Motorsport Data Acquisition System and Live Telemetry using FPGA based CAN controller

[1]M Kanthi, [1]A Banerjee, [1]A V Jindal, [2]A Shankar, [3]V Sachdeva

[1]Electronics and Communication Engineering, [2] Electrical and Electronics Engineering, [3]Electronics and Insrtrumentation Engineering.
Manipal Institute of technology, Manipal Academy of Higher Education, Manipal, Karnataka, India, 576104

Abstract: The paper describes the design and working of a motorsport data acquisition, logging, live telemetry, and display system developed using the Controller Area Network (CAN) communication protocol as the backbone of the arrangement. The main controller of the CAN system is the myRIO which was programmed using LabVIEW. A Formula One car hosts over a hundred sensors during each of its races. The data acquisition/logging system, although does not directly affect the car's performance, is indispensable when it comes to the testing and design phase of the car. Designers can validate their assumptions and calculations, real-time data during testing can be a safety indicator and it provides insight to the driver about the performance of the vehicle. The FPGA-based controller for CAN is designed for data acquisition and live telemetry system with the interest of the formula car team in mind. The design choices were made to improve and deliver a more effective system than the pre-existing ones. All choices of controllers, sensors, formatting were custom made for the requirements of the team. All programmable devices were coded individually to suit the system and the graphical user interface was designed internally. Data acquired by the proposed system helps in making sure that the car achieves the goals that were envisioned when it was designed.

Keywords: Controller Area Network, myRIO, STM-32, XBee, data acquisition, telemetry, FPGA, lap time, display and GUI, testing and validation, performance enhancement.

1. Introduction

The system for Formula Student car was designed keeping in mind the requirements to improve the performance of the car on track as well as off-track for design improvements. The most challenging part was to integrate the sub-systems into one under the governing communication protocol and to provide an easy and understandable user interface for observing the vehicle's performance. The proposed work is to design and implement a data acquisition/logging and telemetry system to be on-board a race car. Figure.1 shows the system outline.

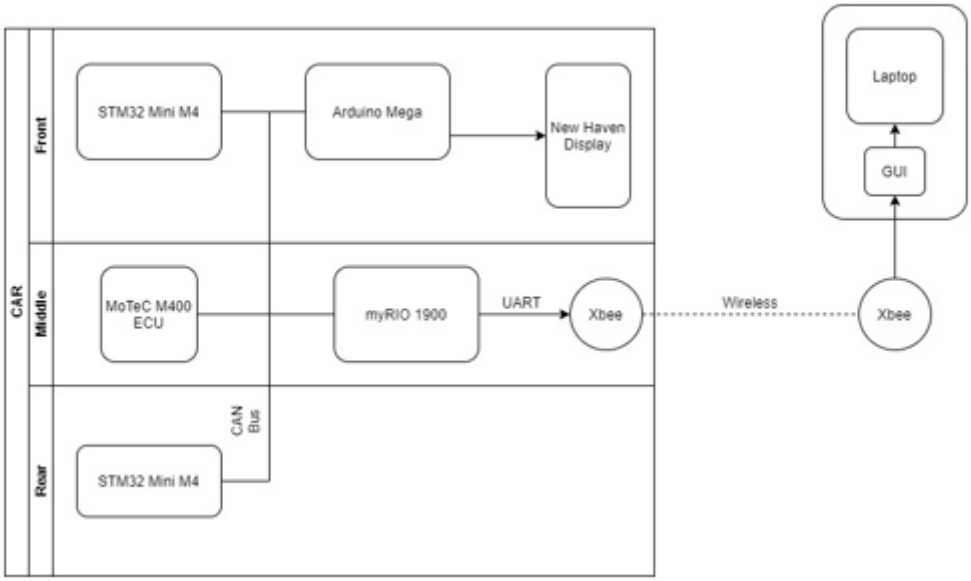

Figure 1. System Structure.

The major objectives of the proposed system are to:

- Establish a modular Controller Area Network (CAN) bus, which runs throughout the vehicle.
- Interfacing multiple nodes to the said CAN bus, each with its functionalities.
- Transmitting the data of multiple sensors onboard the vehicle.
- Display valid data to the driver while operating the car.
- Wireless data transmission back to host across a long range.

The data acquisition system is one of the most important systems for a Formula Society of Automotive Engineers (FSAE) for monitoring and analyzing various systems and parameters of the car [1]. The DAQ system samples real-world signals using transducers and further conditions the signal finally converting the conditioned signal to digital format for data handling by a computer and logging the data streams for analysis and storage. A DAQ system for motorsport applications should be rugged and capable of real-time performance [2]. The main controller of the CAN system is the myRIO which was programmed using LabVIEW, a user-friendly software. It is used to acquire data from the STM32 MINI M4 via the CAN bus and log the data in excel sheets. The data is further processed in the form of graphs using MATLAB. LabVIEW FPGA is a highly integrated development environment and FPGA is used to store CAN buffers and decrease the overall time of receiving and processing data [3,4]. DMA FIFO is used to transfer data between FPGA and the host processor, thus providing efficient separation of processing tasks so that the host processor remains free to perform operations during data transfer [5]. Live telemetry is used to enhance the trackside experience of the team during testing of the car [6]. It enabled us to view relevant data while the car was on track and judge how the car performs in real-time and adjusts the car's setup accordingly for the next run. It was also used to pass on information to the driver while the pit stops the car if a system did not behave as expected [7,8].

