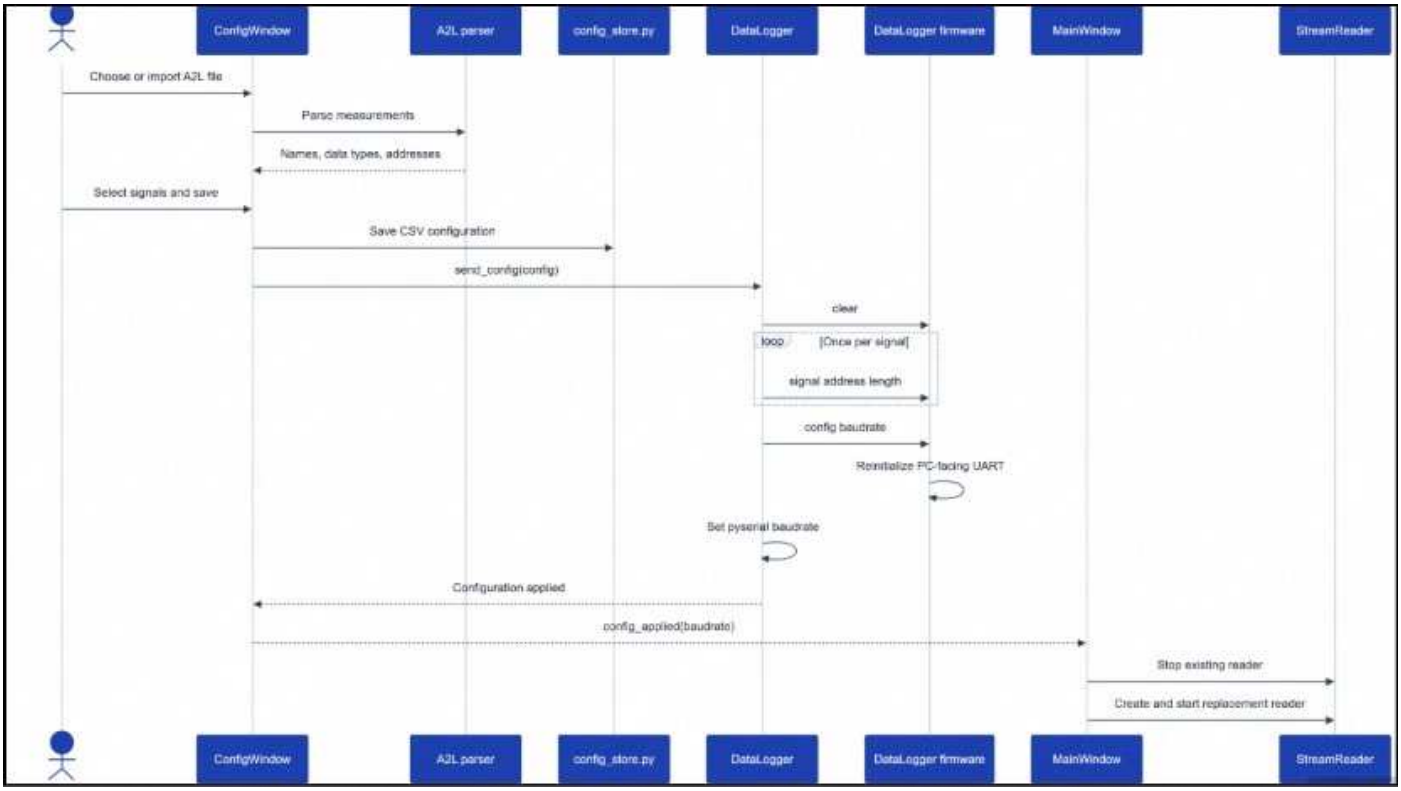


Figure 7: GUI data from interface start.

