**Development of *LabVIEW* FPGA program for Energy Management System (EMS) Controller for Hybrid Electric Vehicle (HEV)**

BY

ABDUL AZIZ BIN MUSTAFFA KAMAL BASHA

FINAL PROJECT REPORT

Dissertation submitted to the Department of Electrical & Electronic Engineering
in Partial Fulfilment of the Requirements
for the Degree
Bachelor of Engineering (Hons)
(Electrical & Electronic Engineering)

Universiti Teknologi PETRONAS
Bandar Seri Iskandar
31750 Tronoh
Perak Darul Ridzuan

## CERTIFICATION OF APPROVAL

**Development of *LabVIEW* FPGA program for Energy Management System (EMS) Controller for Hybrid Electric Vehicle (HEV)**

By

Abdul Aziz Bin Mustaffa Kamal Basha

A project dissertation submitted to the
Department of Electrical & Electronic Engineering
Universiti Teknologi PETRONAS
in partial fulfilment of the requirement for the
Bachelor of Engineering (Hons)
(Electrical & Electronic Engineering)

Approved:

_____

Mr. Saiful Azrin Bin Mohd Zulkifli
Project Supervisor

UNIVERSITI TEKNOLOGI PETRONAS
TRONOH, PERAK

January 2016

## CERTIFICATION OF ORIGINALITY

This is to certify that I am responsible for the work submitted in this project, that the original work is my own except as specified in the references and acknowledgements, and that the original work contained herein have not been undertaken or done by unspecified sources or persons.

_____

Abdul Aziz Bin Mustaffa Kamal Basha

## ABSTRACT

This dissertation explains the construction of *LabVIEW* Field Programmable Gate Array (FPGA) for Energy Management System (EMS) Controller for Hybrid Electric Vehicle (HEV). The HEV is engineered to reduce the world's dependency on fossil fuels. An HEV is designed to utilize two power sources which are from electric motor and an internal combustion engine (ICE). These sources need to be carefully controlled so that the energy of both sources can be synergized to achieve fuel and power efficiency in the vehicle. The control algorithm is implemented by an EMS Controller for which in this project, it will run on a *National Instruments* (NI) *CompactRIO*, cRIO-9076. This EMS controller algorithm will be built and designed in FPGA of NI *LabVIEW* to extract and control parameters from the electric motor controller, which is the Motor Control Unit (MCU) and the engine controller, which is the Engine Control Unit (ECU). The extracted and controlled parameters are engine RPM, vehicle speed and vehicle fuel consumption. These data will be output using the embedded server to the client, which is a windows-based tablet PC and the embedded server is cRIO-9076. The communication between server and client will be implemented using HTTP-based communication protocol making the data appear in HyperText Mark-up Language (HTML) which will be rendered into the Graphical User Interface (GUI) web page interface. This GUI will enable the driver to monitor and control the MCU and ECU of the Hybrid Electric Vehicle.

## ACKNOWLEDGEMENT

In completing my Final Year Project (FYP), I would like to express my deepest gratitude towards my supervisor - Mr. Saiful Azrin Mohd Zulkifli for providing continuous support and advice for this project, simulating suggestions, motivation and encouragement. I am gratefully indebted to him for assistance which is really valuable and indispensable in the course of finishing this research and writing dissertation.

I owe my wholehearted thanks and appreciation to technologists of the electrical and electronic and mechanical engineering departments for providing assistance and permission to use the required equipment that is vital to complete this project. Not to forget, the FYP committee members for coordinating a well-organized programme for the Final Year Project at UTP

Last but not least, I would like to show appreciation to my family and my colleagues at UTP for their continuous support toward my FYP project. Furthermore, I'd like to also thank all those who I have not mentioned and have directly or indirectly contributed to the completion of this report. None of this would have been possible without everyone's help.

**TABLE OF CONTENTS**

## LIST OF TABLES

## LIST OF FIGURES

## LIST OF ABBREVIATIONS

| | | |
|---|---|---|
| CAN | : | Controller Area Network |
| ECU | : | Engine Control Unit |
| EMS | : | Energy Management System |
| FPGA | : | Field Programmable Gate Array |
| GUI | : | Graphical User Interface |
| HEV | : | Hybrid Electric Vehicle |
| HTML | : | Hyper Text Mark-up Language |
| I/O | : | Input/Output |
| ICE | : | Internal Combustion Engine |
| LAN | : | Local Area Network |
| NI | : | National Instrument |
| PC | : | Personal Computer |
| REEV | : | Range Extended Electric Vehicle |
| SoC | : | State of Charge |
| USB | : | Universal Serial Bus |

# CHAPTER 1
# INTRODUCTION

The automotive industry nowadays mainly depends on fossil fuel as energy for the prime mover of the vehicle. This natural resource utilization has contributed to critical environmental pollution in recent years. Hence, the world nowadays tries to shift their reliance on fossil fuel to relatively cleaner energy. However, cleaner energy such as solar, wind, biomass and etc. require high cost of implementation, which makes the fossil fuel more suitable for use in vehicle.

Since the automotive industry is a sector that greatly depend on this fossil fuel, revolutionizing the way this natural resource is utilized will effectively aid in reducing the environmental pollution problem.

The automotive industry today tries to come up with the eco-friendly automobile such as electric and hybrid electric vehicle. The electric vehicle is designed to use an electric motor to propel the vehicle, but for hybrid vehicles, it utilizes two engines propulsion sources, which is the internal combustion engine and the electric motor. The electric vehicle is a zero pollution energy source, but with the downside of long charging time since it utilizes battery to store the energy. Meanwhile, for hybrid electric vehicles it still need to rely on fossil fuel as its energy source. Both these types of eco-friendly vehicle are relatively expensive. Despite that, it is a great move by automotive industry in helping to resolve environmental pollution issue.

Therefore, to support the awareness of reducing the reliance on fossil fuel, this FYP project will support the new approach in building the hybrid vehicle by converting a conventional vehicle into a hybrid electric vehicle. This conversion involves conversion components retrofitted in to the internal combustion engine (ICE)-based vehicle.

## 1.1 Background of project

The Hybrid Electric Vehicle (HEV) for split-parallel drivetrain enables propulsion power to be provided to both the front and rear axles of a vehicle. As for normal conventional vehicle, the propulsion power from Internal Combustion Engine (ICE) is applied only on the front axle. This type of vehicle can be converted into HEV of split-parallel configuration with minimal modification to its system. This will enable the vehicle to reduce its fuel consumption. It can be done by replacing the rear wheels with in-wheel motors (IWM). These IWM require a controller to control and monitor its operation called Motor Drive/Controller Unit (MCU). The same situation goes to ICE where its own controller is called Engine Control Unit (ECU). The power from both MCU and ECU will be synergized, controlled and monitored by an Energy Management System (EMS) Controller to achieve optimum efficiency in energy utilization of HEV split-parallel axle architecture. [3, 4] The figure below shows the configuration of this split-parallel architecture.

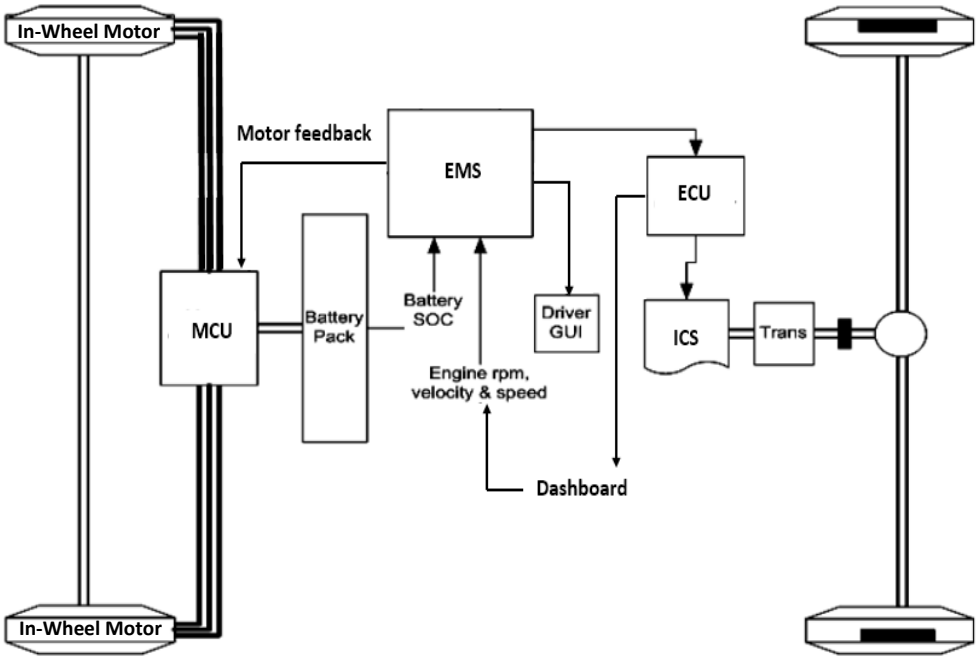



**Figure 1: Split-parallel HEV architecture [5]**

The present EMS has been programmed in 'Real-Time Scan Mode' to communicate between the EMS controller with ECU. However, this type of program mode cannot be used in getting data from the MCU because the Control Area Network

(CAN) module of the EMS controller (NI 9853) can only function in FPGA mode ONLY. or the EMS controller to communicate with MCU, the EMS controller needs this specific module (NI 9853). Therefore, this can be solved by converting the existing program in the EMS controller communicate with the MCU using FPGA program. Thus, the existing EMS control program needs to be converted from the Real-Time Scan Mode program to the FPGA program to enable communication between the EMS and MCU controllers. This program conversion is the first aim of this project. Both programming modes are based on the *National Instruments LabVIEW* software *LabVIEW*., while the controller hardware is CompactRIO cRIO-9076.

For this split-parallel architecture, the other desired design is to enable the vehicle driver to control and monitor the data attained from the EMS controller. EMS controller basically will read the signal status of vehicle's engine revolutions per minute (rpm), velocity and speed (km/h) and also its fuel consumption [4]. These data will be channelled to Graphical Driver Interface (GUI) in a tablet PC and connected to the *CompactRIO* cRIO-9076 via a web service protocol. This tablet PC utilizes a Windows operating system. For GUI interface, whenever web browser receive signal content from EMS controller, the Hypertext mark-up language (HTML) will render the content into the GUI web page interface [6] . This GUI development in windows tablet PC is the second aim of this project.

## 1.2 Problem statement

The HEV motor controller offers many parameters that can be monitored through CAN communication. Some parameters are also controllable in real time. However, CAN bus communication is incompatible with the existing *LabVIEW* Real-Time Scan Mode program of the EMS controller. This project implements *LabVIEW* FPGA programming on the *CompactRIO*, to enable integration of the EMS controller with the HEV motor controller.

## 1.3 Objectives

The objectives of this project are as follows:

1. To convert the existing *LabVIEW* program from Real-Time Scan Mode to FPGA programming in the EMS controller.
2. To develop FPGA program for CAN bus communication between the EMS controller and MCU for motor parameter reading and control.
3. To build a Graphical User/Driver Interface (GUI/GDI) on a Windows-based tablet PC for in-car real-time monitoring, control & data acquisition.

## 1.4 Scope of study

1. To understand differences between *LabVIEW* Real-Time Scan Mode and FPGA programming. mode *LabVIEW*
2. To perform program conversion in existing EMS controller (*CompactRIO*) to enable CAN bus communication with HEV motor controller
3. To develop a Graphical Driver interface (GDI) for real-time monitoring, control and data-logging of vehicle parameters

## 1.5 Relevancy of project

The outcomes of this project shall address the following;

1. *Reducing fossil fuel dependency, pollution and green house impact:* This split-parallel hybrid vehicle has TWO sources to propel the vehicle, which is the ICE and electric motor. Hence, dependency on ICE power utilizing fossil fuel is reduced in comparison to conventional vehicle. This will also reduce carbon monoxide emissions, which can lead to the greenhouse effect.
2. *Economical option:* User will have the option to have their conventional car converted to a hybrid vehicle rather than buying a new hybrid car. [2]
3. *Car resale value:* As of current trend on car resale value, the hybrid vehicle has higher resale value than conventional car. It is because of a hybrid car is equipped with higher standard equipment and most of it has strong reliability

records.[7] Hence, a conventional car converted into a hybrid vehicle certainly has impact on its resale value.

4. *Infrastructure Availability:* With hybrid EMS conversion kit retrofitted into a conventional car, the benefits of flexible vehicle fuelling for the user is realized. They can now charge their HEVs at home, workplace or public charging stations. They can also continue to fill up their car with gasoline. [8] This gives users the option on location that they can refuel their car.

## 1.6 Feasibility of project

The time frame to complete this project is within two semesters of Universiti Teknologi PERTONAS (UTP) academic calendar. The first semester is planned to understand *LabVIEW* software and program, get familiar with the all the tools and hardware, connections between hardware and also finalizing overall project planning. The first objective is expected to finish in the first semester. While in the second semester, the second and third objectives are expected to be finished, aside from prototype building and fine tuning.

# CHAPTER 2
# LITERATURE REVIEW

Common configurations of Hybrid Electric Vehicle (HEV) are of the series, parallel or power-split HEV configurations [9]. These configurations have the ICE and EM deliver propulsion power to the same drive axle – either the front or rear axle of the car. These types of hybrid vehicle are expensive and are designed specifically for a certain car (OEM vehicles). However, one hybrid vehicle architecture which enables a conventional ICE vehicle to be converted into a hybrid automobile is called a split-parallel hybrid where ICE will propel the front axle and EM will propel the rear axle. This type of drivetrain is currently being developed in Universiti Teknologi PETRONAS. This system requires an Energy Management System (EMS) to synergize the TWO power sources, which are ICE and EM and also an on-board battery pack [3, 4]. The present EMS controller is programmed in *National Instruments'* embedded *CompactRIO*- modelcRIO-9076, using *LabVIEW* software [3]. The split-parallel drivetrain also utilizes re-generative breaking system of IWM hardware to boost HEV power efficiency and energy savings [4].

The following are detailed explanation of the HEV architecture, EMS controller, *CompactRIO, LabVIEW*, IWM System and re-generative breaking.

## 2.1 Hybrid Electric Vehicle (HEV) architecture

**1) Parallel hybrid:** In this configuration, both ICE and EM will work in tandem to deliver the propulsion power to the front axle [10]. This single parallel hybrid transmission system, rely on ICE as main prime mover with EM functions as an additional power source. Hence it will perform very well during high speed than the series hybrid system [11]. This EM also will work as a generator to recharge the vehicle. Aside from that, parallel hybrid also normally utilize regenerative braking to generate the energy and enhance its hybrid system efficiency [10].

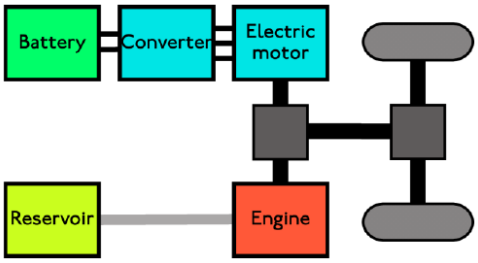



**Figure 2: Parallel hybrid architecture [9]**

**2) Series hybrid:** This system is also known as Extended-Range Electric Vehicles (EREV)[12] or Range-Extended Electric Vehicles (REEV) [9]. For this architecture, the EM will solely provide the propulsion power to the front axle while the ICE will act as generator to charge the battery bank. Hence due to this configuration, the vehicle will fully functional as an electric car on a short distance [11]. The ICE for series hybrid also is smaller than then the parallel hybrid configuration one, but it requires larger battery bank since this battery bank will be the one empowering the EM. This large battery bank causes the series hybrid configuration vehicle becomes more expensive than the parallel architecture one [10].

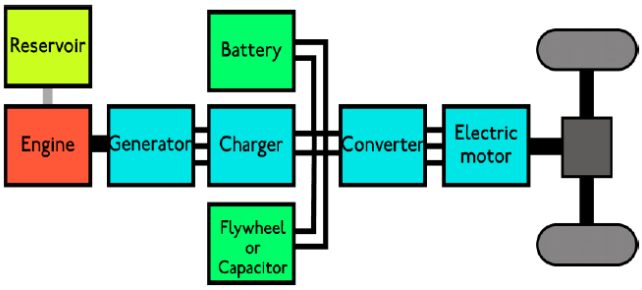



**Figure 3: Series hybrid architecture [9]**

**3) Power-split hybrid:** The other name of this configuration is series-parallel hybrid system. The propulsion energy that drives the front axle can be either from mechanical or electrical wise [9]. The EM will be the vehicle prime mover during the low speed (as in series configuration) and ICE for the high speed one (as in the parallel hybrid system). The reasons are that the series hybrid work efficiently during low speed and parallel hybrid work efficiently during high speed [10]. This power-split hybrid configuration requires larger ICE and but smaller and highly efficient EM [9]. It also requires large battery pack to store the power. This types of architecture certainty has better performance and fuel saving than parallel or series hybrid configuration [10].