A pair of XBee Pro 900Hp modules (ZigBee Protocol) was used to establish this communication between the car and the laptop. The ZigBee protocol was developed for standardized application software on top of the IEEE 802.15.4 wireless standard. Its benefits include low cost, low power, and simplicity [9]. The CAN is a robust communication protocol and is the automotive industry and racing standard [10]. CAN is an International Standardization Organization (ISO) defined serial communications bus originally developed for the automotive industry to replace the complex wiring harness with a two-wire bus. The specification calls for high immunity to electrical interference and the ability to self-diagnose and repair data errors [10, 11]. It allows us to establish a high-speed multi-master bus with no issues with the arbitration. CAN bit timing is a feature of CAN communication with which bit rate, bit sample point, and the number of samples per bit can be programmed. Hence, we can optimize the performance of the network for a given application. We can also adjust the different bit

timing parameters regarding the oscillator tolerance and propagation delay to achieve optimal performance [12, 13].

The STM-32 MINI M4 is a development board powered by the ARM Cortex M4. It has an inbuilt CAN controller with multiple analog input pins. It perfectly fits into a standard DIP40 socket and is equipped with a 16 MHz SMD crystal oscillator. The board is preprogrammed with a USB HID bootloader, so it doesn't require any external programmers. This board collected data from various sensors through the CAN nodes and proved to be an efficient CAN controller catering to our needs in this project [13]. The Arduino MEGA has a staggering 54 digital input/output pins, 15 pins for Pulse Width Modulation, and 16 analog input pins [13]. In the project, it was used to control the display system of the car due to the easy interfacing with the Newhaven display using the open-source Arduino IDE which comes with a bundle of libraries to help code the Arduino MEGA.

2. Related Work

The primary use of a Data Acquisition System is comparing different channels of sensor data with one another. Often calculations and analysis require values of different sensors at the same time instant. Hence, it is essential to synchronize all the nodes of the systems so that sensors are sampled at the same time instant as well as correlate these same time instant data from different nodes in the myRIO before logging. A lot of work has already been done with regards to the synchronization of multiple nodes in a CAN bus, however, most of them require additional hardware or software implementations which increase the complexity of the system.

Turksi [14] describes a software method in which all the nodes sample their time based on the reception of a synchronization message. One or more predefined 'reference nodes' transmit their sampled value following the synchronization message and all the other nodes modify their time bases according to the value of the reference node by transformation equations. Gergeleit and Streich [15] describe a high-resolution real-time clock synchronization algorithm. The algorithm defines a master node. All the remaining nodes (slaves) are synchronized to the clock of this node. The master transmits an indication message at regular intervals followed by a second message which contains the timestamp at which the master transmitted the indication message. Each slave takes a local timestamp on the reception of the indication message and compares it with the timestamp received from the master to accordingly adjust their clocks. Rodriguez-Navas et al [16] propose a hardware solution where additional hardware modules called 'clock units' are attached to each of the nodes. The clock units provide the microcontrollers with a clock that is transparently synchronized to the other nodes by sending and receiving reference messages. The clock units have specialized CAN controllers which allow the value of the time to be written to the data field of a reference message while they are being transmitted and indicate the sampling point of the SOF bit of a message. Lee and Allan [17] built on the method proposed by Gergeleit and Streich by introducing fault tolerance. They implement a multi-master system by defining a group of nodes as Master Candidates Group (MCG). These nodes undergo a master selection procedure to determine the master which will send the synchronization messages. If any of the nodes in the MCG undergoes a fault it is replaced by a node from the rest of the slaves.

All the above-mentioned methods focus on having microsecond accuracy as well as fault tolerance mechanisms. Both are not critical requirements for the proposed system. Our method is simpler to implement and there is no requirement for additional messages. All the nodes transmit a set of messages in a sequence rather than simply broadcasting the data. The first set of messages is transmitted by the engine control unit at the rate of 50hz and serves as synchronization messages for all the other nodes to adjust their clocks as well as begin the next cycle of acquisitions. Once the engine control unit has finished transmitting, the next node starts transmitting the data acquired in the past 20ms. Hence, a message format and order of transmission was decided which allows myRIO to differentiate between data from the same sensor but sampled at different time instants. While this method has been designed by choosing the sensors and controllers, which can be used in any distributed system. It introduced a node tasked only with sending a periodic synchronization message.

3. Theory and Methods

Motorsport is a continuous improvement process, logged data helps designers avoid the mistakes their predecessors committed or take useful information out of their successes. A design is only as good as the performance of the part in the field and tests, but to judge whether a part/system is behaving like it

is supposed to. Data from different sensors monitoring different physical parameters are required to validate the design. A Data Acquisition and Telemetry System helps analyse the vehicle's behaviour during different tests and setups. Simulations can predict the behaviour of a certain part but to ensure proper results and safety such a system is a must. A Telemetry System also helps to analyse the driver's performance. Looking at real-time data while the driver is on the track can help to give invaluable feedback to improve performance.

3.1. Why Controlled Area Network (CAN)?

CAN is a multi-master, message broadcast system that allows a maximum signaling rate of 1 megabit per second (bps). In a CAN network, many short messages like temperature or rotations per minute (RPM) are broadcasted to the entire network, which provides data consistency in every node of the system. The protocol is motorsport and commercial industry standard due to its modular and 2-wire physical bus which has led to the replacement of complex wiring harnesses. Additionally, CAN sends specialized standard CAN message formats that allow up to 8 bytes of data per message, each with their 11-bit identifier which decides the message arbitration and prevent message collision.