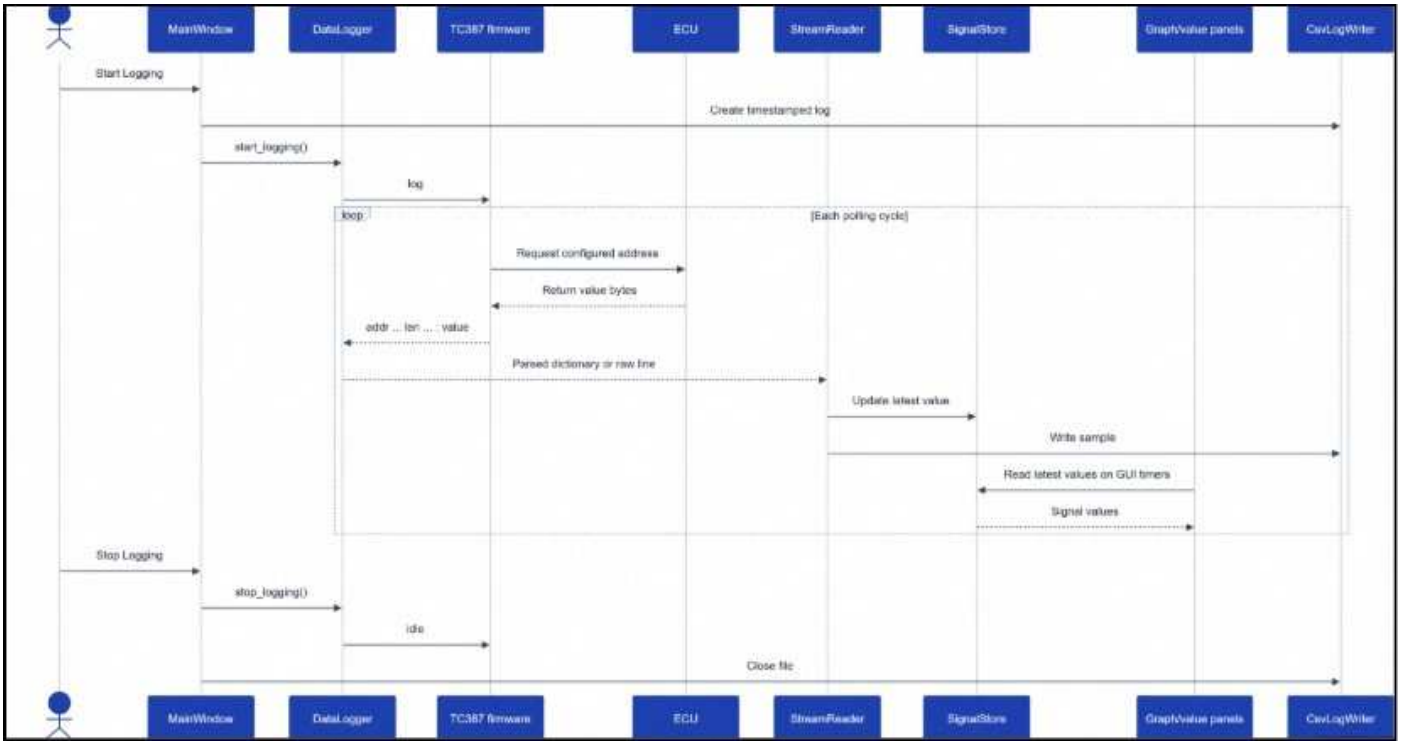


Figure 8: GUI data flow to the DataLogger to obtain data.

1.4.4.1 Future Improvements

Though many capabilities were implemented for the DataLogger, some requests from stakeholders could not be completed and such are handed over to this new team. For example, implementation of multi-lane logging is very possible, with our plans showing how it can be connected, this sort of feature is left for the future. One suggestion we had from multiple teams was for the DataLogger to be able to tun in a ‘black box’ type of mode, saving data anomalies onto an SD card over a long period of time for certain monitoring and debugging tasks. Taking this into account for the system, with the intermediary board fully supporting this and creating flowcharts with both logging modes available, allows for future development by building upon the groundwork created.

1.4.5 Presenting

Late august we presented our project to teams on site, with one more official meeting with stakeholders in the system, our managers and site leads. This was a great opportunity to demonstrate the functionality of the prototype system and answer questions about it before our full demo in ‘The Link’ on site. This was held in the afternoon in a communal area for everyone in office to come see, some but few for the first time, our DataLogger. This, other than showing off our work gave a great excuse to and thank everyone who helped throughout my placement year.

1.5 Other

This section, though less technical, I feel contains equally important parts of my time here. Either being about forming connections with colleagues, other students or people outside ZF, this was the real experience of working in a global company with many people day to day on work and is the main value of a placement. The technical work is important for returning to university, but this experience cannot be gained outside of industry.