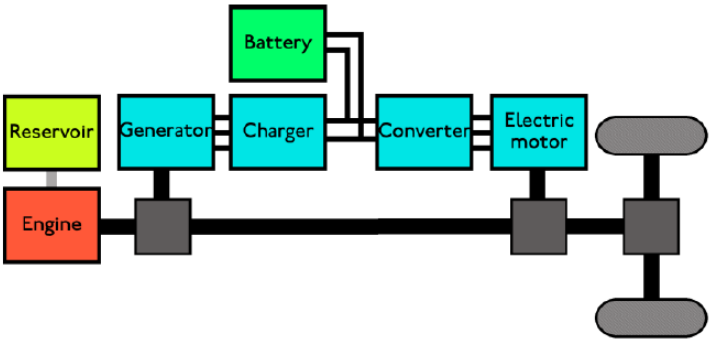



**Figure 4: Power-split hybrid architecture [9]**

**4) Split-parallel hybrid:** In this configuration, ICE and two EM will provide propulsion power to different vehicle axel. ICE will provide propulsion power to the front axle. While, the two EM will propel the rear axle which is a non-driven wheels. Therefore, this EM is called 'hub motor' or 'In Wheel Motor (IWM)'. Other than that, battery bank will also be connected to this real axel. Since, there is no hard connection between both front and rear axle, battery bank cannot directly be charged by ICE. Battery will be charged by IWM when vehicle is moving only. This means that the power is deliver from ICE to battery through the load which are vehicle's framework, wheels and road coupling rather than mechanical device. Hence, due to this design, the car is called split-parallel Through-The-Road (TTR) configuration. This TTR design actually allow the normal conventional vehicle to be converted into HEV with some modification. Therefore, after this conversion is done the configuration is now can be called as TTR-IWM hybrid architecture. [4, 5]

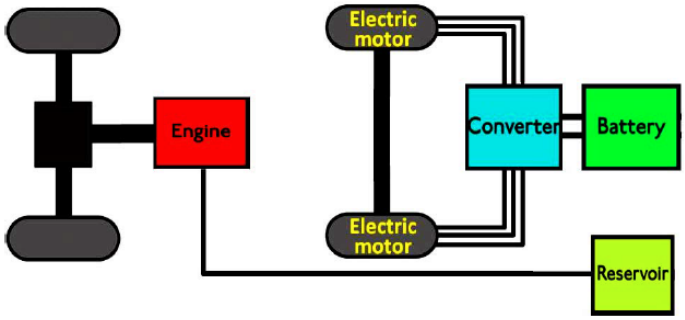



**Figure 5: Split-parallel hybrid architecture [9]**

## 2.2 Energy Management System (EMS)

The general purpose of EMS controller is to reduce vehicle fuel intake aside from self-sustaining battery. Specifically for the split-parallel hybrid architecture, the EMS is

connected to ECU and MCU which enable it to read the input of throttle and vehicle speed, engine rpm and also the battery SOC [4]. Then based on these inputs, the EMS controller will plan the power management strategy to dictate the best operation of this TTR-IWM drivetrain [5]. All the inputs from ECU and MCU later, will be controlled and monitored by EMS controller and will synchronized this input data to the Graphical User Interface (GUI) in Windows-based tablet PC. This GUI will be developed using *National Instruments'* software *LabVIEW*. The GUI implementation will allow the vehicle driver to monitor and control the specific car parameters according to their desire [6]. For this project, all the algorithm development of EMS controller program will be done using Field Programmable Gate Array (FPGA) where it will run in National Instrument *CompactRIO*, cRIO-9076. For GUI development, the web service protocol will be use in the development.

### 2.2.1 EMS Control Strategy

There are three controls strategy for HEV which are normally use in HEV [21]:

1. Parallel Electric Assist; ICE acts as key propulsion power and electric motor acts as secondary source. Based on certain control strategy the Battery State of Charge (SoC) will be conserved at certain level.
2. Fuzzy Logic; It utilize "load levelling" idea. The electric motor use as a backup power to ICE. Meanwhile, the ICE is run at full efficiency. This system determines the vehicle power requirement and compute circulation among ICE and electric motor.
3. Adaptive Control; A complex control approach that monitors the systems in real time. It takes into account both fuel consumption and emission level which resulted in better EMS strategy control decision.

Therefore, in this project it is more suitable to implement the parallel electric assist control strategy due to its simpler and flexible nature. Since the electric motor require seamless integration with ICE, it need to work in certain routine;

- Motor controller will only operate when driving torque lower than certain minimum speed
- To charge battery during regenerative braking

- Electric motor will act as the key propulsion for vehicle. This is when engine ICE is very low. In this situation the ICE will be turn off
- During low battery State of Charge (SOC) – Motor will be charged by excess torque from ICE

## 2.3 CompactRIO and LabVIEW

*CompactRIO* (Fig 6) is a programmable hardware which rely on *LabVIEW* software to create its program. It usually builds in Real-time Scan Mode or FPGA program. For, Human Machine Interface (HMI) coding, it will enable the program information to be display in GUI. This hardware will lessen the system development time and complexity of a project. It offers the capability to monitor and control of various applications. This *CompactRIO* also has communication and logging function for maximum flexibility and performance of the system [13, 14]. *CompactRIO* need to be configure as the following figure in order to execute real-time control and monitoring. [13]

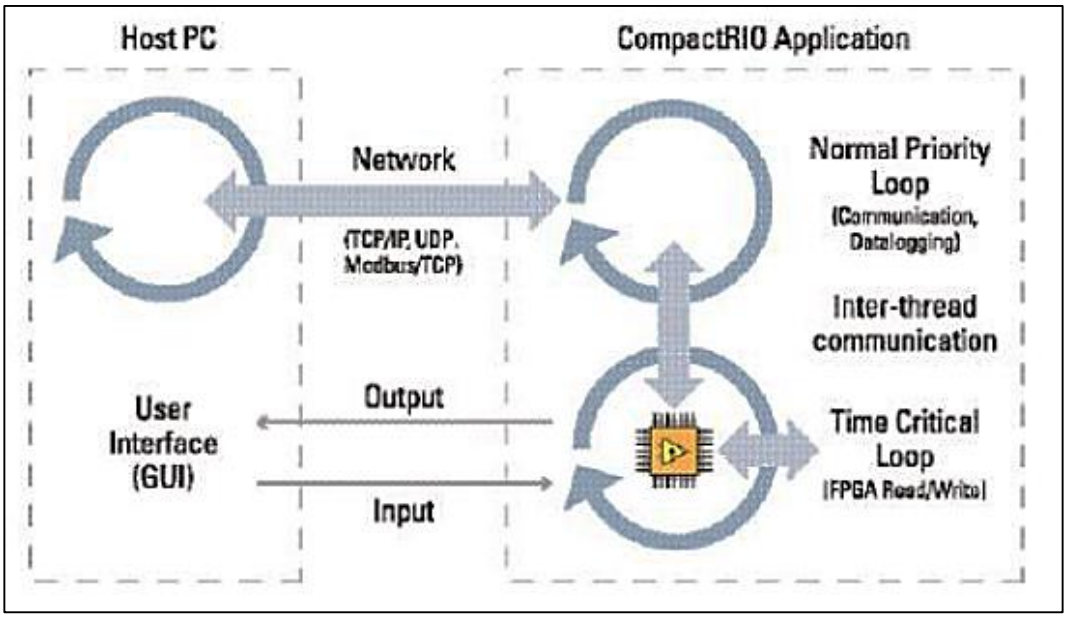



**Figure 6: CompactRIO software architecture [13]**

### 2.3.1 cRIO-9076

The cRIO-9076 is a hardware that will be use to execute for EMS control program for this project. This hardware contains 400 Mhz real-time processor, 4 slot chassis with and embedded and configurable LX45 FPGA chip. The other hardware features are as of figure below:

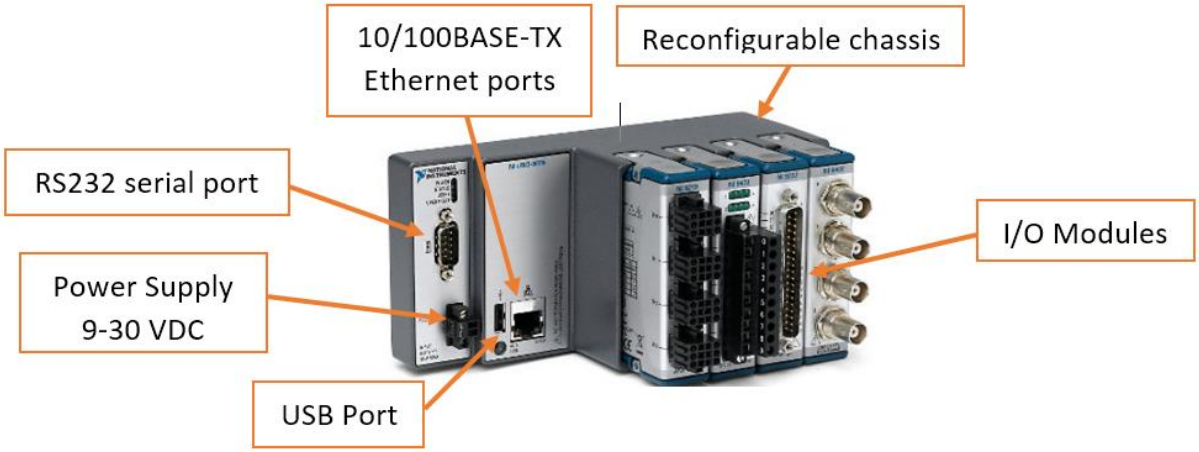



**Figure 7: NI *CompactRIO*, cRIO-9076[13]**

#### 2.3.1.1 MODULE

In this project, there are only TWO modules that will be used which are NI 9401 and NI 9853.

##### 2.3.1.1.1 NI 9401

This module is a bidirectional input and output module with 8 channel where each of it compatible with 5V/TTL. This hardware can response to digital signal in about 100ns. It usually being utilized in program that require input or output of high speed counter or timers, digital communication protocols, pulse generation and a lot more. [15] This module can run on both Real-time Scan Mode or in FPGA program.

#### 2.3.1.1.2 NI 9853

This module is a high-speed Control Area Network (CAN) module which has one port for internally powered and other port for externally powered. This module has ability to transfer or transmit signal at 100% bus load up to 1 Mbit/s. This module capable to synchronize with any NI *CompactRIO* input and output module at 25ns resolution. [16] This module can only run in FPGA program ONLY.

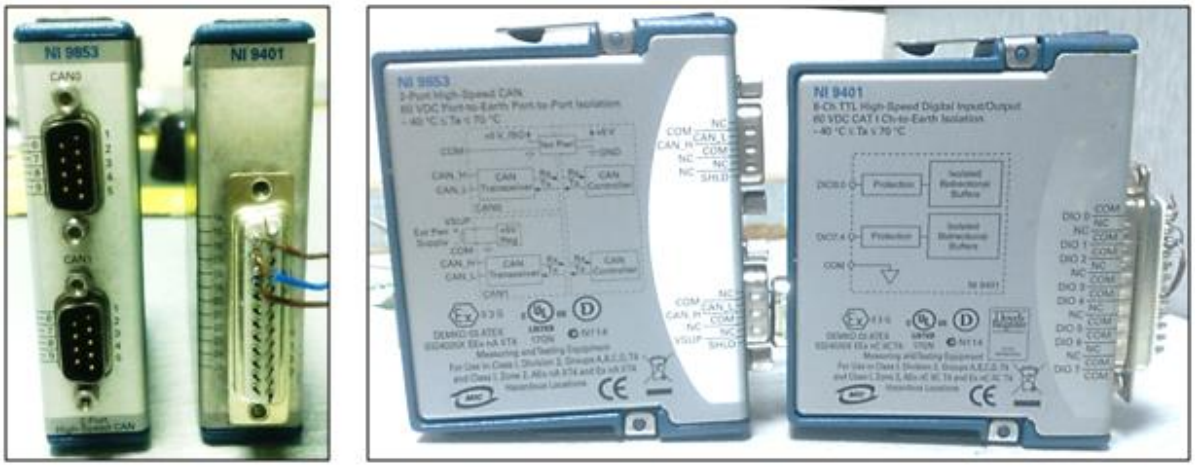



**Figure 8: Module NI 9401 and NI 9853**

### 2.3.1.2 Field Programmable Gate Arrays (FPGA)

FPGA are made of silicon chip with unconnected logic gates. This gates can be wired together using software where it can be compiled into bit stream configuration file. The FPGA will takes different "personality" once it reconfigures its gates. It is useful in an application where time and cost of developing and fabricating application-specific integrated circuit (ASIC) is prohibitive. This ASIC is different from FPGA since its functionality is fixed according to the design. While for FPGA, it is design and execute in hardware without operating system. This FPGA offers; [17]

1) Flexibility – FPGA can be configured according to current needs and reconfigure according to future demand. There is no need to go through the long fabrication process as ASIC. Hence, it will save time to build the desired program.
2) Performance – FPGA is control by programmable interconnects of gates, hence it can implement parallel task to run the program simultaneously and

independence of each other. The L*abVIEW* FPGA run on up to 120 Mhz clocks. Hence, it can run and accomplish more executions per clocks. With default clock rate for about 40 Mhz, it can response to digital signal for about 25ns.

3) Reliability - FPGA utilized deterministic hardware which is dedicated to every task and its parallelism ability reduce the reliability concern.
4) Offload Processing – FPGA can free up the CPU on host computer by get rid the exhaustive processing.
5) Cost – FPGA is programmable silicon which neglected the fabrication cost of component circuitry assembly in contrast of ASIC.

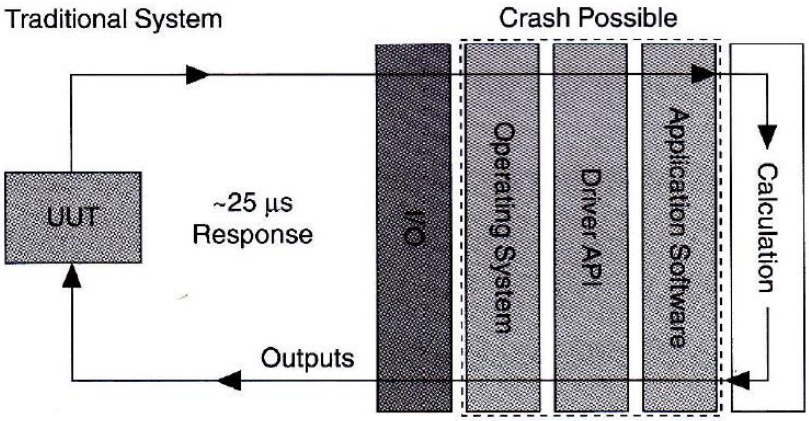



**Figure 9: FPGA clock speed [17]**

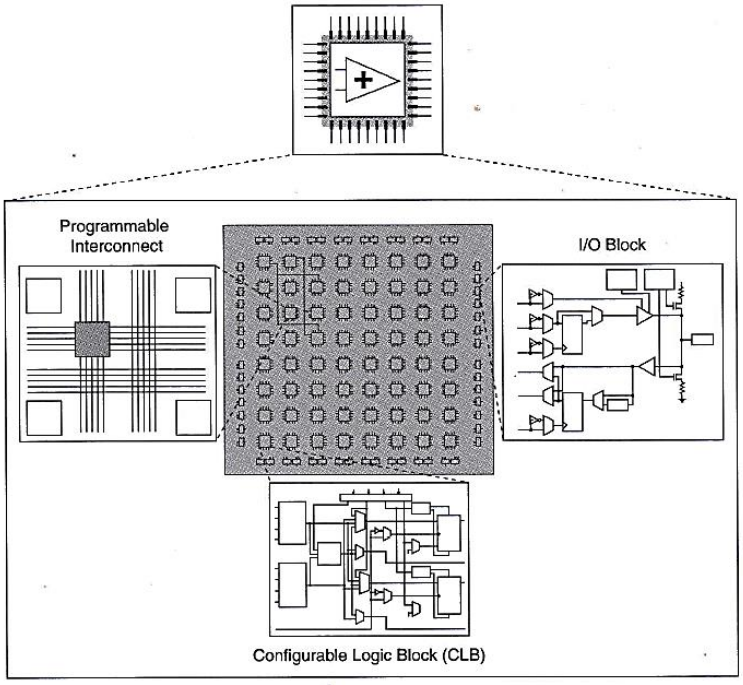



**Figure 10: FPGA technology [17]**

#### 2.3.1.2.1 The operation of *LabVIEW* FPGA

When *LabVIEW* IV is compiled into FPGA hardware by *LabVIEW* FPGA module, the *LabVIEW* IV itself is converted into test-based VHDL code. Based on this code, then the Xilinx ISE Compiler create the hardware circuit realization of the related *LabVIEW* design in term of bit file. This bit file then is loaded to FPGA chip and reconfigure the gate array logic (Refer figure 10 Below).