3.2. Design Considerations

The main factors that were taken into consideration before picking any component:

1. Ability to withstand the harsh motorsport environment.
2. Ease of programming and interface.
3. Logged data to be viewed and understood with ease.
4. CAN compatibility and other communication protocol compatibility.
5. Ability to self-diagnose and repair data and physical layer errors.

3.3. Data Acquiring

The process of Data Acquiring is divided between the Electronic Control Unit (ECU) and the two STM-32 mini M4 microcontrollers, each of them hosts ADC channels with several analog input pins and since most of the sensors used are analog sensors, any addition of sensors is as easy as configuring one more new micro-controllers onto the CAN bus with the new sensors attached to it. Different sensors demand different sampling rates. To address this each microcontroller is programmed to read and transmit and specific time intervals and to ensure data consistency a message order is followed. The IMU VectorNav VN200 is connected directly to myRIO via UART over USB. The VectorNav is connected to the USB port of myRIO, and specific commands are sent which configure the VectorNav and specify which channels of data are required, which format they are to be sent (ASCII or binary) and the rates at which they are to be transmitted. Table. 1 gives the VectorNav VN200 IMU Characteristics. For the current system, the data channels being acquired are three-axis accelerometer values, three-axis gyroscope values, and GPS location data at 100 Hz sampling frequency.

Table 1. VectorNav VN-200 IMU characteristics.

Parameter	Min	Type	Max	Units
Gyroscope Sensitivity				
Full-Scale Range	-2000		+2000	°/s
In run bias stability			10	°/hr
Noise Density		0.0035		$°/s\sqrt{Hz}$
Bandwidth		256		Hz
linearity			0.1	% FS
Alignment Error	-0.05		+0.05	°
Accelerometer Sensitivity				

Full-Scale Range	- 16		+16	G
Sensitivity Scale Factor	2.048		16.384	LSB/g
Initial Calibration Tolerance		±3		%
Sensitivity Change vs. Temperature		±0.02		%/°c
Nonlinearity		0.5		%
Cross-Axis Sensitivity		±2		%
GPS				
Solution update rate		5		Hz
Time-to-first fix (Warm/Cold start)		36		S
Time-to-first fix (Hot start)			1	S
Altitude			50,000	M
Velocity			500	m/s
Environment				
Operating Temperature	-40		+85	C
Storage Temperature	-40		+85	C
Electrical				
Input Voltage	3.3		17	V
Current Draw (At 5V)		80		mA
Power Consumption			500	mW

3.4. Wireless Transmission

To evaluate the real-time data while the car is running, a telemetry system was implemented with a pair of XBee Pro 900 HP modules to establish communication between the car and a laptop. The XBee module placed on the car receives data from the myRIO 1900 through UART. It then broadcasts this data to the other XBee connected to the laptop. The Software was designed using MATLAB App Designer to display various data received in an organized way.

3.5. Log File Retrieval

The aim is to retrieve sensor data logged in by the myRIO in between test runs of the car. The data was retrieved wirelessly using XBee pro 900Hp modules. MATLAB is used to collect and store data in excel sheets. The myRIO logs data of multiple DAQ sensors in TDMS files. This file is retrieved via a command sent from the laptop, which prompts the XBee on the car to transmit the TDMS file instead of data packets for telemetry. MATLAB code enabled to get the entire log file consisting of all the sensor data in a form of a well-arranged excel sheet, for review. Despite the size of the excel sheet being quite large, we had managed to shrink down the process of log-file retrieval to a minute.

3.6. Lap Time Calculation

In data acquisition, acceleration, gyroscope, and GPS data are being acquired from VectorNav. Its highly accurate data provided a variety of applications. One such application would be using this data to calculate lap-timing data, which is extremely important during the testing phase of our car because we need to measure the time set by the car to quantify its performance. The most important event that needs to be timed accurately is the acceleration event (covering 75 meters in the shortest time possible) because, in this event, even milliseconds prove to be the deciding factor for the car's position in the event. The flowchart for lap-time calculation is shown in Figure.2. This lap-time data was then included in our telemetry system, so that we could monitor the laps timed by the car in every run on our stationed laptop, providing pivotal data for drivers to improve their timings.

Figure 2. Lap time calculation

3.7. Display System

A Newhaven NHD-240128WG-BTFH-VZ 240 x 128 LCD is placed on the dash of the car. It is controlled by the Arduino MEGA board which receives data through the CAN bus and transfers it to the display. The display shows information like gear position, rpm which are critical for the driver for optimum performance on the track. Information like oil pressure, fuel level tells the drivers the current state of the car so that they can adjust their driving accordingly. The data transfer between the board and the display takes place through an 8-bit parallel data transfer. Figure.3 shows the working display.

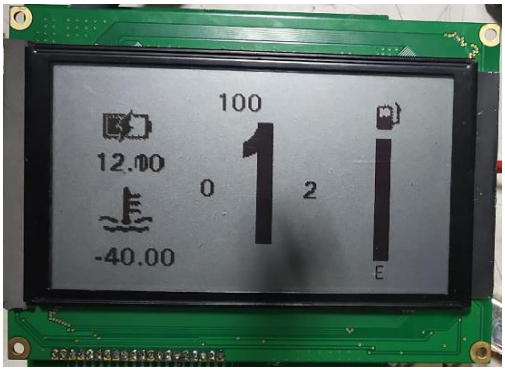

Figure 3. Working display showing relevant critical sensor data.