Towards the end of the planning period for the DataLogger project, some year 10 students were at ZF for the week for work experience. Other than their questions about A-Levels and university for

both computer science and electronics, we had a day with them one Friday to show our project. The timing of this was good as other than just an explanation of documentation so far, we were able to do some practical activities by designing some draft interfaces for the GUI and to go into a lab and build one of our custom cables. Made simply with a 16-pin connector and cables, they practiced stripping wires and crimping terminals onto them.

Late march, our site held an event organised by the UK Department of Business and Trade with 18 UK companies across cybersecurity, AI, robotics/servo-drives and hydrogen. Alongside the other placement students, I helped tour groups around our site and technical building showing test labs, various machinery and the large EMC chamber. The event concluded in the evening with a more formal chance to speak to everyone, and a speech from the British Consul General for Munich. On similar but smaller scale to this, I also helped with giving tours to members of the British Gear Association. These were great opportunities to speak to those across the automotive industry, and tech overall, to hear about their experience and prediction so the future in this space.

In the summer, ZF held a BBQ summer event on site. This, especially given the heat at the time, was great to speak to everyone in a more casual setting. Some brought in classic cars and bikes they owned, with other stalls setup to get the best time on a Trackmania [12] track.

To end the year, I attended the UKESF [13] Professional Development Course due to their sponsorship of my placement. This was an incredible experience to conclude with, getting to meet the other students and for talks given from guest speakers. Less relevant to this report, this post shows all UKESF students at the event and those speakers who stayed for the formal dinner where I got to speak to some of them further. https://lnkd.in/p/eDbyRVqX

1.6 SFIA8 Skills

START DATE		01/09/25	
EXPECTED DURATION		12 months	
SKILL ACRONYM	**SKILL LEVEL**	**DESCRIPTION**	**NARRATIVE**
PROG	3	Designs, codes, verifies, tests, documents, amends and refactors moderately complex programs/scripts. Applies agreed standards and tools to achieve a well-engineered result. Monitors and reports on progress. Identifies issues related to software development activities. Proposes practical solutions to resolve issues. Collaborates in reviews of work with others as appropriate.	See sections 1.6.1, and 1.3.1, 1.3.2
TEST	3	Designs test cases and test scripts under own direction, mapping back to pre-determined criteria, recording and reporting test outcomes. Participates in requirement, design and specification reviews, and uses this information to design test plans and test conditions. Applies agreed standards to specify and perform manual and automated testing. Automates testing tasks and builds test coverage through existing or new infrastructure.	See sections 1.6.2, 1.3.1

		Analyses and reports on test activities, results, issues and risks.	
PRMG	4	Defines, documents and executes small projects or sub-projects. Works alone or with a small team actively participating in all phases of the project. Applies appropriate project management methods and tools. Identifies, assesses and manages risks effectively. Agrees project approach with stakeholders and prepares realistic project plans (including scope, schedule, quality, risk and communication plans). Tracks activities against the project schedule, managing stakeholder involvement as appropriate. Monitors costs, times, quality and resources used takes action where these exceed agreed tolerances.	See sections 1.6.3, 1.4

1.6.1 Programming/Software Development Level 3

My introductory work into the config handling in PyDownloader and RRR generation for the TARA involved following simpler processes for working in large codebases for the first time. This gave a good starting point for other tasks later.

To progress to Level 3, I worked on more complex functionality and gained deeper understanding of preexisting code. PyDownloader being a full rewrite of an older system showed a deep level into reviewing the code of others, followed by refactoring this for new functionality in the new system. Reading through and understanding how old PyDownloader works gave insight into how codebases become after time and people working within.

1.6.2 Testing Level 3

To reach to Level 3, I worked on complex functionality and testing within my work. BDD testing within PyDownloader on code written by myself and others in the team gave a great opportunity to work on these more complex programs while also reviewing and suggesting changes to pull requests from others. Gherkin syntax for fixtures and features was completely new to me as a way of testing and was interesting to learn how that operates in a collaborative environment too. Part of this was working with the task leads to identify the priority areas and order for getting to 100% test coverage in the system.

1.6.3 Project Management Level 4

Project management as a skill is almost entirely covered by the yearlong DataLogger project with the other 3 students. This, as described in more detail in the section about the project itself, involved extensive planning and communicating with stakeholders across the site as well as tracking and monitoring of our time. Management of this project was possible due to the timescale given which allowed for real management process to be completed. Scope documents,

requirements tables and others were created in detail and presented in formal meetings to managers.

Monitoring of times and costs were covered with a Gantt chart to track project progress (see 1.4.2.2) and redundancy + cost sheet to keep track of what hardware we had available on site along with associated costs to replace if no more were available.