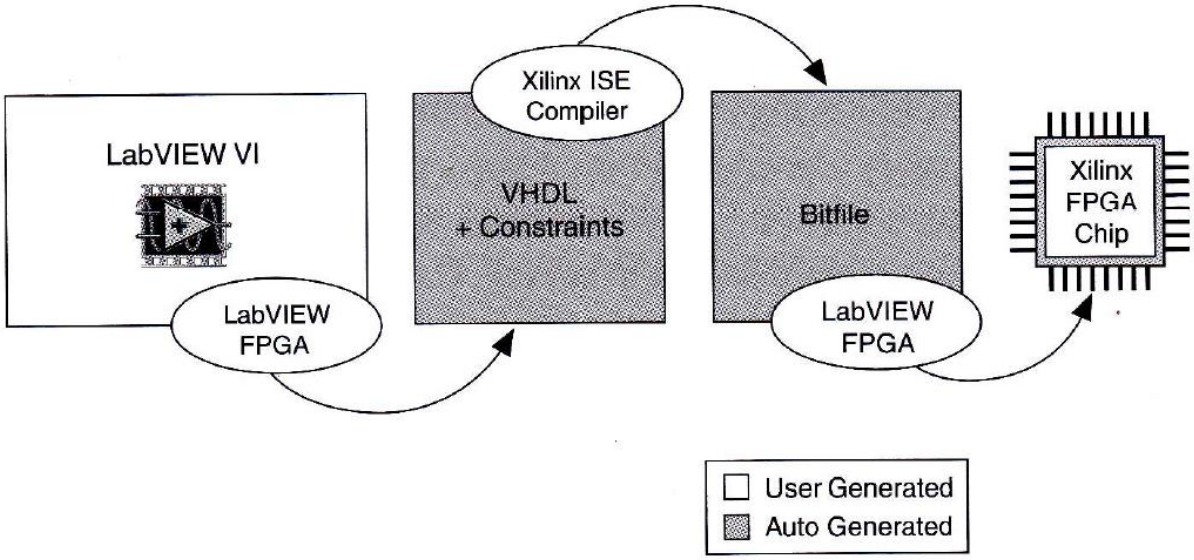
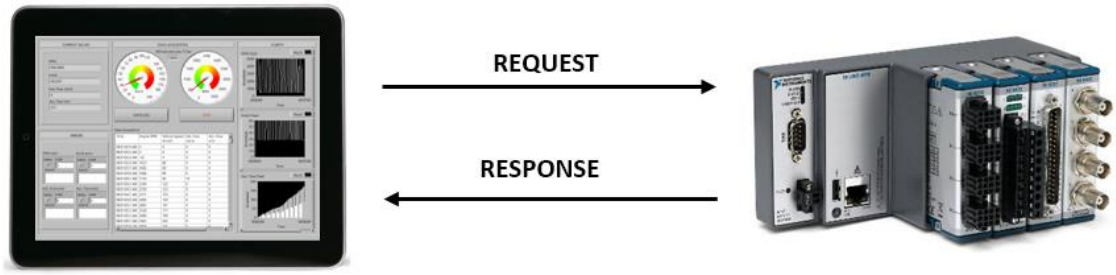



**Figure 11: How *LabVIEW* FPGA works [17]**

### 2.3.1.3 Web service

As of the third objective of this project, is to build the GUI in windows-based tablet PC to implement the in-car real time monitoring, control and data acquisition form the EMS controller. In this case, the EMS controller program is built in *CompactRIO*, cRIO-9076. This communication pattern between *CompactRIO* and windows-based tablet PC will be done in 'client-server model'. The *CompactRIO* act as an embedded server and windows-based tablet PC act as the client or human-machine interface (HMI) (Refer figure 11 below). [6]




**Windows-based Tablet PC [1] (Client/HMI)**

***CompactRIO*** **(Embedded Server)**

**Figure 12: Embedded system serving content to client machine [6]**

If the embedded server comprehends the client request, then it will reply with a response. If not then, it will reply with an error. Both of these hardware must speak the same language called Hypertext Transfer Protocol (HTTP) to mutually understand each other request and response. The program in the embedded server that is able to communicate over HTTP is termed as *'web service'.* It basically handles the client request and reply with a response according to the developer program. Meanwhile, the program in the client computer is called *'client application'.* Its task is basically interpreting the response of web service according to developer programme. [6]

This data of request and response is called content. The examples of content are image, sounds, text and a lot more. Each of this content have their own standard. When the web service gives a response to web server, this web server will interpret this content into web page. This process is called rendering. [6]

The content itself can be categorized by static and dynamic content. For 'static content 'it is a not-executed and just transmitted content such as of plain text, image and XML file. While the 'dynamic content' is generated by executing programming code. Hence for this project the FPGA programming data execution is categorized as the 'dynamic content'. [6]

Before the full usage of internet, the communication between HMI and embedded server is done using TCP/IP which is a custom protocol or Modbus which is an industrial standard. This custom protocol has higher cost of maintenance and low interoperability while industrial standard is lack or security or scalability. [6] Hence, because of that the web service technology is being created and has more merit than custom protocol and industrial standard. The following is the advantage of web service technology.

1) Cross Platform – The embedded target that utilize web service can support most HMIs.
2) Connection management – The HTTP language enables persistence connection and termination to the point.
3) Multi-client – It can operate multi HTTP at the same time.
4) Security – This web service can encrypt the transmitted content and also have ability to verify the party operate the server throughout authentication process which require user name and password.

## 2.4 In-Wheel Motor (IWM) System

To enhance the efficiency of power distribution and energy saving for hybrid vehicle, the re-generative braking system sourced from *Kelly Motor Controls* is applied in this split-parallel hybrid design. This system is a combination of mechanical braking and electrical re-generative braking. It will enable the kinetic energy loss during braking to be captured and converted into electrical energy. This captured electrical energy will be stored in the battery bank. [4]

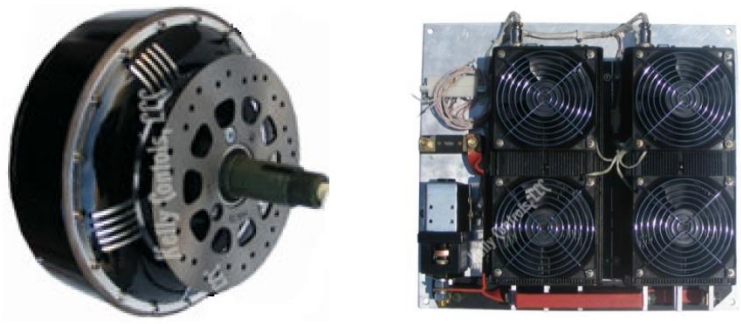


**Figure 13: IWM (left) and motor controller (right) [4]**

## 2.5 Re-generative Braking System (RBS)

This system occurs at the braking pedal of the car. When the pedal is pressed for the first 30% of initial pedal position, the electrical energy is harvested from IWM. This harvested energy will still be depended on battery SOC. The re-gen breaking will be little when the battery is nearly full. This will affect the effectiveness of RBS. Meanwhile, for the remaining 70% of the pedal travel zone, the mechanical breaking

is used. This mechanical braking will utilize OEM braking mechanism for safety purpose and maximum breaking performance. [4]

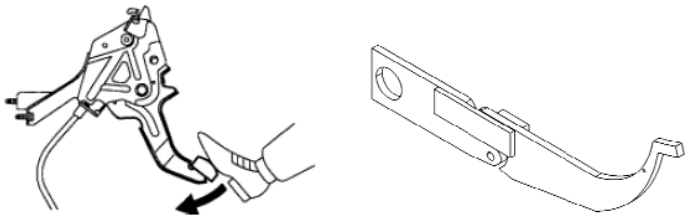


**Figure 14: Brake pedal that will be integrated with re-gen breaking system [4]**

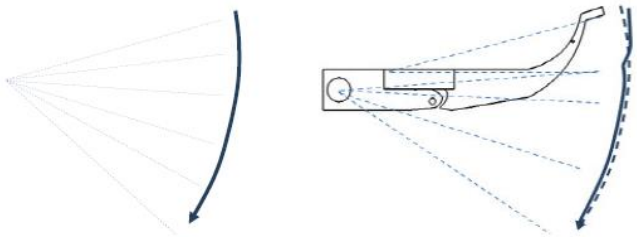
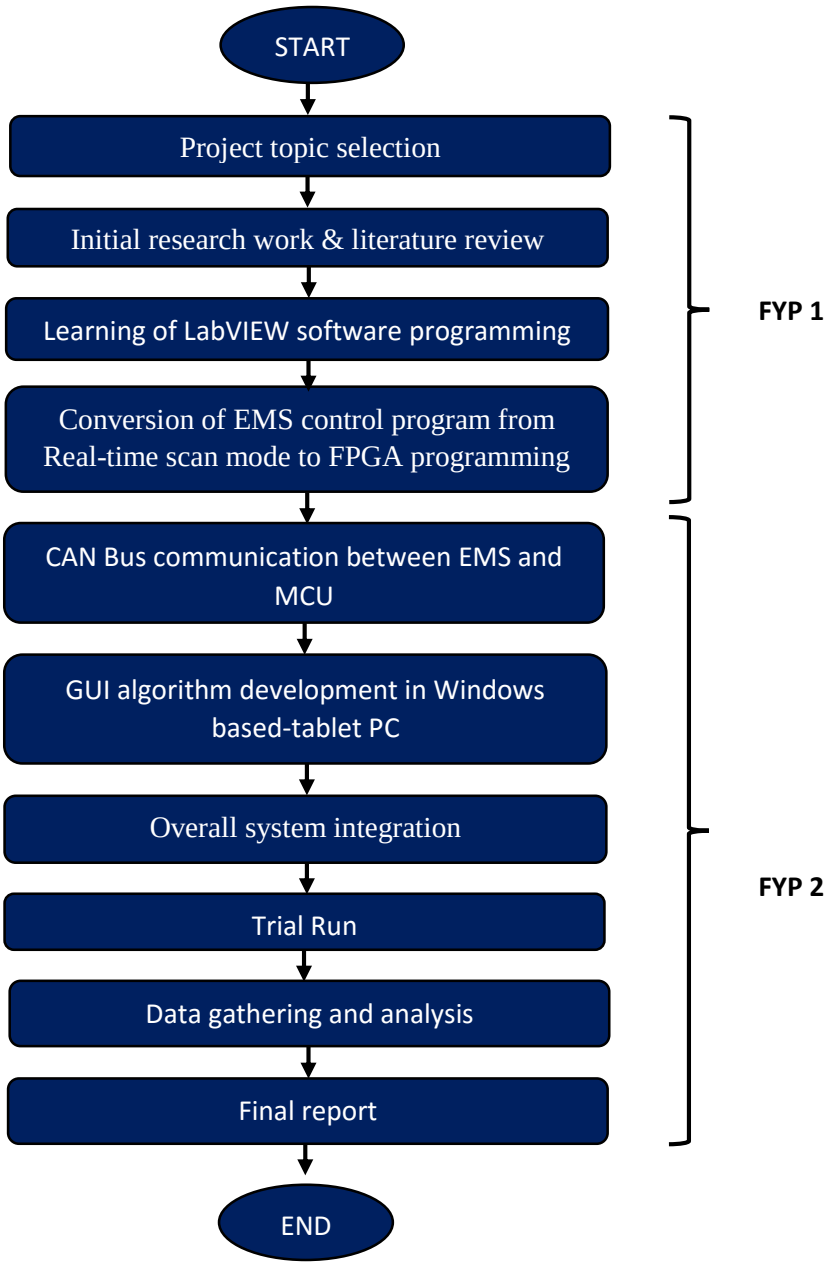


**Figure 15: Travel pattern of re-generative breaking [4]**

# CHAPTER 3
# METHODOLOGY

## 3.1 Research Methodology



## 3.2 Implementation Concept

### 3.2.1 Pictorial Schematic

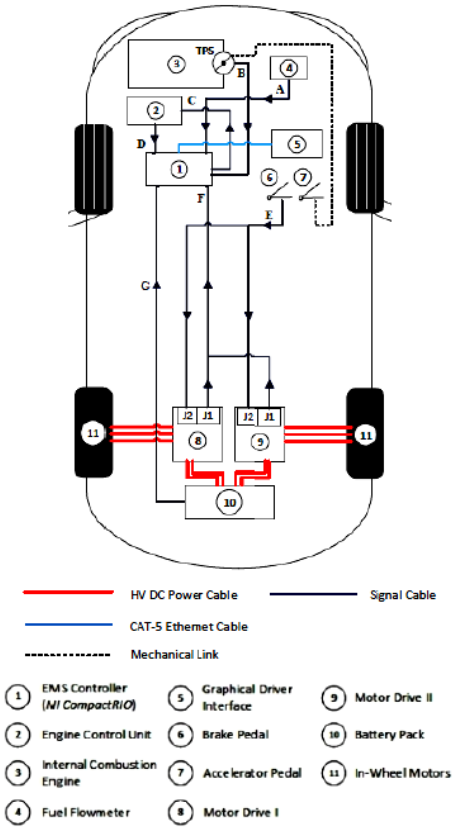



**Figure 16: EMS layout and connectivity [3]**

### 3.2.2 Block Diagram

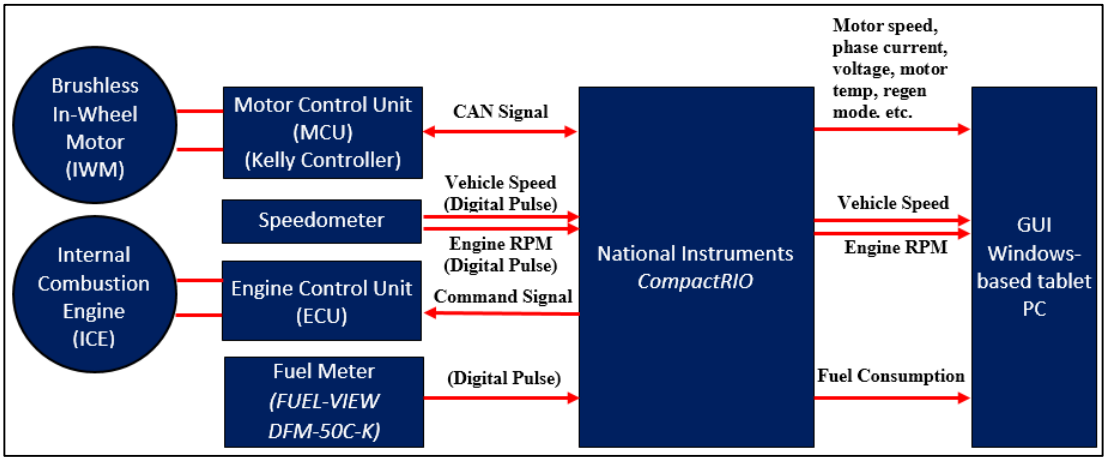



**Figure 17: Block Diagram**

### 3.2.3 Input/ Output List

| No. | Subsystem | I/O Type | Description |
|---|---|---|---|
| 1. | Fuel flow | Digital pulse | Tapped from Fuel Meter, 5 ml/pulse |
| 2. | Vehicle speed | Digital pulse | Tapped from speedometer |
| 3. | Engine speed | Digital pulse | Tapped from speedometer |
| 4. | Motor speed, phase current, voltage, motor temp, regen mode. etc. | CAN signal | Tapped from Kelly Controller |

**Table 1: Input/ Output list**

## 3.3 Project Activities

### 3.3.1 Project background and literatures

The project starts by investigating the background of the project such which are the crucial topics that are related for this project. Then, the research on the literature of each topic are made for better understanding of the overall project content itself. The crucial topics are HEV architecture, EMS, *CompactRIO* and the modules used in this project, FPGA, web service, IWM and RBS. Then, the step-by-step procedure is planned to achieve the objectives of this project.

### 3.3.2 Learning *CompactRIO* & *LabVIEW* Real-time Scan Mode program

The overall learning process is a self-learning process with supervision from FYP supervisor. It starts with the installation of *LabVIEW* software in the PC and in the *CompactRIO*, the connection between hardware and the software configuration to prepare the *CompactRIO* and PC so that everything is working fine during the next process later on. For the Real-time Scan mode program, the learning is assisted by the example from the National Instrument web site itself, and the *LabVIEW* training

module. Most of the time the learning process is done throughout the internet and National Instrument forum.

### 3.3.3 Verification of existing Real-time Scan Mode program

The verification of existing program is done to test the understanding toward existing Real-time (RT) Scan mode algorithm. It is basically the remake of this program. This step also is crucial for the author to get familiar with the *LabVIEW* type of programming.

### 3.3.4 Learning *LabVIEW* FPGA program

The learning process of FPGA program also is a self-learning process with supervision from FYP supervisor. As of Real-time (RT) Scan mode program, the learning is assisted by the example from the National Instrument web site itself, and the *LabVIEW* training module and most of the time the learning process is done throughout the reading from internet, the National Instrument forum and trial and error. There are some difficulties in constructing the FPGA that is noticeable than the construction of program in Real-time Scan mode.

i. The pallet in FPGA program is more lesser than in RT scan mode. Some important pallet that is found in RT mode cannot be found in FPGA panel.
ii. The compilation time of FPGA program is roughly more than 9 min and has the tendency to fail rather than success. This failure is mainly due to the very long unfinished and unusual compilation time where the compilation is mainly got stuck during the translation process.
iii. The input taken from the module need to be program not as RT mode where, it has special digital configuration depending on the type of module. For example, in RT mode, the input can be configured to be a counter, counter driven output, PWM or quadrature. Meanwhile, for FPGA mode the user need to program himself to get the certain configuration.