3.8. Graphical User Interface

A graphical user interface was designed using MATLAB App Designer shown in Figure.4, to represent the data received from the XBee module as clearly as possible. Data such as engine coolant temperature and oil pressure so that the team can ensure that all systems are running nominally and there is no issue with the car. Throttle position and RPM are displayed along with fuel level and battery voltage. The throttle position along with the RPM is used to check the responsiveness of the engine to driver input.

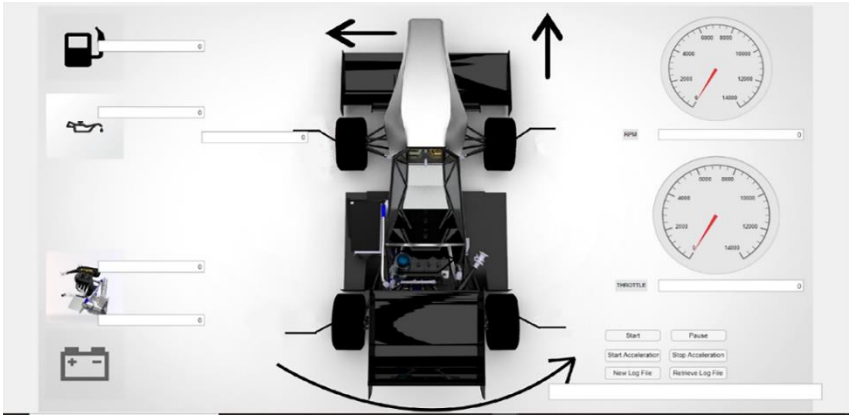

Figure 4. Graphical User Interface.

3.9. Message Order Formatting

As CAN only allows short messages to be broadcast at a time, to ensure data consistency a specific message order was designed, and each message was formatted to deliver the exact data required, this was done with consideration of the specific sampling rate of each sensor and with the ease of logging in mind. Figure 5 describes the message order. Each message cycle is initiated by the ECU and follows the specified order as illustrated below, each message set is comprised of a group of messages until all messages have been sent in a set the next set is not sent, this is done to ensure data consistency while logging and further transmitting.

Each message in a set is formatted individually with data from specific sensors, this is done independently by each node in the message cycle. These nodes are first to read from the sensors via the ADC channels then each sensor data is divided into high and low bytes, then these bytes are formatted into their respective CAN messages.

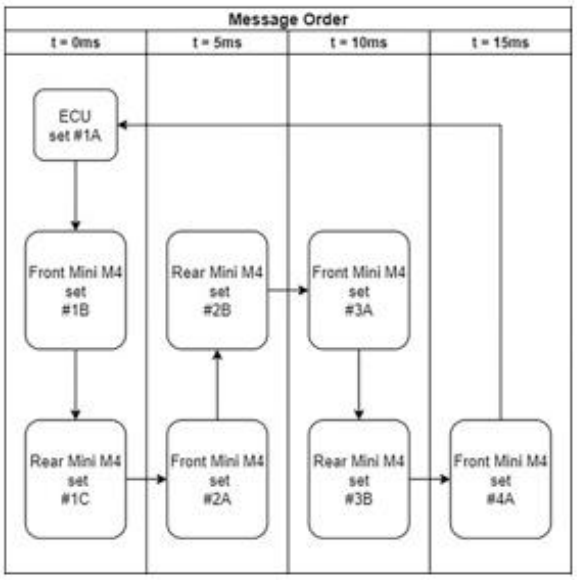

Figure 5. Message sending order.

The primary use of a data acquisition system is comparing different channels of sensor data with one another. Often calculations and analysis require values of different sensors at the same time instant. Hence, it is essential to synchronize all the nodes of the systems so that sensors are sampled at the same time instant as well as correlate these same time instant data from different nodes in the myRIO before logging. A lot of work has already been done with regards to the synchronization of multiple nodes in a CAN bus, however, most of them require additional hardware or software implementations which increase the complexity of the system. Hence, a message format and order of transmission was decided which allows myRIO to differentiate between data from the same sensor but sampled at different time instants.

MoTeC M400 can be setup up to transmit a set of sensors at a specific frequency. This frequency has been set to 50hz considering the sampling rates of all the sensors connected to ECU that are also needed in the data acquisition system.

This set of CAN messages is used as a synchronizing clock with the STM mini M4 microcontrollers, resetting their acquisition clocks on receiving the first CAN message from the ECU. Each of the mini M4's has sensors connected with acquisition rates of either 200 Hz, 100 Hz, or 50 Hz. Hence, each 20ms period between two consecutive sets of ECU messages could be divided into 4 parts each of 5ms. All the sensors are sampled and transmitted in the first period; only the 200 Hz sensors are sampled and transmitted in the second period, the 100 Hz and 200 Hz are sampled and transmitted in the third period and again only, the 200 Hz sensors are sampled and transmitted in the fourth period. In each period, the Mini M4 in the front of the car transmits a set of messages containing data acquired in that period, followed by the Mini M4 in the rear of the car. This ensures transmission from different nodes occurs in the same order in which they are acquired and can be easily correlated in the myRIO. Each set of transmissions from a node consists of multiple CAN messages, the number of sensors sampled in that period. Each CAN message can contain up to 4 sensor values. Two bytes are used for each sensor data containing the 12-bit ADC value.

