SEAT - Solar Electronics Automotive Testbench

Mathias Strand*, Henrik Särnblad†, Filip Starkenberg‡
School of Innovation, Design and Engineering, M.Sc.Eng Robotics
Mälardalens University, Västerås, Sweden
Email: *msd16007, †hsd16003, ‡fsg18002@student.mdh.se

Abstract*—*Author: Henrik Särnblad
This report covers the development of a prototype test bench for Mälardalen University Solar Team (MUST) that will be used to test new hardware and software for a Solar Sustained Vehicle (SSV). The report focuses on three key areas: prototype modules, module communication and a Printed Circuit Board (PCB) implementation of the driver user interface.

***Index Terms*—MUST, ssv, test bench**

I. Introduction

Author: Mathias Strand

The electrical and electronic systems that modern cars inhabits are often large complex systems of components, which has to work together in order to be a safe tool to users and everyone around it. These systems often integrate over 100 microcontrollers for the tasks that the car is supposed to perform [1]. From managing air condition to making sure that safety mechanisms such as air bags are deployed at the right time.

A SSV is a vehicle that can receive energy from solar panels and use that energy to propel the vehicle. These vehicles needs energy storage since the energy influx from the sun varies throughout the day. Most often these storage devices are large battery packs and in some cases there are capacitive storage solutions. The electronics of such vehicles should be energy efficient and the components and computing power should not be over dimensioned for their intended purposes. These vehicles also needs to be as small and light as possible for solar racing. This leads to space limitations in the cockpit due to the need of both the aerodynamics performance and small cross sectional area of the solar car.

The scope of this project is the specification and creation of a test bench for the electronic system in a solar sustained vehicle developed by the MDH Solar Team. This system is to be used as a portable testing platform for new features for a future implementation of a SSV. The most common use of these vehicles is for use in competitions around the world, and the goal of the competitions are to advance the knowledge and engineering of sustainable transporting systems. [2]

II. Background

Author: Mathias Strand

This project builds upon a previous project from the summer of 2020, a project that produced a report called ASC2020. The goal of that project was investigating communication capabilities of wired CAN-bus and wireless Zigbee modules as well as suitable sensors and environmental aspects of building a SSV. This version intends to deepen and integrate the previous work for a more reliable testing platform. The previous platform was implemented on a breadboard and unsuitable for moving or transporting. The users also needed to be very familiar with the connections if any component was moved.

A. Hardware present in project

The microcontrollers that was used in this project were mainly STM Nucleo L476RG which is a low power development board that feature standard communication interfaces such as serial peripheral interface (SPI), Inter-Integrated Circuit (I2C), Universal Asynchronous Receiver/Transmitter (UART) as well as Controller Area Network (CAN). It is compatible with Arduino Uno shields, which makes prototyping easier. For the wireless purposes a set of Nrf52840 evaluation boards were used. The Nrf52840 is a Bluetooth Low Energy (BLE) 5.2 device that features long range communication.

B. Visual and audio

The cars need to accommodate road regulations of each race they are intended to be used in, however visual and audio signals of the car should consume as little power as possible. The lighting and sound equipment was acquired from the 2019 iteration of the MDH solar car. It consisted of two front light modules with Daylight Running Lamp (DRL) and turn indicators, two indicators for use on the side of the car, two rear lights with integrated turn indicators and finally a center tail and brake light. The audio module consisted of a regular car horn.

C. Motor and motor controller

A 1kW Brushless Direct Current Permanent Magnet (BLDC) motor was used in this project, it connects to a motor controller designed specifically for it by the same company that designed the motor. It is controlled with three switches and two potentiometers which regulates two 5V signals. The switches are for turning the controller on and off, setting the power consumption mode and controlling the acceleration direction. The potentiometers control the amount of acceleration and regenerative braking. [3]

D. CAN

CAN was used in this project as a robust communications method between Micro Controller Units (MCUs). In depth information about about CAN is available in the report background of ASC Summer project 2020.