The FPGA program required the deep understanding toward the type of usage of each pallet itself before it can be easily use. Luckily, the National Instrument training

module and internet provided the example need to understand the application of the related pallet.

### 3.3.5 Translation of existing Real-Time Scan mode program to FPGA mode

The translation process of the existing program to FPGA mode still uses the same formula and method as of in Real-time Scan mode program. The translation of the program involving TWO process. First is the data acquisition and processing in FPGA programming. Second is data interfaced for end-user front panel in real-time programming.

All the data is obtained and calculated in FPGA mode. This is because this FPGA mode clock is faster the real-time mode clock. Therefore, all the data can be obtained and calculated faster. These processed data are then transferred to real-time programming throughout FIFO function for end-user data display interface. This transfer is done due several reasons as below;

- In FPGA mode, for everything that are modified even for as small modification in the block diagram arrangement or wire, it require the whole program to be compile back and this takes a lot of time which is normally for about 15 minutes. This is really troublesome because trial and error require a lot of modification to be done.
- In real-time programming the program can be compiled faster than in the FPGA mode for about half a second. Hence, all the trial and error process regarding the program algorithm is done in real-time mode.

### 3.3.6 Trial Run

The trial runs are performed for verification of existing program to ensure the understanding towards current algorithm is perfect and also for every objective completed. As of the first objective, the result from FPGA algorithm is being compared to Real-time Scan mode program to ensure the result is comparable. It is not done to increase accuracy of the reading.

## 3.4 PROJECT ACTIVITIES & KEY MILESTONES

| FYP 1 | | | |
|---|---|---|---|
| **No.** | **Activities** | **Start Date** | **End Date** |
| 1. | Project topic selection | **21/9/15** | **4/10/15** |
| 2. | Initial research work & literature review | **28/9/15** | **1/11/15** |
| 3. | Submission of extended proposal | **30/10/15** | **30/10/15** |
| 4. | Learning of *LabVIEW* software programming | **26/10/15** | **27/12/15** |
| 5. | Proposal defence | **18/11/15** | **18/11/15** |
| 6. | Conversion of EMS control program from Real-time Scan mode to FPGA programming | **23/11/15** | **27/12/15** |
| 7. | Writing of interim report | **10/12/15** | **17/12/15** |
| 8. | Submission of draft interim report | **17/12/15** | **17/12/15** |
| 9. | Submission of interim report | **24/12/15** | **24/12/15** |

**Table 2: FYP 1 activities and key milestones**

| FYP 2 | | | |
|---|---|---|---|
| **No.** | **Activities** | **Start Date** | **End Date** |
| 1. | Conversion of EMS control program from Real-time Scan mode to FPGA programming. | **18/1/16** | **20/3/16** |
| 2. | CAN Bus communication between EMS and MCU | **14/3/16** | **17/4/16** |
| 3. | Submission of progress report | **9/3/16** | |
| 4. | GUI algorithm development in windows based-tablet PC | **28/5/16** | **17/4/16** |
| 5. | Overall system integration | **4/4/16** | **17/4/16** |
| 6. | Trial run | **4/4/16** | **17/4/16** |
| 7. | Data gathering and analysis | **11/4/16** | **17/4/16** |
| 8. | Writing of final report | **11/4/16** | **17/4/16** |
| 9. | Submission of final report | **18/4/16** | |

**Table 3: FYP2 activities and key milestone**

### 3.5 GANTT-CHART

| FYP 1 | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **No.** | **Project Activities / Milestones** | **Week** | | | | | | | | | | | | |
| | | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** |
| 1. | Project topic selection | | | | | | | | | | | | | | |
| 2. | Initial research work & literature review | | | | | | | | | | | | | | |
| 3. | Submission of extended proposal | | | | | | ◆ | | | | | | | | |
| 4. | Learning of *LabVIEW* software programming | | | | | | | | | | | | | | |
| 5. | Proposal defence | | | | | | | | | ◆ | | | | | |
| 6. | Conversion of EMS control program from Real-time Scan mode to FPGA programming | | | | | | | | | | | | | | |
| 7. | Writing of interim report | | | | | | | | | | | | | | |
| 8. | Submission of draft interim report | | | | | | | | | | | | | ◆ | |
| 9. | Submission of interim report | | | | | | | | | | | | | | ◆ |

**Table 4: FYP 1 Gantt-chart**

| FYP 2 | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **No.** | **Project Activities / Milestones** | **Week** | | | | | | | | | | | | |
| | | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** |
| 1. | Conversion of EMS control program from real-time scan mode to FPGA programming | | | | | | | | | | | | | | |
| 2. | CAN Bus communication between EMS and MCU | | | | | | | | | | | | | | |
| 3. | Submission of progress report | | | | | | | | ♦ | | | | | | |
| 4. | GUI algorithm development in windows based-tablet PC | | | | | | | | | | | | | | |
| 5. | Overall system integration | | | | | | | | | | | | | | |
| 6. | Trial run | | | | | | | | | | | | | | |
| 7. | Data gathering and analysis | | | | | | | | | | | | | | |
| 8. | Writing of final report | | | | | | | | | | | | | | |
| 9. | Submission of final report | | | | | | | | | | | | | ♦ | |

**Table 5: FYP 2 Gantt-chart**

## 3.6 TOOLS

### 3.6.1 Software

1. *LabVIEW* 2013
2. *LabVIEW* run-time engine 2013
3. *LabVIEW* FPGA module 2013
4. *LabVIEW* Real-time module 2013
5. Microsoft Office

### 3.6.2 Hardware

1. National Instrument CompactRIO (Crio-9076)
2. National Instrument NI 9401 (8 Channel digital I/O)
3. National Instrument NI 9853 (High speed CAN)
4. FUEL-VIEW DFM-50C-K
5. 12 VDC Power supply
6. 24V Battery
7. Digital multimeter
8. Ethernet cable

# CHAPTER 4
# RESULT AND DISCUSSION

## 4.1 Verification of existing Real-Time scan mode program

The verification of existing program only involves FOUR parameters which are instantaneous fuel consumption (ml/s), total fuel consumption, vehicle speed (km/h) and Engine RPM. All these parameters formula made by following the existing program data as reference.

### 4.1.1 Instantaneous fuel consumption (ml/s) and total fuel consumption (ml)

The input of for both of these parameters comes from fuel meter called 'FUEL_VIEW DFM 50C-K' which give the normally high output voltage which is 12V. Whenever the fuel is consumes by car, this output voltage will be lower down to 0.7V for amount of 80ms.

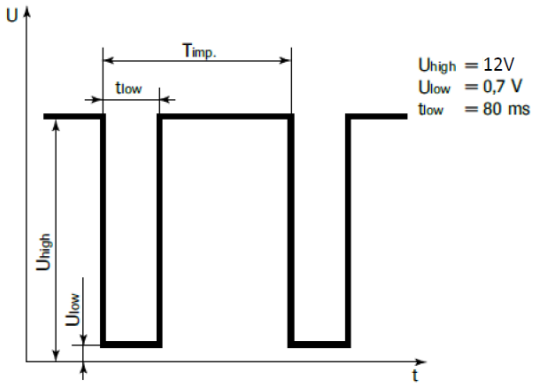



***Figure 18: Scaled pulse for DFM 50C-K [2]***

This means for every fuel consumption pulse is generated throughout falling edge. One falling edge is considered as one pulse. In *CompactRIO*, this pulse will be read by digital input module which is 9401. Both instantaneous and total fuel consumption will use the same digital input/output pins of this module. The pin for instantaneous and total fuel consumption is set to be DIO0 and is initialized as below.

*I. Initialization I/O pin*

This DIO0 is set to count the falling edge of fuel meter output pulse as below. Note that, the CTR0 = DIO0.

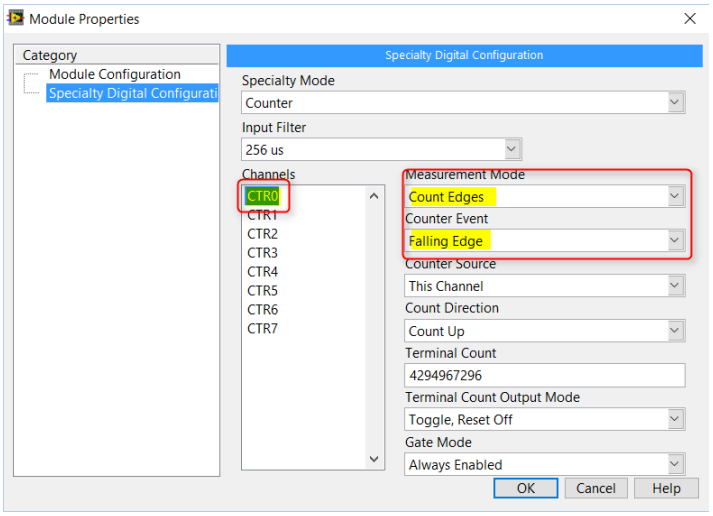



**Figure 19: Both DIO0 is initialized to sense falling edges**

The input filter is set to be 256us due to the analyzation of fuel meter to output where one pule is equivalent to 5ml of which in produce in the range of second minimum. The variation outside of this time frame will be ignored. Hence, the input filter must me much faster than the pulse production. This change is considered as noise.[2]

*II. Data processing*

The reading of the pulse is design to be taken for every 500ms. This pulse is counted by calculating the difference of previous and current counter. This pulse count now is considered as pulse per 500ms.

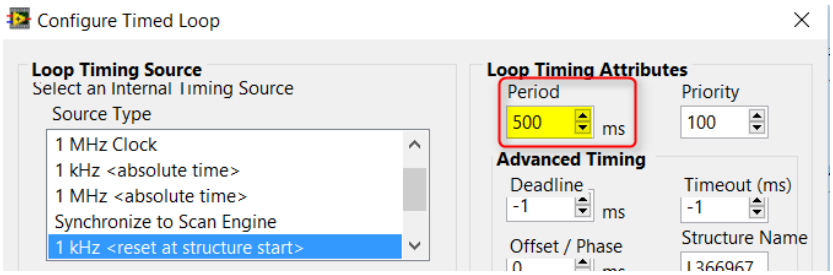



**Figure 20: Setting the pulse read*ing time to be execute for every* 500ms**

Since, one pulse is equivalent to 5ml and which has the minimum consume time of a second, therefore the current pulse need to be multiply with 5 to make the pulse in millilitre reading and multiply again by 2 to make it into millilitre per second (ml/s). This ml/s is the instantaneous fuel consumption.

Meanwhile, for total fuel consumption is a measurement of accumulation of fuel consumption. This accumulation can be made by calculating the addition of current millilitre litre reading to previous millilitre reading. The total fuel consumption is calculated in ml. The program for both instantaneous and total fuel consumption parameters are as below.

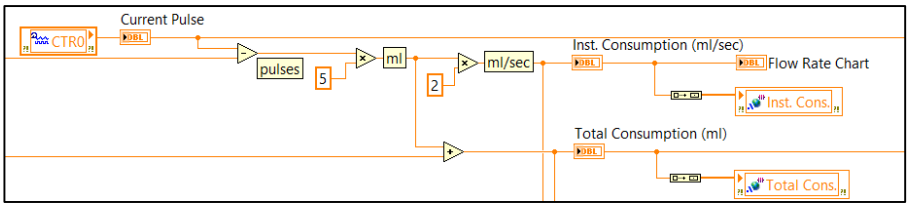



**Figure 21: Fuel-flow measurement in real time scan mode**

### 4.1.2 Vehicle speed (km/h) and Engine RPM

*I. Initialization of I/O pin*

The input of the vehicle speed and engine RPM is read by tapping the wire from the dashboard of car itself to read the frequency of both parameters. This frequency still will be read by NI 9401 module. Due to limitation of NI 9401 where it can only read the frequency above

500Hz, so there is possibility that the frequency will be infinite. This problem happens because the frequency measurement depended on the pre-set sampling time of the module but not the period measurement configuration. Hence, the pin DIO1 and DIO2 is set to read the period from both vehicle speed and engine RPM. Therefore, both of this pins need to set to period measurement configuration. Note that DIO1=CTR1 and DIO2=CTR2.

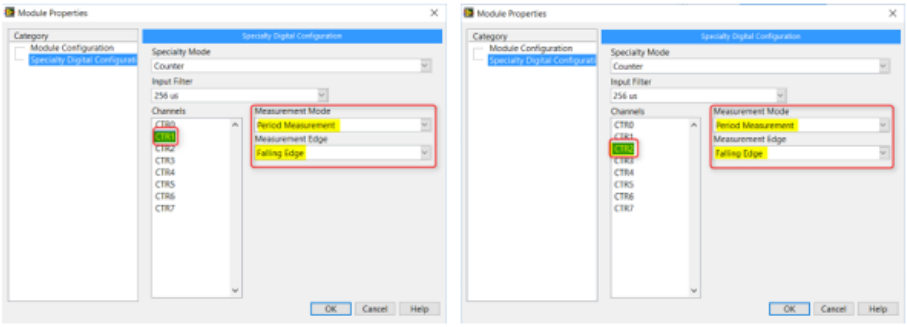



**Figure 22: Both DIO1 and DIO2 is initialized to read time period**

*II. Data processing*

The time period for each of the vehicle speed and engine RPM later is convert into frequency by the formula of frequency = 1 / Period.

Since, the pin is set to read period from both vehicle speed and engine RPM, there is a possibility that the period value to be zero which will cause the frequency to be infinite. Hence, the value 839000 is chosen to give negligibe vehicle speed value if the program detect the zero period value. (((1/839000)x1000000)x1.487)-1.7628 = 0.0095 km/h [2]. The other calculation is just follow the linear regression formula that is made based on signal conditioning linear regression equation for vehicle speed below made by Kurniawan, Y. (2013).

Vehicle speed in km/h = 1.487 (frequency from vehicle speed sensor) + 1.768

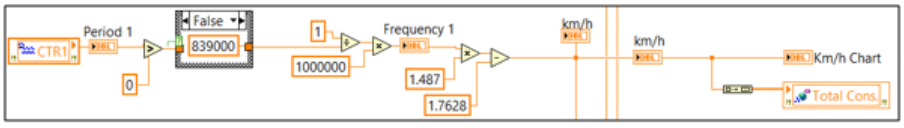



**Figure 23: Vehicle speed measurement in real time scan mode**

For, engine RPM the program is perform in similar manner and is made by following the linear regression equation for engine RPM below.

RPM = 30.078 (frequency from rpm sensor) + 160.02


**Figure 24: Engine RPM measurement in real time scan mode**

### 4.1.3 Data Acquisition

The data from FOUR parameters of instantaneous fuel consumption (ml/s), total fuel consumption (ml), vehicle speed (km/h) and engine RPM need to be recorded for further analysis. There are TWO type of data acquisition implemented in this program. First, is saving the data in *CompactRIO* drive and second is data tabulation in front panel purpose.

*I. Saving data in CompactRIO memory*

Each of the FOUR parameters is shared throughout the network-published shared variable called RT FIFO. This RT FIFO enable to create communication loop by transferring the data from deterministic loop to the host over the network. These shared variables than are used to save the four parameters data into the *CompactRIO* drive with the tdms format. The name of the folder to save this file need to be initialize of which in this case is RT-RVF (green colour) and the name of the tdms file is RT-RVF (Pink colour). The program to save the parameters data into *CompactRIO* memory is as below.

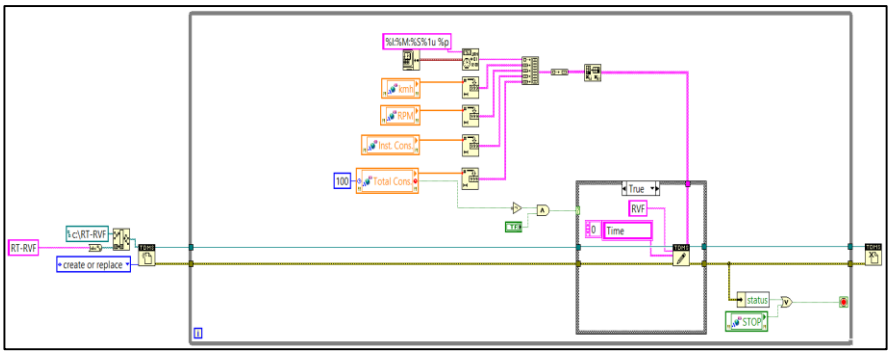



**Figure 25: Saving data in CompactRIO memory program**

*II. Data tabulation in front panel*

The program to tabulate all the related parameters in front panel is as followed;

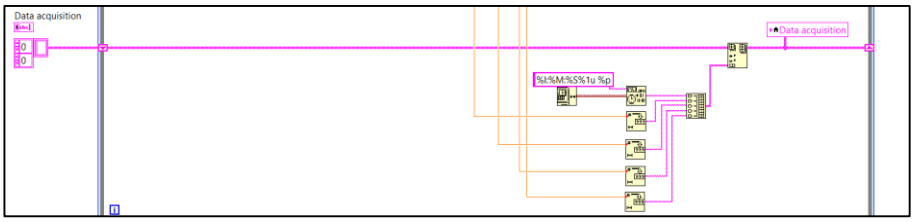



**Figure 26: Tabulation of data program**

### 4.1.4 Front Panel

This front panel enables all the related parameters data to be monitor. The data check column in yellow box is for monitoring the overall parameters data. The gauge in green box will display the vehicle speed and engine ROM data in real time. The table in red box will tabulate the engine RPM, vehicle Speed, instantaneous and total fuel consumption data. While, the charts in blue box are used to tabulate the related data.