2. Reflective Section

I have improved at so much during my time on placement with the support from everyone at ZF. Going into a ‘real’ workplace for the first time and getting to immediately work on real projects that are used by teams on site seems daunting but the willingness of people across the whole company to provide help and share their expertise with myself and all the placement students is so valuable.

In section 1.5, I mention gaining awareness of how organisations work and the non-technical ‘soft skills’ as a large reason for doing a work placement during university. Gaining confidence throughout and knowing how to go about communicating with individuals and teams is a great example of this. Speaking to shareholders, such as the servo drives team, for the DataLogger project really helped with understanding how to organise formal meeting but also what questions were better served by waking into the technical building to simply go find people. The relationships built since the start clearly help with this but almost equally being aware of the appropriate organisational process to go through.

I came into this placement with a somewhat sceptical view on a lot of standard corporate process, but my ability to work with these and gain a real understanding has strengthened throughout. Formal events, such as the trade mission day, showed this much further than standard office culture. Seeing the work that goes into these and assisting with the tours showed how these professional events come together for the benefit and corporate networking of all involved. This goes quite far outside what I would typically volunteer myself for in any other context, but this had built up my ability in these areas from what I had considered a weakness before.

Although much of my work at ZF was new to me with it being more electrical orientated, certain university modules really stood out with their application to these tasks. In first year working on my first real C code controlling a 7-segment display for morse code gave a strong baseline for understanding how the intermediary board state machine functioned for when I controlled this via the interface P side. Mentioned in 1.3.1 about PyDownloader, learning unit testing in java throughout software engineering modules was transferable to this BDD approach and lead to me helping explaining test standards and mock objects to others.

Going back into university, I feel I will do better in my final year than had I not completed a work placement. Working in large codebases actively alongside others is a much different style of software development, leading to my code becoming more readable and better documented. Irrespective of simply an extra year of learning and technical knowledge, working in collaborative environments on my day job tasks and especially the DataLogger I have gained valuable experience. Going through all the stages of a project, from initial scoping and requirements through to documenting our implementation is a valuable experience you never really get the opportunity for during education.

Having spent this time at ZF will be really beneficial going past university into my future career. Although individual companies won’t be structurally identical to here, the leaning from working in such a global organisation will be applicable almost anywhere. Insight into the automotive sector

has been fascinating and much different to what I usually learn about, and no matter what industry I go into in the future I'm sure I will have experiences where I think back to my time this year. Alongside this, connections made at the UKESF professional development course and future involvement with them have helped guide the direction my work may go. Though still not knowing exactly what I want to work on in my career, I am more focused on and understand what I see as important in the world for when I graduate.

3. Acronyms

A2L: ASAM MCD-2 MC/ASAP2 measurement and calibration description file
ASCLIN: Asynchronous/Synchronous Interface
BDD: Behaviour-Driven Development
CAN: Controller Area Network
DAP: Device Access Port
ECU: Electronic Control Unit
EEE: Electrical and Electronic Engineering
EMC: Electromagnetic Compatibility
EPP: Electric Power Pack
EXAM: EXtended Automation Method
JTAG: Joint Test Action Group
PyDl: PyDownloader
RRR: Residual Risk Report
Rx: Receive
TARA: Threat Analysis and Risk Assessment
TDD: Test-Driven Development
Tx: Transmit
UART: Universal Asynchronous Receiver/Transmitter
UDS: Unified Diagnostic Services
UKESF: UK Electronics Skills Foundation

4. References

[1] https://www.zf.com/mobile/en/homepage/homepage.html
[2] Azure DevOps, a project management and code hosting site. Not to be confused with GitHub, a project management and code hosting site which is also owned by Microsoft.
[3] https://astral.sh/blog/uv
[4] https://moonrepo.dev/moon
[5] https://www.vector.com/int/en/products/products-a-z/hardware/network-interfaces/vn16xx/
[6] https://www.peak-system.com/products/hardware/external-pc-interfaces/pcan-usb/
[7] https://www.iso.org/standard/72439.html
[8] https://uk.farnell.com/infineon/kitminiwiggler3usbtobo1/debugger-mcu-aurix-tricore-xmc/dp/2443543
[9] https://www.iso.org/standard/70918.html
[10] https://www.blender.org/
[11] https://mermaid.js.org/intro/
[12] https://www.trackmania.com/
[13] https://www.ukesf.org/