S.NO	COMPONENT	MESSAGE SETS	MESSAGE IDS	DLC	CAN MESSAGE FORMAT							
					BYTE 1	BYTE 2	BYTE 3	BYTE 4	BYTE 5	BYTE 6	BYTE 7	BYTE 8
1	MoTeC M400 ECU	#1A	1520	8	RPM	RPM	OPS	OPS	ECT 1	ECT1	ECT2	ECT2
			1521	8	WS-FR	WS-FR	WS-FL	WS-FL	WS-RR	WS-RR	WS-RL	WS-RL
			1522	6	TPS	TPS	BT VOL	BT VOL	FL	FL	EMPTY	EMPTY
2	FRONT MINI M4	#1B	1523	8	BRT-FR	BRT-FR	BPPS	BPPS	PITOT	PITOT	SUS-FR	SUS-FR
			1524	8	SUS-FL	SUS-FL	STEER	STEER	TT-FR-0	TT-FR-0	TT-FR-1	TT-FR-1
			1525	8	TT-FR-2	TT-FR-2	TT-FL-0	TT-FL-0	TT-FL-1	TT-FL-1	TT-FL-2	TT-FL-2
		#2A	1529	8	TT-FR-0	TT-FR-0	TT-FR-1	TT-FR-1	TT-FR-2	TT-FR-2	TT-FL-0	TT-FL-0
			1530	4	TT-FL-1	TT-FL-1	TT-FL-2	TT-FL-2	EMPTY	EMPTY	EMPTY	EMPTY
		#3A	1533	6	SUS-FR	SUS-FR	SUS-FL	SUS-FL	STEER	STEER	EMPTY	EMPTY
			1529	8	TT-FR-0	TT-FR-0	TT-FR-1	TT-FR-1	TT-FR-2	TT-FR-2	TT-FL-0	TT-FL-0
			1530	4	TT-FL-1	TT-FL-1	TT-FL-2	TT-FL-2	EMPTY	EMPTY	EMPTY	EMPTY
		#4A	1529	8	TT-FR-0	TT-FR-0	TT-FR-1	TT-FR-1	TT-FR-2	TT-FR-2	TT-FL-0	TT-FL-0
			1530	4	TT-FL-1	TT-FL-1	TT-FL-2	TT-FL-2	EMPTY	EMPTY	EMPTY	EMPTY
3	REAR MINI M4	#1C	1526	8	BRT-RL	BRT-RL	SUS-RR	SUS-RR	SUS-RL	SUS-RL	TT-RR-0	TT-RR-0
			1527	8	TT-RR-1	TT-RR-1	TT-RR-2	TT-RR-2	TT-RL-0	TT-RL-0	TT-RL-1	TT-RL-1
			1528	2	TT-RL-2	TT-RL-2	EMPTY	EMPTY	EMPTY	EMPTY	EMPTY	EMPTY
		#2B	1534	4	SUS-RR	SUS-RR	SUS-RL	SUS-RL	EMPTY	EMPTY	EMPTY	EMPTY
			1531	8	TT-RR-0	TT-RR-0	TT-RR-1	TT-RR-1	TT-RR-2	TT-RR-2	TT-RL-0	TT-RL-0
			1532	4	TT-RL-1	TT-RL-1	TT-RL-2	TT-RL-2	EMPTY	EMPTY	EMPTY	EMPTY
		#3A	1531	8	TT-RR-0	TT-RR-0	TT-RR-1	TT-RR-1	TT-RR-2	TT-RR-2	TT-RL-0	TT-RL-0
			1532	4	TT-RL-1	TT-RL-1	TT-RL-2	TT-RL-2	EMPTY	EMPTY	EMPTY	EMPTY

LEGEND	
SYMBOL	DEFINATION
	HIGH BYTE
	LOW BYTE
	EMPTY
XX-XX-XX	POSITION
RPM	RPM CALCULATED BY ECU
OPS	OIL PRESSURE SENSOR
ECT	ENGINE COOLANT TEMPRETURE
WS	WHEEL SPEED SENSOR
TPS	THROTTLE POSITION SENSOR
BT VOL	BATTERY VOLTAGE
FL	FUEL LEVEL SENSOR
BRT	BRAKE ROTOR TEMPERATURE
PITOT	PITOT TUBE
SUS	SUSPENSION POTENTIOMETER
STEER	STEERING ANGLE SENSOR
TT	TIRE TEMPRETURE
FR	FRONT RIGHT
FL	FRONT LEFT
RR	REAR RIGHT
RL	REAR LEFT
0	AREA STRIP OF TIRE 1
1	AREA STRIP OF TIRE 2
2	AREA STRIP OF TIRE 3

Figure 6. Message format.

3.10. Data Acquisition and Logging

The myRIO-1900 microcontroller was chosen because of the versatility it offers due to its SoC containing both an ARM processor and an FPGA. This allowed us to off-load the non-deterministic CAN communication code onto the FPGA fabric allowing all the other tasks to run on the ARM processor without being blocked from polling the external CAN controller. This was achieved by developing a custom library in LabVIEW to interface with the external CAN controller (MCP2515) which runs on the FPGA. The messages received from the external CAN controller are stored in a buffer that could be accessed by the ARM processor whenever it's free.