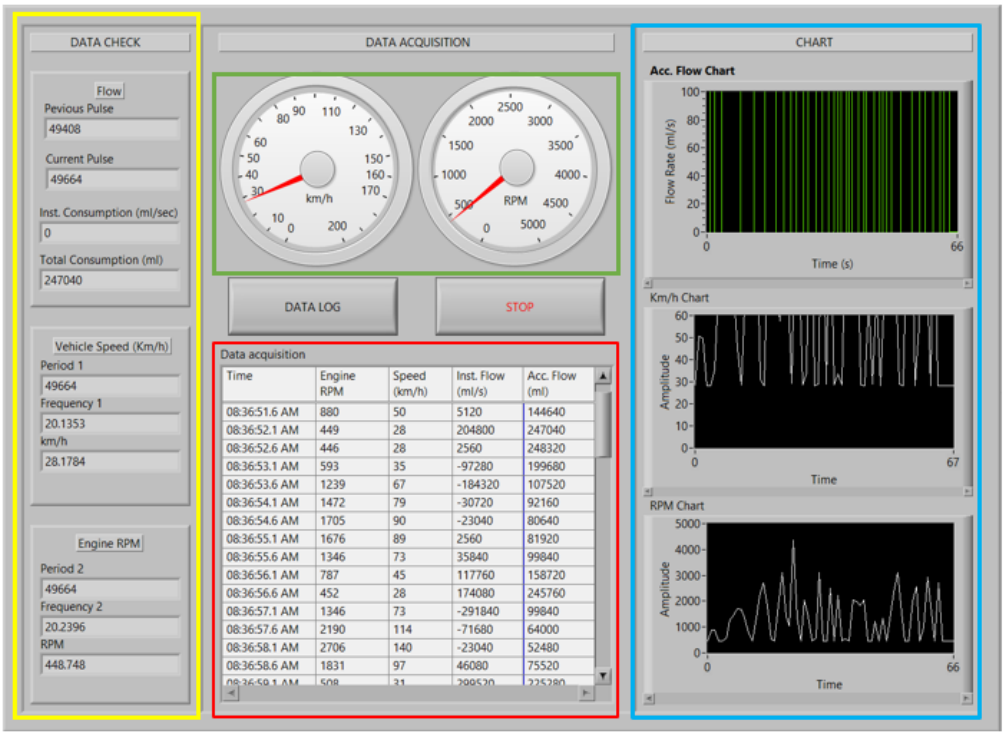



**Figure 27: : The user interface of related parameters measurement**

### 4.1.5 Verification of data

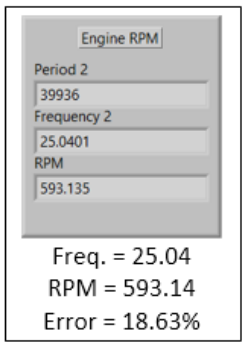
The data obtain need to be verify to ensure the reliability of existing program before translating the existing program to FPGA mode. The verification is done by comparing the testing results with the data obtained by Kurniawan, Y. (2013) which is only limited to vehicle speed and engine RPM. The verification between both data is compared throughout the error measurement formula where, *[Error %= ((Original Value – Measured Value)/Original Value) x 100]*. The data is considered valid as long as long as the error percentage is less than 20%. Note that, the all the value calculated and displayed is converted into TWO decimal place.

*I. Engine RPM vs frequency*

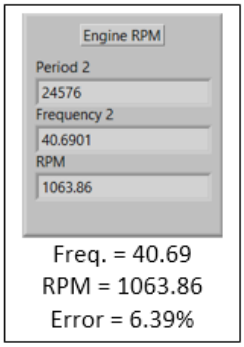
For the below error calculations, the original value is the engine RPM value obtained by Kurniawan, Y. (2013) and the measured value is the testing result value.

| f (Hz) | RPM |
|---|---|
| 25 | 500 |
| 40 | 1000 |
| 52 | 1500 |
| 70 | 2000 |
| 90 | 2500 |
| 100 | 3000 |
| 124 | 3500 |
| 140 | 4000 |

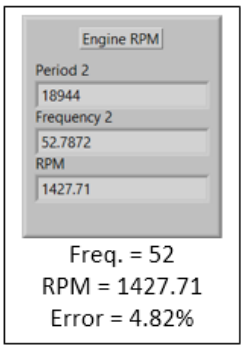
**Table 6: RPM vs Frequency [2]**



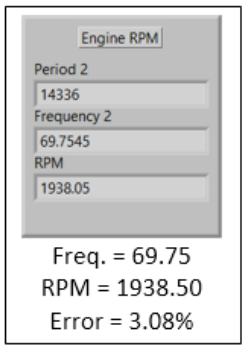

Freq. = 25.04
RPM = 593.14
Error = 18.63%



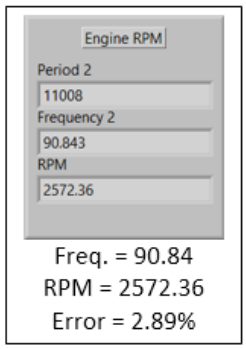

Freq. = 40.69
RPM = 1063.86
Error = 6.39%



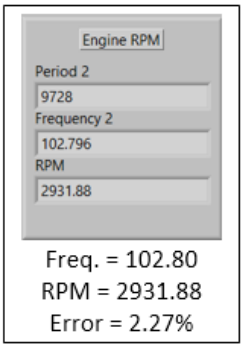

Freq. = 52
RPM = 1427.71
Error = 4.82%



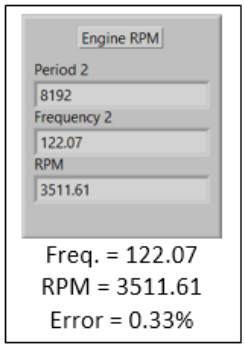

Freq. = 69.75
RPM = 1938.50
Error = 3.08%



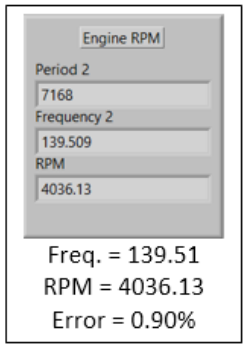

Freq. = 90.84
RPM = 2572.36
Error = 2.89%




Freq. = 102.80
RPM = 2931.88
Error = 2.27%




Freq. = 122.07
RPM = 3511.61
Error = 0.33%




Freq. = 139.51
RPM = 4036.13
Error = 0.90%

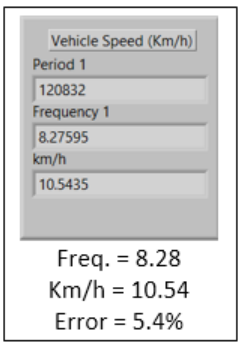
As of above results, all the percentage of errors for all the engines' RPM values are less than 20%. Therefore, the testing result value are accepted.

*II.* *Vehicle speed vs frequency*

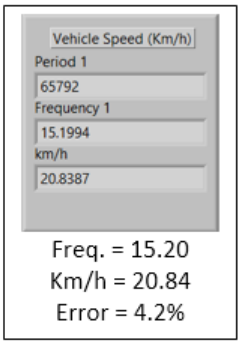
For the below error calculations, the original value is the vehicle speed (km/h) value obtained by Kurniawan, Y. (2013) and the measured value is the tested result value.

| f (Hz) | Speed |
|---|---|
| 8.3 | 10 |
| 15 | 20 |
| 20 | 30 |
| 28.5 | 40 |
| 35 | 50 |

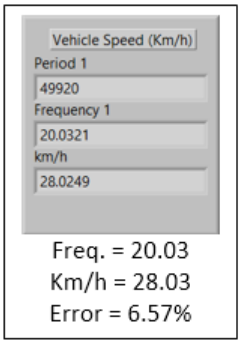
**Table 7: Vehicle Speed vs Frequency *[2]***



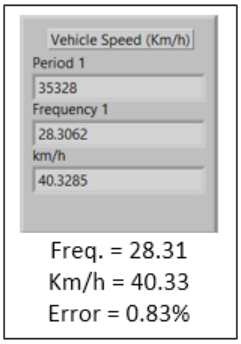
Freq. = 8.28
Km/h = 10.54
Error = 5.4%



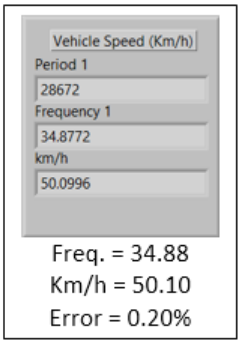
Freq. = 15.20
Km/h = 20.84
Error = 4.2%



Freq. = 20.03
Km/h = 28.03
Error = 6.57%



Freq. = 28.31
Km/h = 40.33
Error = 0.83%



Freq. = 34.88
Km/h = 50.10
Error = 0.20%

As of above results, all the percentage of errors for all the vehicle speed (km/h) values are less than 20%. Therefore, the testing result values are accepted.

## 4.2 Translation of existing Real-time Scan mode program to FPGA mode

The translation process of the existing program to FPGA mode still uses the same formula and method as of in Real-time Scan mode program. Therefore, the result of this translation is comparable to the existing program. For this conversion step, the data will be obtained and calculated in FPGA mode. Then these data will be transferred to Real-Time program for end user interface.

### 4.2.1 Data acquisition and calculation in FPGA mode

#### 4.2.1.1 Instantaneous fuel consumption (ml/s) and total fuel consumption (ml)

Since the input of the Instantaneous fuel consumption (ml/s) and total fuel consumption (ml) sill comes from ‘FUEL_VIEW DFM 50C-K’. It means that the fuel

consumption pulse is generated throughout falling edge. Therefore, the input initialization for this parameters need to be the falling edge counter.

*I. Initialization I/O pin*

This DIO0 output is tested to be a Boolean output where it will give either 1 or 0. One is 12v, and zero is 0.7V. By utilizing the shift register concept, the current pulse is compared with previous pulse. If the value is being found that much bigger than the previous one (1>0), the comparator will output the value 1. This means that it initiate that whenever the DIO output value change from 1 to zero, this indicate the falling edge.

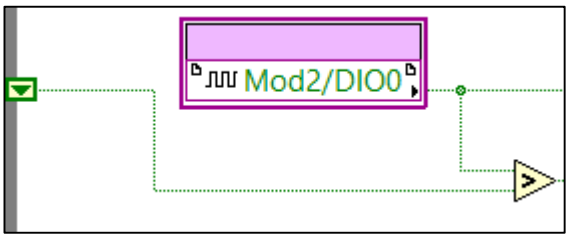



**Figure 28: DIO0 is initialized to sense falling edges**

*II. Data processing*

The reading of the pulse is design to be taken for every 500ms throughout the while loop timer.

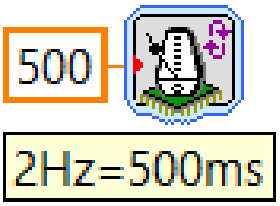



**Figure 29: The pulse reading time to be execute for every 500ms**

Since, one pulse is equivalent to 5ml and which has the minimum consume time of a second, therefore the current pulse need to be multiply with 5 to make the pulse in millilitre reading and multiply

again by 2 to make it into millilitre per second (ml/s). This ml/s is the instantaneous fuel consumption.

Meanwhile, for total fuel consumption is a measurement of accumulation of fuel consumption. This accumulation can be made by calculating the addition of current millilitre litre reading to previous millilitre reading. The total fuel consumption is calculated in ml. The program for both instantaneous and total fuel consumption parameters are as below.

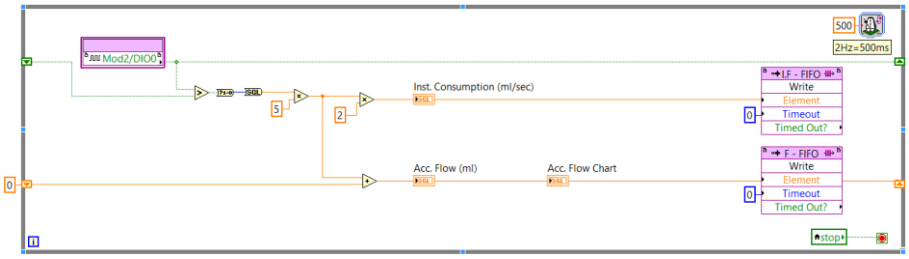



**Figure 30: Fuel-flow measurement in FPGA mode**

### 4.2.1.2 Vehicle Speed (km/h) and Engine RPM

*I. Initialization of I/O pin*

The same with the real-time scan mode program the DIO1 and DIO2 need to be configured for the period counting. Bu utilizing the frame sequence pallet, the timer will count the falling edge time and by using the shift register the current timer value will be compared the previous one to find the actual time of period.

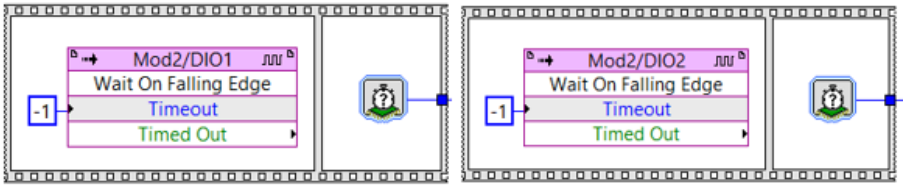



**Figure 31: Both DIO1 and DIO2 is initialized to read time of falling edge**

*II. Data processing*

The time period for each of the vehicle speed and engine RPM later is convert into frequency by the formula of frequency = 1 / Period. Then, all the formula is exactly follow the real-time scan mode program formula.

Vehicle speed in km/h = 1.487 (frequency from vehicle speed sensor) + 1.768

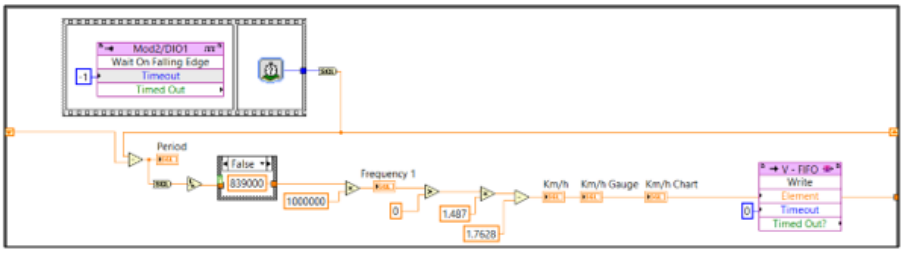



**Figure 32: Vehicle speed measurement in FPGA mode**

For, engine RPM the program is perform in similar manner and is made by following the linear regression equation for engine RPM below.

RPM = 30.078 (frequency from rpm sensor) + 160.02

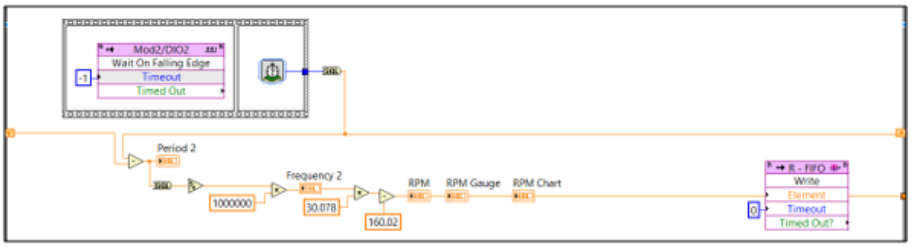



**Figure 33: Engine RPM measurement in FPGA mode**

### 4.2.1.3 Front Panel for programmer

This front panel can only be seen by the author. This front panel will only be use to check and verify the data with existing program. These data later will be transfer and display in end user's front panel using Real-time program. For the following front panel, the data check column in yellow box is for monitoring the overall parameters data. The gauge in green box will display the vehicle speed and engine ROM data in

real time. While the chart in blue box represent the accumulate fuel consumption (ml), vehicle speed (km/h) and engine RPM.


**Figure 31: The FPGA interface of related parameters measurement**

### 4.2.1.4 Verification of data

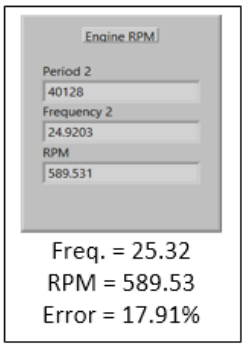
The data obtained need to be verified with the data obtained by Kurniawan, Y. (2013) of which is only limited to vehicle speed and engine RPM. The verification between both data is compared throughout the error measurement formula where, *[Error %= ((Original Value – Measured Value)/Original Value) x 100]*. The data is considered valid as long as long as the error percentage is less than 20%. Note that, the all the value calculated and displayed is converted into TWO decimal place.