The procedure of data logging is divided into four parts with the first part running on the FPGA controller and the subsequent parts running on the real-time processor. The first part is retrieving data from the sensors via the CAN bus. As the myRIO does not have an internal CAN controller, a library is created in LabVIEW to interface with an external CAN controller (MCP2515). The code is run on the FPGA and stores the received messages in a buffer. Storing the CAN message in a buffer enables the real-time code to quickly retrieve all the messages of the past 20ms and use the remaining time to execute other tasks. The second part involves taking data from the VectorNav, formatting it in engineering values, and storing it in a buffer. The third part includes retrieving data from both the CAN buffer and VectorNav buffer every 20ms and then writing it to the log file. As the VectorNav data is sampled at 100 Hz (10ms), there are two sets of VectorNav data. Before logging, the safety-critical sensor data is put in a buffer for telemetry. The final part is sending the safety-critical sensor data for monitoring the car to the XBee module via UART. This XBee transmits the data on XBee outside the car connected to the stationed computer. The data retrieved from all these sensors are logged and stored in the myRIO itself in the form of excel sheets by coalescing the data in the form of an array. Block diagram of LabVIEW code for myRIO CAN messages being received, telemetry and VectorNav data also being received, and all of this is stored in a TDMS excel file. The block diagram of LabVIEW code for the myRIO is shown in Figure.7. The stationed computer controls the logging status by communicating with the myRIO through the XBee and sending commands through the MATLAB GUI. The commands are sent to either "start log file data", "stop log file data", "create new logfile", "pause log file", or "restart log file".

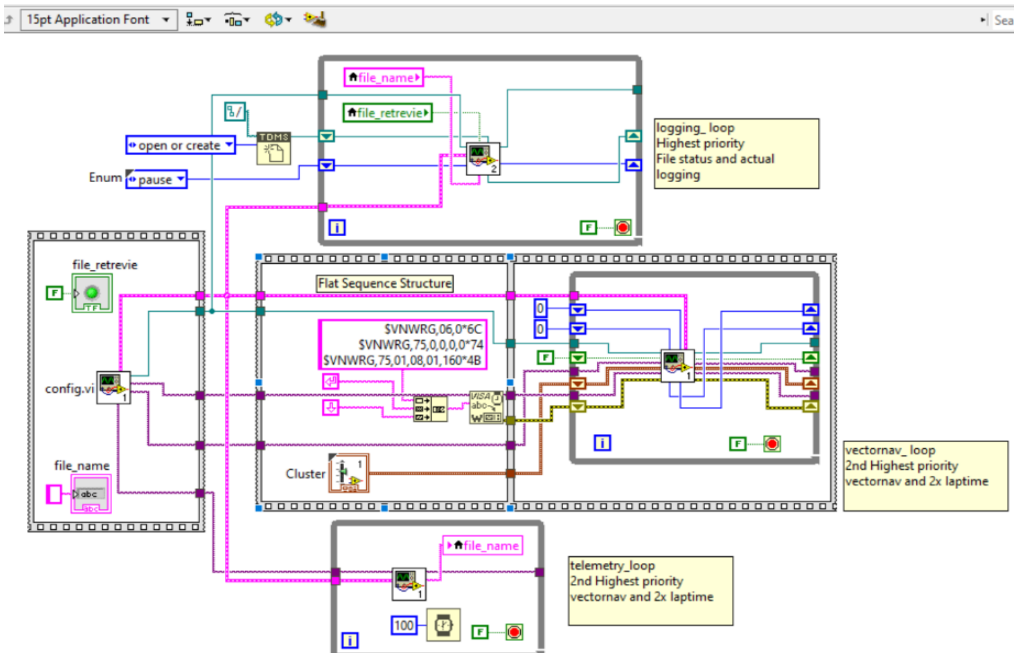

Figure 7. Block diagram of LabVIEW code for myRIO.

4. Results and Discussion

Data acquired by the proposed system helps in making sure that the car achieves the goals that were envisioned when it was designed. The data helps in ensuring that maximum performance is obtained without compromising the safety and reliability of the car. Proper analysis of data helps us in achieving three goals. They are:

4.1. Maximizing the performance

The readings from sensors help us understand how the car is behaving. Accordingly, changes such as length of suspension links, tire pressures, ARB angle, and spring stiffness could be made which affect various mechanical aspects of the car such as force transmission, load on individual tires, the temperature of tires, etc, which results in a faster car. This process involves iterating through various values of different parameters and comparing the lap times and sensor data to determine whether the

change has resulted in an improvement. To maximize the performance of the car, it is essential to tune it according to how the car is behaving rather than how it should behave according to the design. An example of this process would be iterating through different tire pressures and comparing the corresponding difference in peak values of the longitudinal acceleration graph obtained from the system. Figure.8 shows the longitudinal acceleration graphs of a straight-line acceleration event. The tire pressures are modified between runs and the peak acceleration is measured. This process is continued until maximum acceleration is obtained. The two graphs in Figure.8 show the change in acceleration by changing the tire pressure.

Another example would be changing the ARB angle and ride height. This changes the load transfers between the inner and outer tires during a turn. Reduced load transfer between the tires enables the drives to take faster turns. Different setups are tried until a maximum lateral acceleration is obtained.

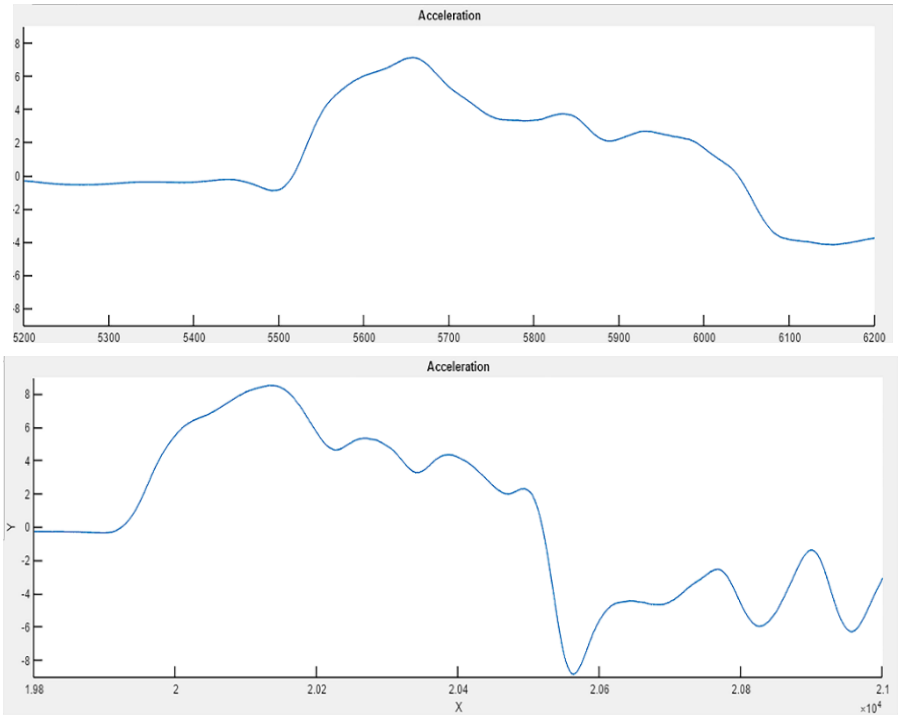

Figure 8. Longitudinal acceleration graphs of a straight-line acceleration event.

4.2. Driver Training

Tuning the car to obtain maximum performance is not very useful if the driver doesn't take advantage of it. Key Performance Indicators (KPI) are used to compare two drivers, KPIs are calculated using the sensors measuring the drivers' input i.e., the throttle, brake, and steering. Comparing the KPIs of the two drivers, the better driver could be chosen.

Further data could be analysed to understand and rectify the mistakes that drivers make. The graphs shown in Figure.9 are the acceleration, gyroscope, and throttle graphs of a circular lap with slaloms. By observing the lateral acceleration and yaw values, the turns and slaloms could be identified. It could then be checked if the driver starts applying the throttle at the correct point in a turn.

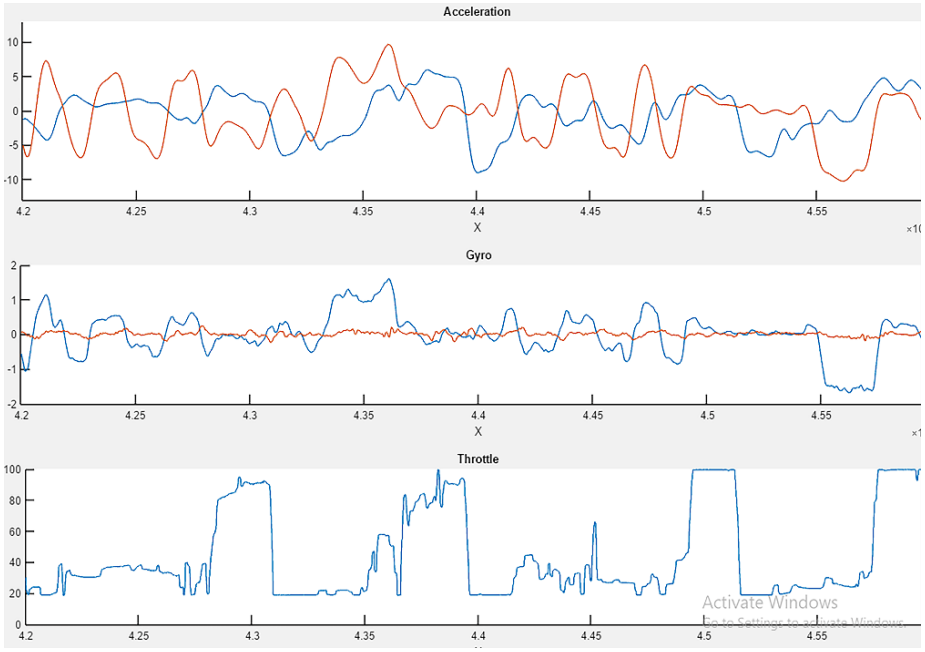

Figure 9. Acceleration, gyroscope, and throttle graphs of a circular lap with slaloms.

4.3. Safety and Reliability

The data also enables us to ensure that the parts of the car are working as expected and there are no issues that cause damage to the car. Parameters such as temperatures, pressures, and forces critical for the safety of the car are dependent on other parameters such as overall acceleration of the car, speed of the car, RPM of the engine, and air-fuel ratio used for combustion in the engine. Hence any sensor values not following the expected relationship with other sensor values indicates some failure or issue in the system.