*I. Engine RPM vs frequency*

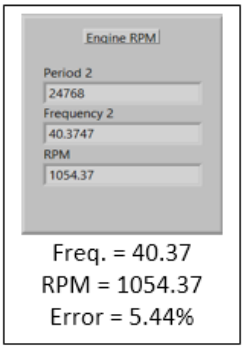
For the below error calculations, the original value is the engine RPM value obtained by Kurniawan, Y. (2013) and the measured value is the testing result value.

| f (Hz) | RPM |
|---|---|
| 25 | 500 |
| 40 | 1000 |
| 52 | 1500 |
| 70 | 2000 |
| 90 | 2500 |
| 100 | 3000 |
| 124 | 3500 |
| 140 | 4000 |

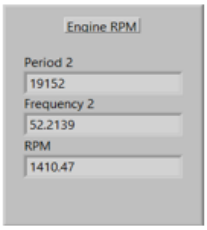
**Table 8: RPM vs Frequency [2]**



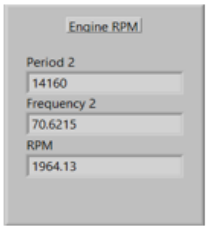
Freq. = 25.32
RPM = 589.53
Error = 17.91%



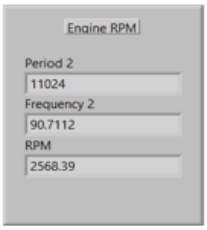
Freq. = 40.37
RPM = 1054.37
Error = 5.44%



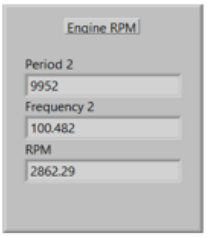
Freq. = 52.21
RPM = 1410.47
Error = 5.97%



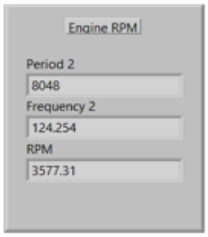
Freq. = 70.62
RPM = 1964.13
Error = 1.79%



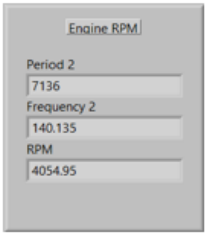
Freq. = 90.71
RPM = 2568.39
Error = 2.74%



Freq. = 100.49
RPM = 2862.29
Error = 4.59%



Freq. = 124.25
RPM = 3577.31
Error = 2.21%



Freq. = 140.135
RPM = 4054.95
Error = 1.37%

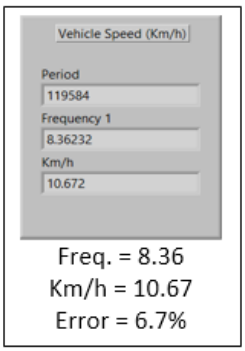
As of above results, all the percentage of errors for all the engines' RPM values are less than 20%. Therefore, the testing results values are accepted.

*II. Vehicle speed vs frequency*

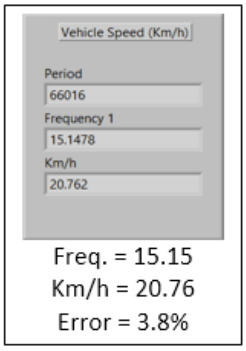
For the below error calculations, the original value is the vehicle speed (km/h) value obtained by Kurniawan, Y. (2013) and the measured value is the testing result value.

| f (Hz) | Speed |
|---|---|
| 8.3 | 10 |
| 15 | 20 |
| 20 | 30 |
| 28.5 | 40 |
| 35 | 50 |

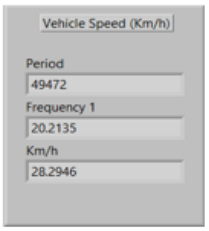
**Table 9: Vehicle Speed vs Frequency *[2]***



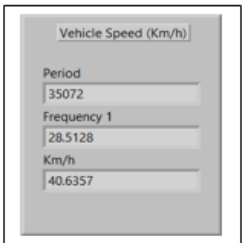
Freq. = 8.36
Km/h = 10.67
Error = 6.7%



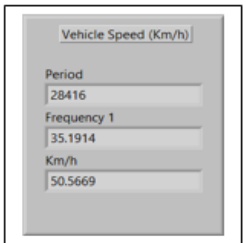
Freq. = 15.15
Km/h = 20.76
Error = 3.8%



Freq. = 20.21
Km/h = 28.29
Error = 5.7%



Freq. = 28.51
Km/h = 40.64
Error = 1.6%



Freq. = 35.19
Km/h = 50.56
Error = 1.12%

As of above results, all the percentage of errors for all the vehicle speed (km/h) values are less than 20%. Therefore, the testing result values are accepted.

### 4.2.2 Data interfacing in Real-time programming

#### 4.2.2.1 FIFO Data transfer

The data transfer from FPGA to real time program utilized the FIFO read function. Then, these data are displayed in front panel in with charts. Only vehicle speed and engine RPM utilized an additional gauge dial function display. The algorithm of this data transfer is as follow;

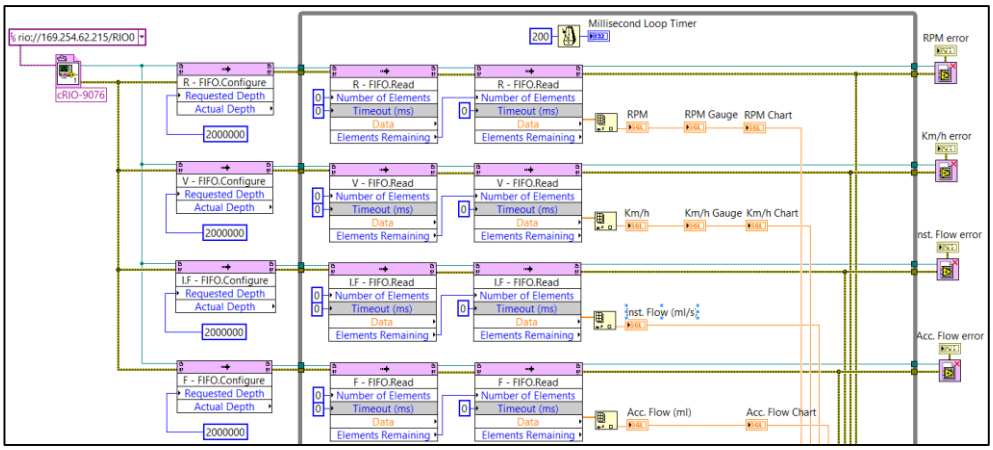



**Figure 34: Saving data in *CompactRIO* drive program**

### 4.2.2.2 Data acquisition

As of existing program the FOUR parameters of instantaneous fuel consumption (ml/s), total fuel consumption (ml), vehicle speed (km/h) and engine RPM will be saved in the *CompactRIO* drive and these data will be tabulated in front panel.

*I.* *Saving data in CompactRIO memory*

The algorithm to save the FOUR parameters in *CompactRIO* memory is as below.

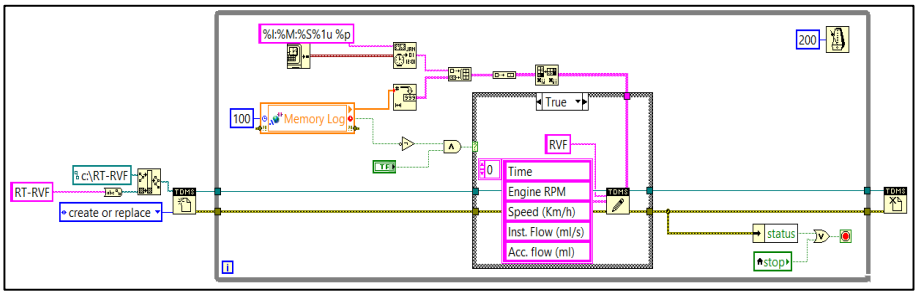



**Figure 35: Saving data in *CompactRIO* drive program**

*II.* *Data tabulation in front panel*

The program to tabulate all the related parameters in front panel is as followed;

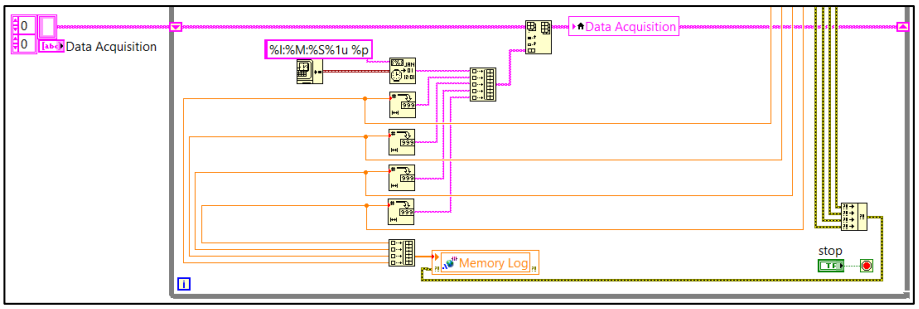



**Figure 36: Tabulation of data program**

#### 4.2.2.2.1 Front panel for end-user

This front panel enables all the related parameters data to be monitor. The data check column in yellow box is for monitoring the overall parameters data. The gauge in green box will display the vehicle speed and engine RPM data in real time. The table in red box will tabulate the engine RPM, vehicle Speed, instantaneous and total fuel consumption data. While, the charts in blue box are used to tabulate the related data.

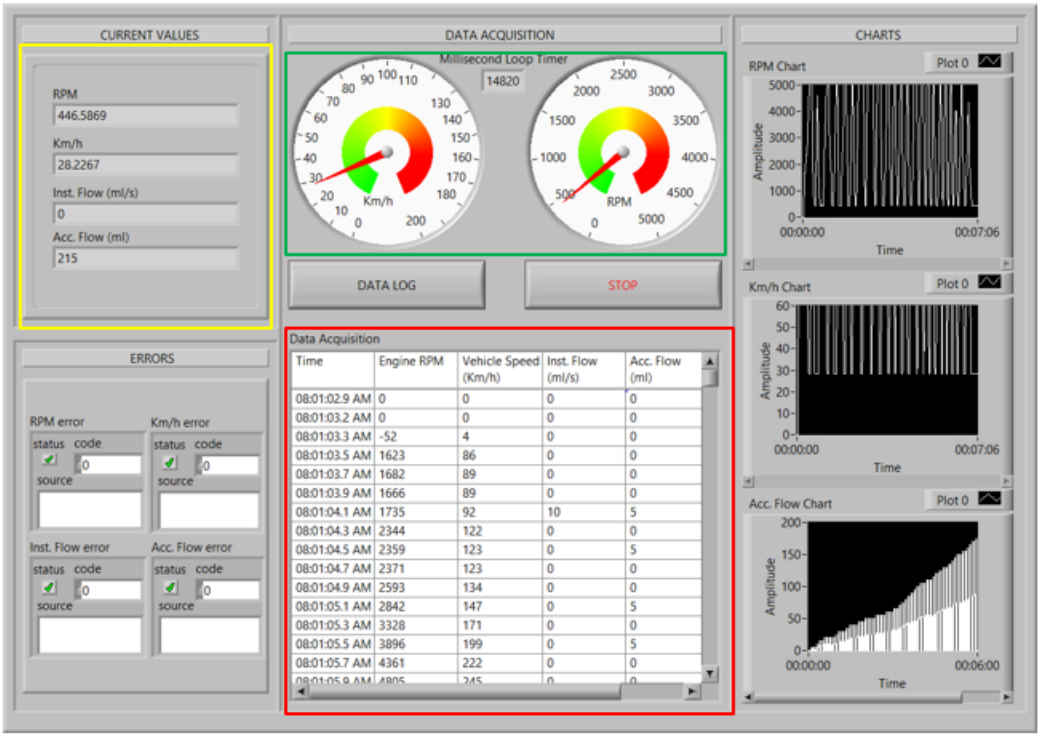



**Figure 31: The real-time mode interface of related parameters measurement**

All the data is displayed, according to original data. Hence, the program conversion is a success.

## 4.3 FPGA program for CAN bus communication between the EMS and MCU controller

To establish connection between Energy Management System (EMS) and Motor Controller Unit (MCU) controller, it requires the MCU to be setup first. The MCU utilized in this project is Kelly Controller and the EMS controller in this project is *CompactRIO* cRIO-9076. After that, the controller need to

### 4.3.1 Kelly Controller setup

#### 4.3.1.1 Connection between Kelly Controller and windows PC

To setup the controller, the MCU need to be connected to windows PC using USB to RS232 cable. In this case the extension cable will be connected to this USB to RS-232 cable to increase the cable length. The Kelly Motor need to be stop first before this connection is made because it will cause the cable to getting hot. The connection of this hardware are as follow.

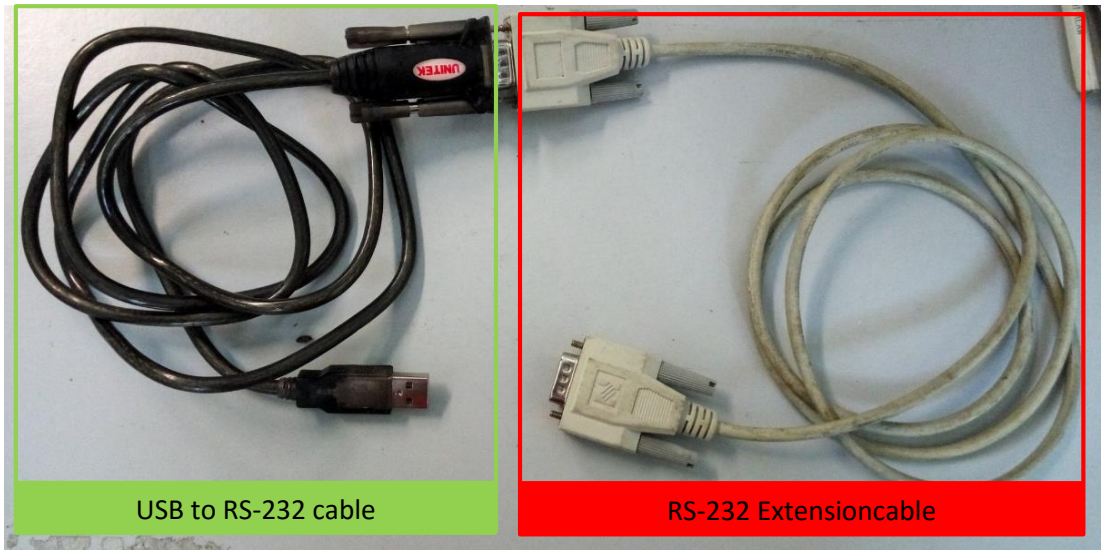



**Figure 34: USB to RS-232 and RS-232 extension cable**

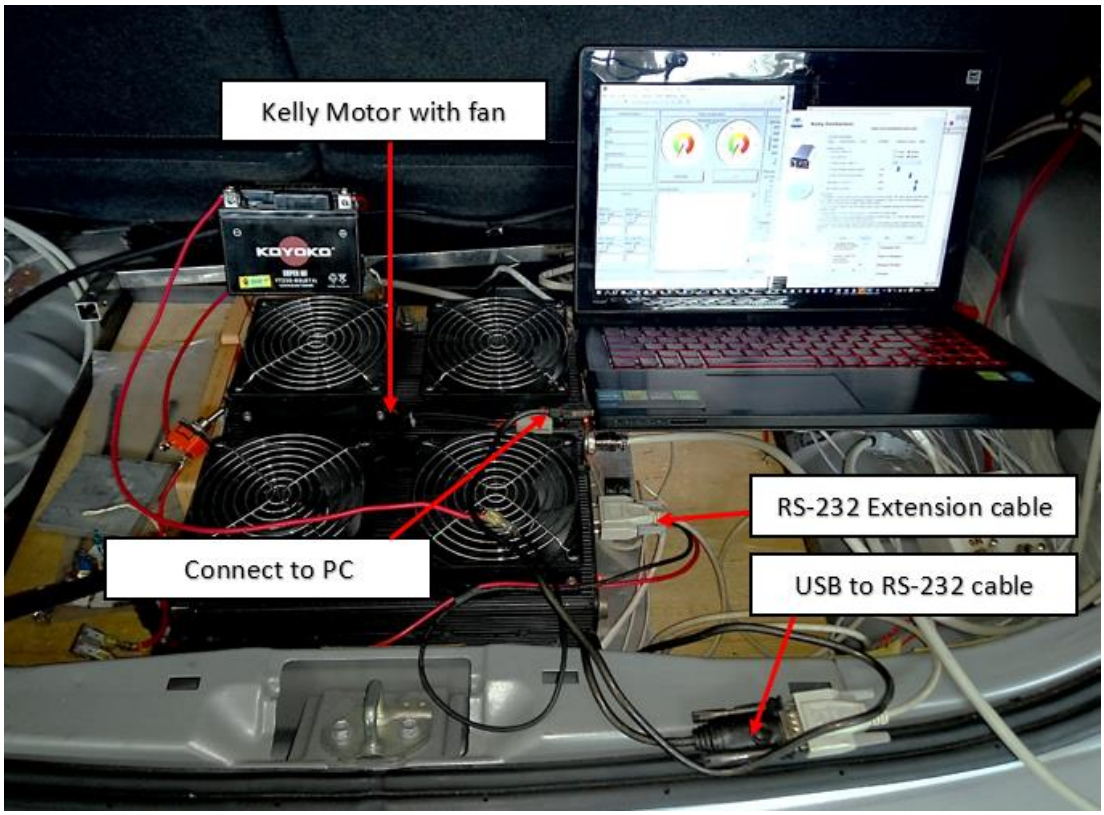



**Figure 35: Connection between Kelly controller and PC**

The setting of the Kelly Controller is as in the following figure from step 1 until step 6. This step is a standard given by manufacturer of Kelly Controller.


**Figure 36: Kelly Controller setting step 1 [21]**


**Figure 37: Kelly Controller setting step 2 [21]**


**Figure 38: Kelly Controller setting step 3 [21]**


**Figure 39: Kelly Controller setting step 4 [21]**


**Figure 40: Kelly Controller setting step 5 [21]**


**Figure 41: Kelly Controller setting step 5 [21]**

After all these steps already been follow, the Kelly Controller setting is complete. Now the USB to RS-232 cane can be disconnected from PC and Kelly controller. The Kelly Controller now is ready to be connected with *CompactRIO*. The steps to connect the controllers will be discussed in the following chapter.

### 4.3.2 Connection between Kelly Controller, EMS controller and PC

To connect Kelly Controller to *CompactRIO,* it requires wire that has end to end of RS-232 socket with Kelly J2 Cable. Since this cable is not usual in market, the cable itself need to be build. The building process will be explain as followed.

#### 4.3.2.1 Kelly Motor CAN bus cable building

This CAN bus cable is built based on the CAN bus topology below. The CAN Nodes at both side left and right of this cable in this case are RS-232 socket and Kelly J2 socket.

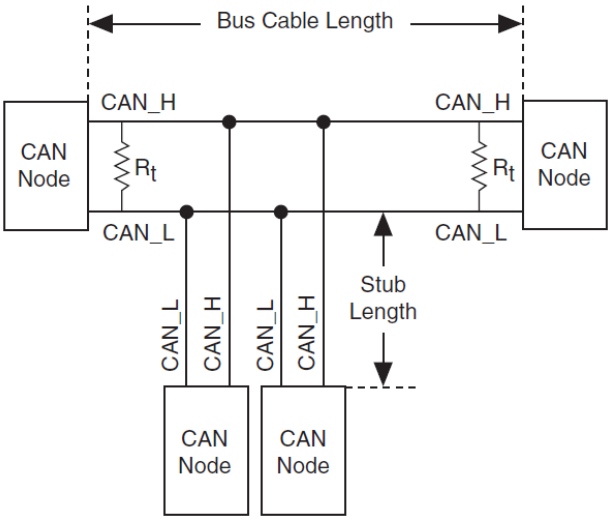



**Figure 42: CAN bus topology [19]**

The CAN_H (Pin 10) of Kelly J2 need to be soldier so that it is connected to CAN_H (Pin7) of RS-232 socket. Meanwhile, The CAN_L (Pin 11) of Kelly J2 n need to be soldier so that it is connected to CAN_L (Pin2) of RS-232 socket. Kindly refer to the following figures for schematics both sockets' pins.

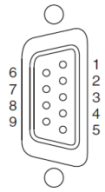
| Connector | Pin | Signal |
|---|---|---|
|  | 1 | No Connection (NC) |
| | 2 | CAN_L |
| | 3 | COM |
| | 4 | NC |
| | 5 | SHLD |
| | 6 | COM |
| | 7 | CAN_H |
| | 8 | NC |
| | 9 | $V_{SUP}$ |



**Figure 43: RS-232 pins schematics [19]**

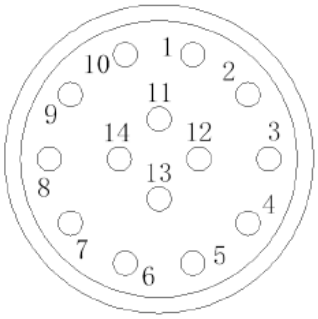



**Figure 44: Kelly J2 pins schematics [20]**

After all the cable at both nodes is already been solider, this cable requires termination resistor of 120 Ohm to prevent communication error, since CAN bus is bidirectional.

This termination resistor need to be placed at the end of CAN_H connection to CAN_L connection at both of the cable. The cable build as of in the following figure.

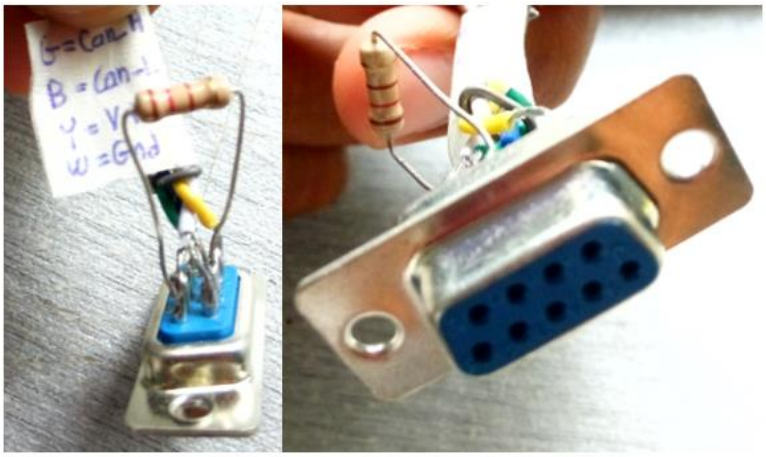


**Figure 45: Example of 120 Ohm cable termination**

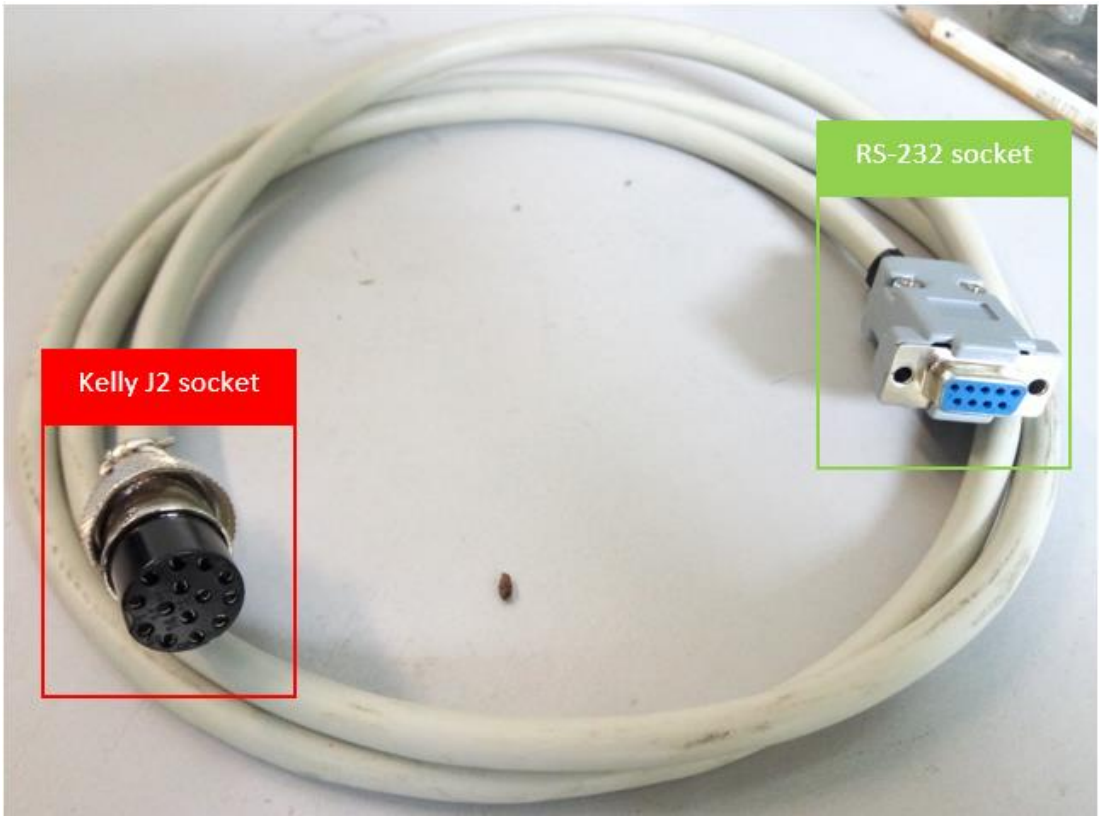



**Figure 46: CAN bus cable**

The CAN bus cable is tested with connectivity function of multimeter and is found that all the connections are corrected. Now, the *CompactRIO* is ready to be connected to Kelly controller and PC.

#### 4.3.2.2 Overall hardware connections

The Kelly Controller, *CompactRIO* and PC are connected as if the following figure.

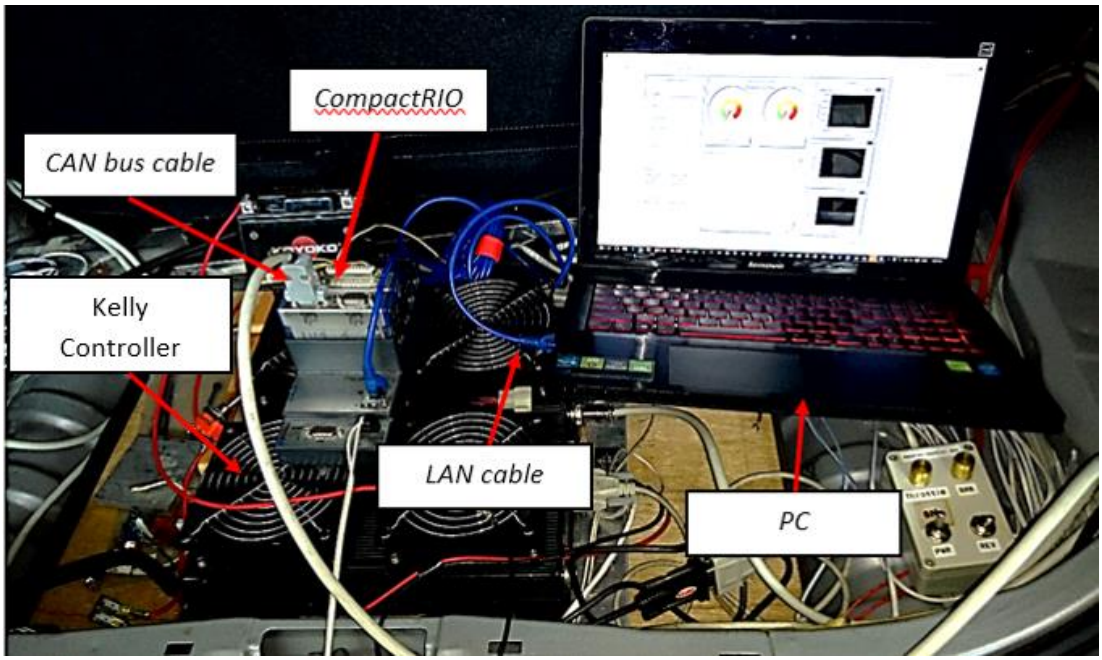



**Figure 47: Overall hardware connection for *CompactRIO*, Kelly Controller and PC**

As of above figure, the *CompactRIO* is connected to Kelly Controller using the CAN bus cable meanwhile, the LAN cable is use to connect the *CompactRIO* and PC. Now, the EMS control program for MCU controller can be executed.

### 4.3.3 EMS control program for MCU

Same as the translation of existing program to FPGA mode, the FPGA programming will only be use to acquire the data from the controller. These acquired data later will be transferred into Real-time program.

#### 4.3.3.1 Data Acquisition in FPGA mode

The input pin of CAN0 of NI 9853 will be use CAN data acquisition for MCU controller. The following FPGA program will be use to acquire data from Kelly Controller.

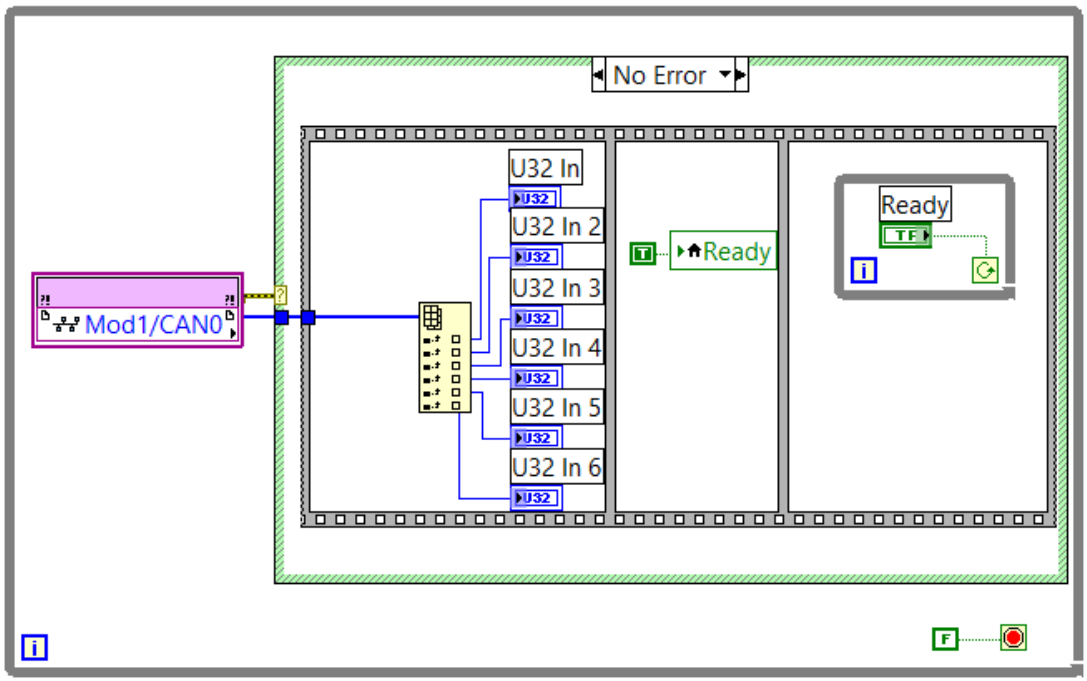



**Figure 48: CAN data acquisition in FPGA programming**

These data will be read in Real-time programming algorithm.

### 4.3.3.2 Data interfacing in Real-time programming

The data in FPGA is read by using the FPGA open pallet (Red box). This function will enable the data in FPGA programming to be read in Real-time programming. These data will be read the Read/write control pallet and output it to front panel (Purple Box). The ID in yellow box, here will enable the user to key in the ID of the CAN bus data in front panel.

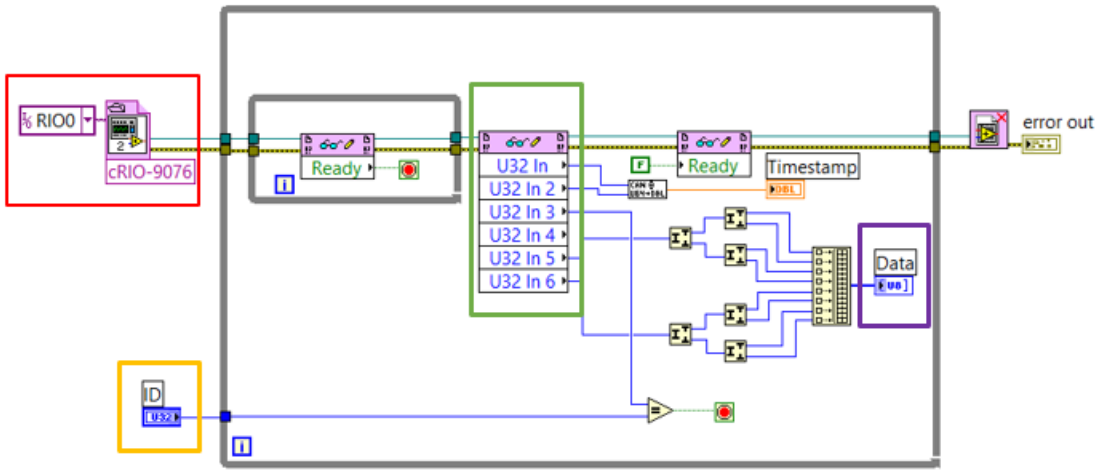



**Figure 49: Data interfacing in front panel in Real-time programming**

The ID of this CAN bus data is given by manufacturer, in the Kelly Motor Controller User's Manual. This ID is in hexadecimal. For this project there are only several parameters will be read by the EMS controller. These parameters are summarized as follow;

| Bil. | Parameters | ID |
|---|---|---|
| 1. | Kelly Motor Phase Current | 26 |
| 2. | Controller temperature | 51 |
| 3. | Kelly Motor Speed RPM | 55 |
| 4. | Current throttle switch status | 66 |
| 5. | Re-generative Braking status | 67 |

### 4.3.3.3 Front panel for end user

The following front panel is the example of whet user will see. Since the program is still in the testing process, the ID in the red box is not being fix yet. Once, the every this is completed the ID will be fix and the data for the related parameters will be display. Meanwhile, the green box will show the error occur if any in the data reading process.