The oil pressure vs engine RPM graph depicted in Figure.10 indicates how well the lubrication system of the engine is working. The oil pump is operated by the motion of the crankshaft so at higher rpm the oil pressure values should be higher. A graph with the more scattered values means lower oil pressures were seen at higher RPMs which indicates that the system is not working properly while the less scattered graph indicates that the system is working as expected. Identifying problems in the lubrication system is very helpful as a failure of the lubrication system could cause the entire engine to fail.

5. Conclusions

A Data Acquisition and Logging System for a formula race car is designed and implemented. The system is designed to identify performance improvement. Although this is a fully functional live telemetry system that satisfactorily acquires, transmits, and displays data, there is still scope for improvement and development. A professional 'off-the-shelf' system has a lot more features including a GPS with track map plotting, lap timing system, plotting of data, etc. The modular CAN bus allows us to expand into these territories of advanced racing technologies by providing us with a strong backbone for all our future additions to interface with the system.

.

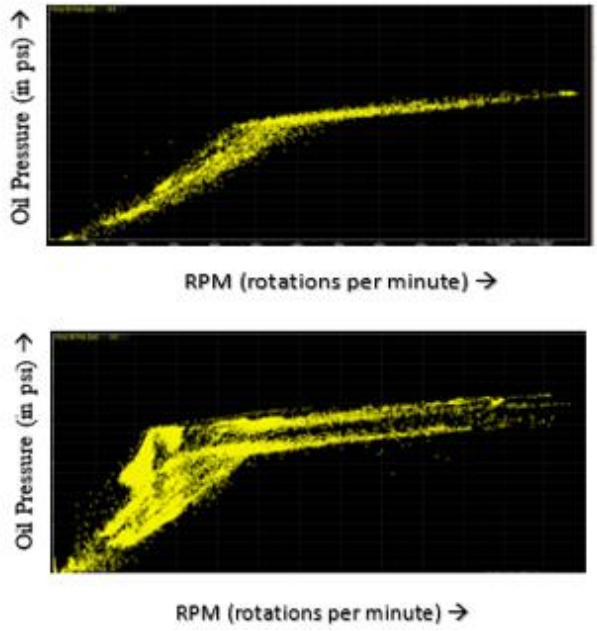

Figure 10. RPM vs oil-pressure graph.

References

[1] Khan S and Sonti S 2009 Data acquisition system for a 600cc Formula SAE race car *IEEE Int. Conf. Veh. Electron. Saf. (ICVES), pp. 46*-49, doi: 10.1109/ICVES.2009.5400316

[2] Nye A, LaChausse J, Doll R, Bush Z, Howard Jr J, Xu S and HsHui H C 2005 Senior Design Team 0510 - *RIT Formula SAE Data Acquisition System Preliminary Design Report*

[3] Murali H S and Meenakshi M 2013 Design and Development of FPGA Based Data Acquisition System for Process Automation *IJERT*, Vol.2, Issue.9, pp.2069-2072

[4] Shuang B, Hairong Y, Qingping C, Zhibo P and Yuying S 2017 A FPGA-Based Reconfigurable Data Acquisition System for Industrial Sensors *IEEE Trans. Ind. Inf.* pp. 1503-1512., 2017, 10.1109/TII.2016.2641462

[5] National Instruments Corporation May 2007 edition *CompactRIO and LabVIEW Development Fundamentals Course Manual*

[6] Nils Braune 2004 Telemetry Unit for a Formula Student Race Car *ETH Zurich*.

[7] Tan F Y 2011 *Development of Electric Formula SAE Real-time Telemetry Software*

[8] Ayyad A and Fathizadeh M 2011 Telemetry and Data Collection to Improve Formula SAE Car *WCECS*, Vol.1, 317

[9] Bagheri A, Ibarra C, Galatoulas L, Nikolaos F and Konstantinos G 2017 Field performance analysis of IEEE 802.15.4 XBee for open field and urban environment applications in Smart Districts *Energy Procedia*, 2017, 122. 673-678.10.1016/j.egypro.2017.07.368

[10] Bosch R 1991 *CAN Specification 2.0, Parts A and B*, BOSCH, retrieved 1991 from https://www.bosch.com

[11] Corrigan S 2000 Introduction to the Controller Area Network, *Texas Instrum*, SLO A101B

[12] Robb S and Kilbride E 2004 *NXP Freescale Semiconductors* - AN1798 - CAN Bit Timing Requirements

[13] Naik N, Mounik G S, Shetty D K, Gopakumar R, Bhat R, Gama V and Sampaio C 2020 Data Acquisition using CAN Communication Protocol for Static and Dynamic Conditions for Automotive Application *Int. J. Adv. Sci. Technol.* Vol.29, No.3, pp.6490-6503

[14] Turski K 1994 A Global Time System for CAN Networks *1st Int. CAN Conf.*

[15] Gergeleit M and Streich H 1994 Implementing a Distributed High-Resolution Real-Time Clock using the CAN-Bus *1st Int. CAN Conf.*

[16] Navas J G R and Bosch J 2006 Hardware design of a high-precision and fault-tolerant clock subsystem for CAN networks *IFAC Proc.* Vol. 36, no. 13, pp. 39-46,2003.