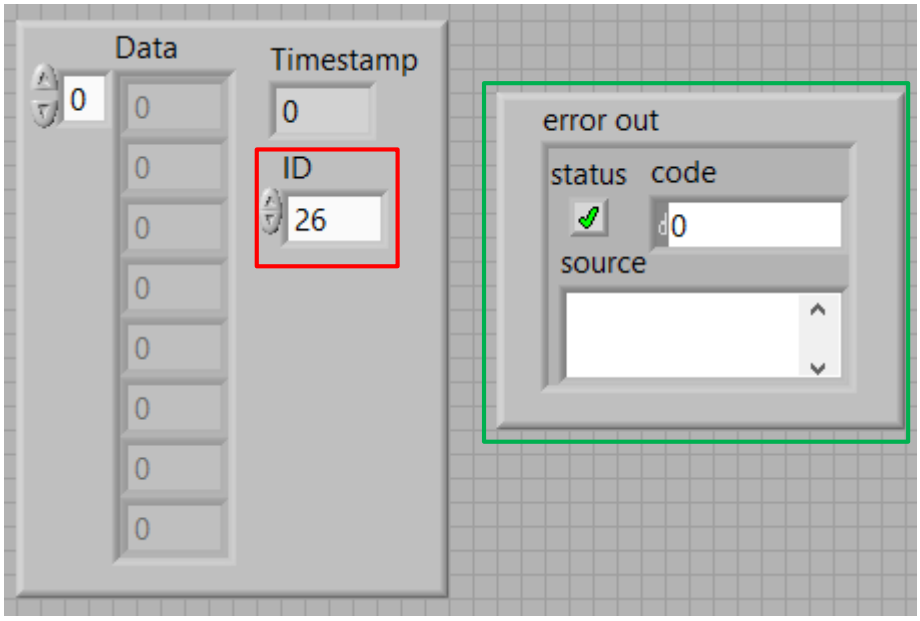



**Figure 50: Front panel for end user**

#### 4.3.3.4 Result

The current program is still being tested. The data of the related ID is still failed to be display. There are several reasons on why, this data is failed to be display.

1. *Due to the clock speed setting of Kelly motor controller*

   The Kelly controller has CAN data rated of about 1Mbit/s. Therefore, the *CompactRIO* reading clock must be set faster than the Kelly Controller clock. For this program, the CAN module NI 9853 is set to run at 1kb/s. Therefore, since the reading clock is faster than the Kelly Controller out data, the data should be display.

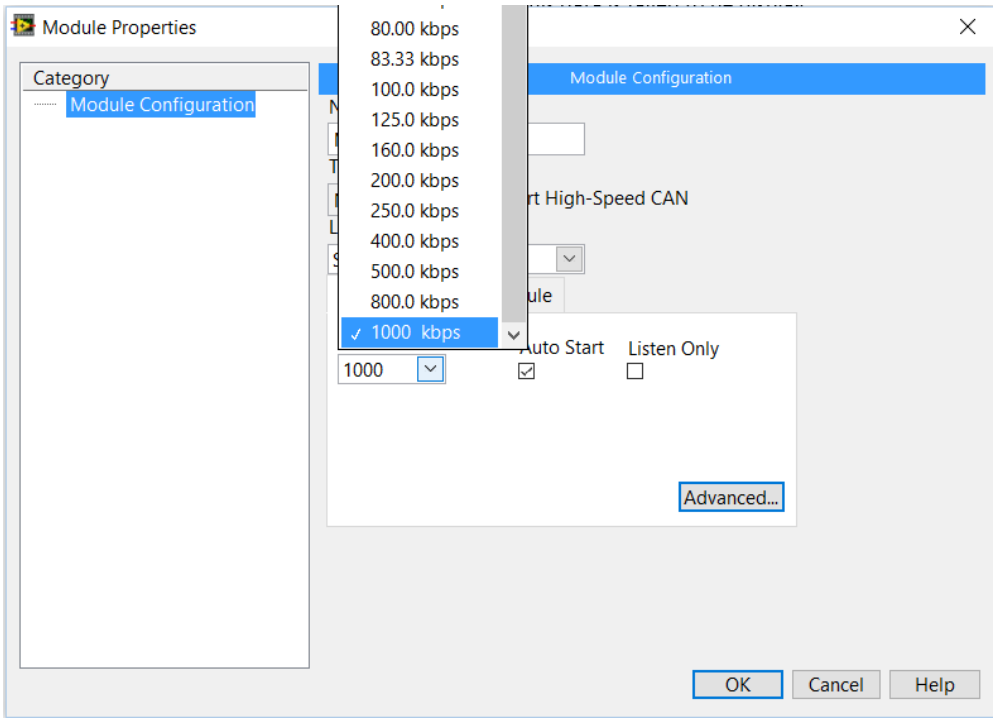



**Figure 51: NI 9853 data rate setting**

   However, there are possibilities that the reading data is set too fast that make the data unable to be display. This possibility takes time to be tested because, since the data acquisition is done in FPGA mode, any changes will result the whole program need to be compile again. This compile time normally will take a long time, for about 15-30 minutes and sometimes the compilation got stuck due to unknown issue. Hence, this possibility remains unsolved.

2. *Due to the faulty of CAN bus cable*

   The CAN bus cable is a self-made cable based on the internet, this cable is only being tested with multimeter connectivity function to see whether the CAN_H or CAN_L between CAN Nodes are connected or not. If it is connected, the multimeter will be beeping. This cable is never being tested to transfer the CAN data. Therefore, this be the one that contributed to this issue.

3. *Due to the wrong data acquisition algorithm implementation*

   The program is made based on the understanding on the certain graphical pallet function, however throughout experience using this software, some of the pallet can only be use for certain hardware only. For example, DAQmx type of program, is made for simple data acquisition which can only be use for certain controllers. National instruments had produced a lot of controllers, such as sbRIO, CompactRIO, PCIe, PXIe, Ni Elvis II and etc. All these controllers can use both Real-time Scan Mode program and FPGA program. However, there are certain function that in both of this program that can only be use for specific controller. For this DAQmx case, sbRIO and CompactRIO are not supported to use this function, but is fully supported for PCIe and PXIe and NI Elvis II controller. Since the author did not receive any training to use this software, there are possibility that this issue comes from this cause.

## 4.4 Graphical Driver Interface (GDI) development

Since, there is some faulty issue in CAN bus data display and also time constraint, this GDI development is unable to be done.

# CHAPTER 5
# CONCLUSION

All data obtained in ems control program in FPGA mode is compared and verified with existing results of real-time scan mode program. The implementation of program conversion has been successful. The completion of ems control program will enable the HEV to achieve certain control strategy for efficient energy distribution and energy storage through re-generative braking, in order to reduce fuel consumption while maintaining acceleration and other performance requirements. This control strategy is a project which is currently ongoing in University of Technology PETRONAS (UTP) entitled *"Design and Development of Split-parallel Through-the-road Retrofit Hybrid Electric Vehicle with In-wheel Motors"*. This FYP project will partially complete this UTP research project.

## REFERENCES


1. Nelson, R. *Keep Tabs on Your Data*. 2013 Available from: http://www4.evaluationengineering.com/articles/201306/keep-tabs-on-your-data.php.
2. Kurniawan, Y., *Development of Energy Management System (EMS) with Driver Interface for Retrofit-Conversion Hybrid Electric Vehicle*. 2013.
3. Zulkifli, S.A., et al. *Implementation of energy management system for a split-parallel hybrid electric vehicle with in-wheel motors*. in *Control Conference (ASCC), 2015 10th Asian*. 2015.
4. Zulkifli, S.A., et al. *Development of a retrofit split-axle parallel hybrid electric vehicle with in-wheel motors*. in *Intelligent and Advanced Systems (ICIAS), 2012 4th International Conference on*. 2012.
5. Zulkifli, S.A., et al. *Operation, power flow, system architecture and control challenges of split-parallel through-the-road hybrid electric vehicle*. in *Control Conference (ASCC), 2015 10th Asian*. 2015.
6. *Web Technology in Embedded Systems*. 2013; Available from: http://www.ni.com/white-paper/14992/en/.
7. *9 Benefits Of Hybrid Cars - 09 - High Resale Value*. Available from: http://www.autobytel.com/hybrid-cars/car-buying-guides/9-benefits-of-hybrid-cars-120071/.
8. *Benefits and Considerations of Electricity as a Vehicle Fuel.* 2015.
9. *Hybrid vehicle drivetrain*. Available from: https://en.wikipedia.org/wiki/Hybrid_vehicle_drivetrain.
10. Taylor, D. *Series vs Parallel vs Series/Parallel Drivetrains*. Available from: http://www.ucsusa.org/clean-vehicles/electric-vehicles/series-vs-parallel-drivetrains#.VjG8-JD6xaR.
11. Hartman, D. *Series Vs. Parallel Hybrid*. Available from: http://www.ehow.com/about_6130613_series-vs_-parallel-hybrid.html.
12. Eberle, U., *The Voltec System: Energy Storage and Electric Propulsion.* 2014.
13. *cRIO-9076*. 2014; Available from: http://sine.ni.com/nips/cds/view/p/lang/en/nid/209758.
14. *How to reduce the time required for programming a PAC*. 2008; Available from: http://machinebuilding.net/ta/t0130.htm.
15. *NI 9401*. 2014; Available from: http://www.ni.com/datasheet/pdf/en/ds-86.
16. *NI 9853*. Available from: http://www.sal.wisc.edu/PFIS/docs/rss-nir/archive/public/Product%20Manuals/ni/ni-9853-datasheet.pdf.
17. *LabVIEW FPGA Course Manual*. 2012.
18. How to Connet Kelly Controller to a Computer. n.d.
19. Operating Instruction NI 9853 (n.d.). Retrieved from http://www.ni.com/pdf/manuals/371453e.pdf

20. Kelly KBL Brushless Motor Controller User’s Manual. n.d.

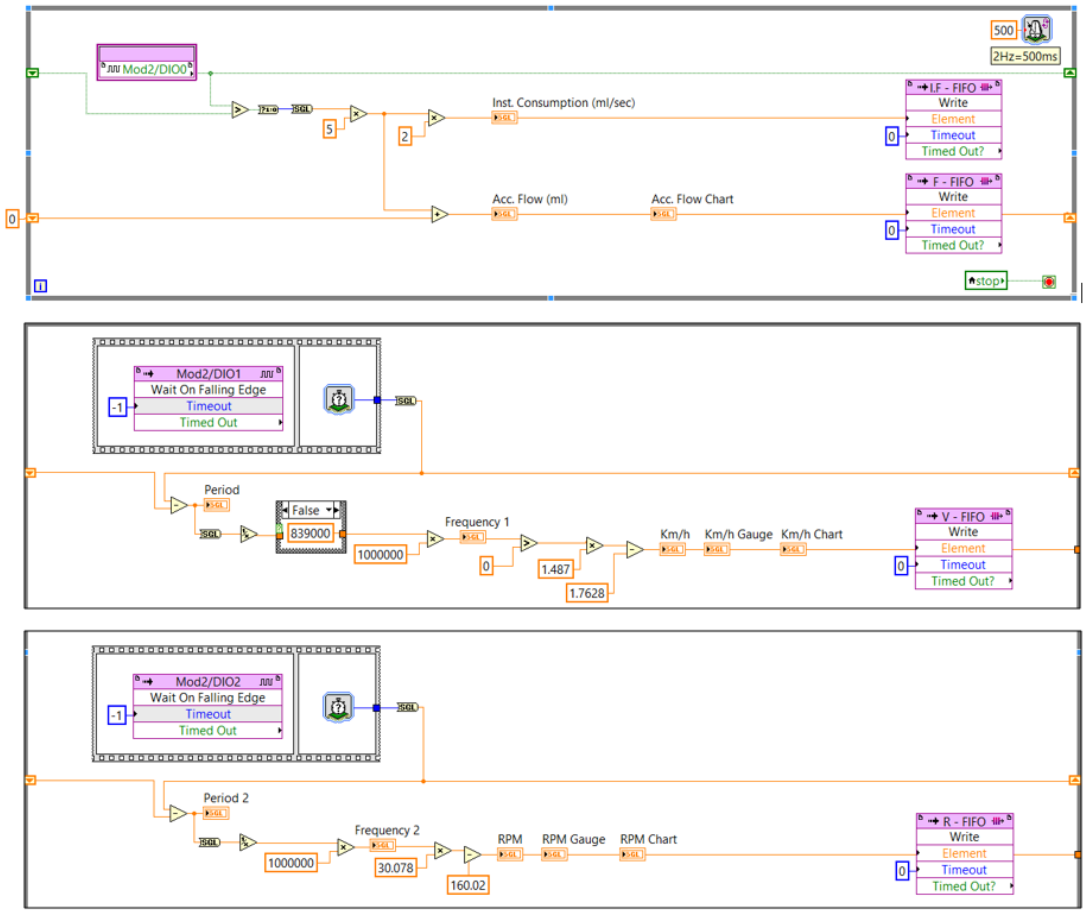
21. Kelly KBLI/KHB/HP Controllers Configuration Program . n.d.

22. A. Bedir, "Design of a Stand-Alone Control Strategy For Retrofit Hybrid Electric Vehicles," Tennessee Technological University, 2010.

# APPENDICES

## APPENDIX A

## LABVIEW FPGA BLOCK DIAGRAM

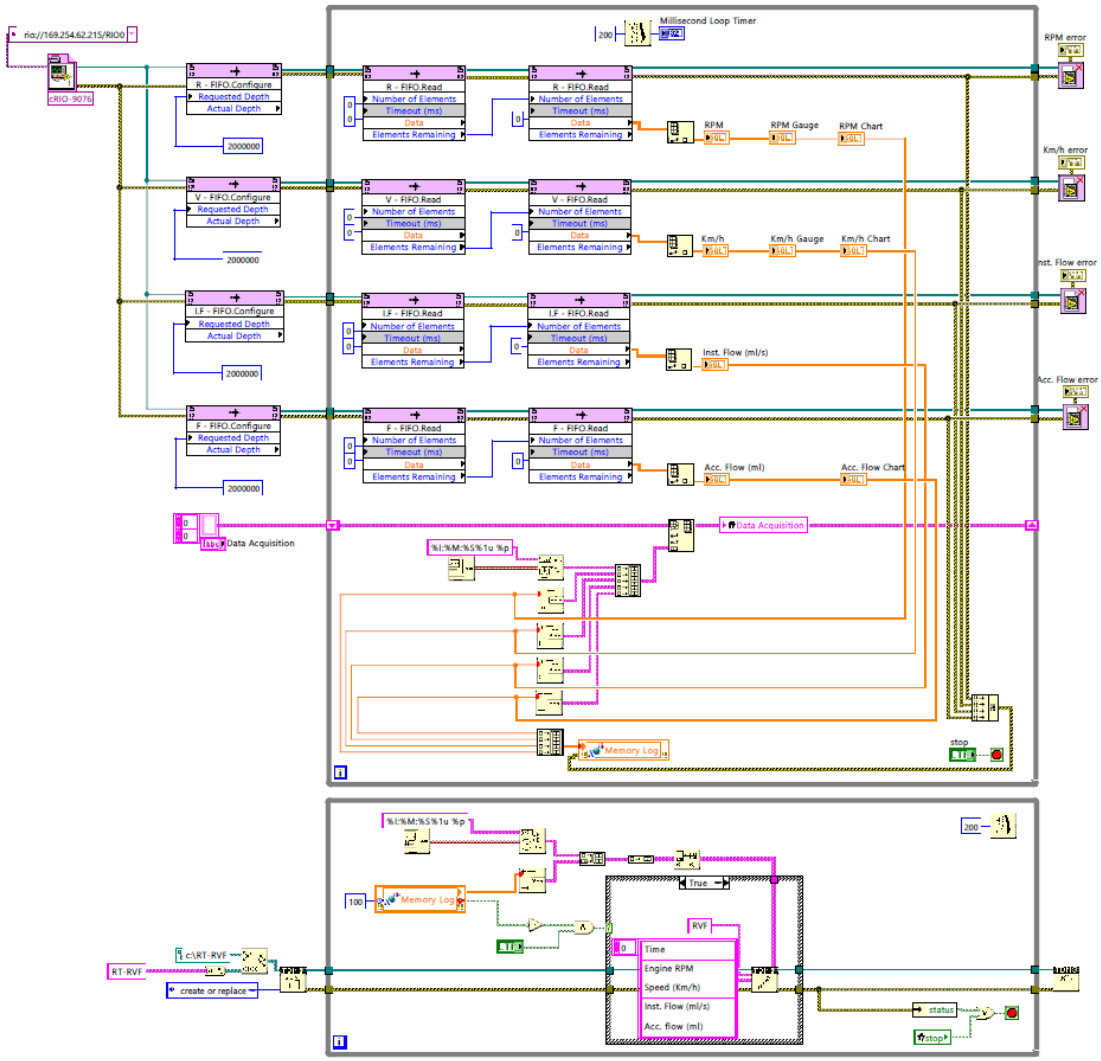


# APPENDIX B

## LABVIEW FPGA FRONT PANEL

# APPENDIX C

# LABVIEW REAL-TIME BLOCK DIAGRAM



# APPENDIX D

# LABVIEW REAL-TIME FRONT PANEL