III. Method

Author: Mathias Strand

A set of requirements was established in the beginning of the project to define what was needed from a working car. The method inspiring this choice of working was the V-model [4] that is common in systems engineering. By treating the car as a black box and defining the subsystem that will be contained in the car. This was done in order to define the overall requirements of the project. The functional requirements can be viewed in the appendix 4. The requirements themselves are derived from what is valuable to the stakeholders of the project, and serves as a tool for defining the end product functionality and tests to ensure that the implementation is performing as initially intended. This approach combined with an iterative process was used to get the final results.

A. Prototype

Author: Henrik Särnblad

From the previous mentioned requirements a test bench layout was derived which can be seen in fig 6. The prototype design of the test bench consists of six major modules connected through CAN or BLE where each module serves a specific purpose. The modules are the *Battery communication*, *Sensors*, *Dashboard*, *Propulsion*, *SSV communication* and *Follow car communication*. Out of these six modules, four are currently implemented as prototypes on the test bench with the goal of being expanded to closer correspond to the final SSV electrical system in the future. The missing ones are the *Battery communication* and *Propulsion*.

Each module serves different purpose which are as follows:

1) Sensors: is responsible for collecting all sensor values throughout the SSV and send these on the CAN bus. This module is a placeholder that will be replaced with more specific microcontrollers in the future.

2) Dashboard: also known as user interface, is responsible for taking input from the driver such as throttle and indicators and send these on the CAN bus but also read relevant values from the bus and display them to the driver.

3) Battery communication: monitors the battery status, such as voltage and warnings, and send this data on the CAN bus for further processing and logging.

4) Propulsion: which purpose is to manage the cruise control and monitor speed and faults, such as overheating, and then send these values on the CAN bus.

5) SSV communication: is responsible for logging all relevant data from the CAN bus and send this data to the follow car via BLE.

6) Follow car communication: has a similar purpose and is responsible for logging all data and send it to the energy domain for further processing.

B. CAN-bus

Author: Henrik Särnblad

The communication between all of the microcontrollers within the car is done through CAN-bus which is a robust communication bus commonly used in automobiles. The Nucleo microprocessors that are used has a built in CAN controller which in turn are connected to a CAN transceiver. All of the CAN transceivers are connected to each other to make up the network.

C. User Interface

Author: Mathias Strand

A design of a test bench interface was derived from the functional requirements that were set up during the early process of the project. These requirements provided an overhead view of the necessary inputs and outputs from the drivers perspective. The requirements was used to determine the amount of buttons that was needed, the amount of connectors to external equipment and the number of status Light Emitting Diodes (LEDs) to indicate hardware functions. The initial design concept can be seen in 1.

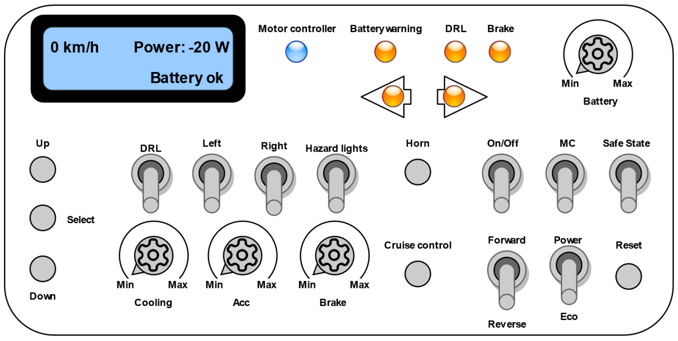


Figure 1. The interface design has inputs for a microcontroller and a MC. The outputs consists of a LCD to show detailed information. The LEDs indicate hardware functionality that needs lower level of detail while still being important and freeing up space from the LCD.

D. Software

Author: Filip Starkenberg

The software is written in C programming language using STM32 Cube IDE and Visual Studio Code.

The software is using Free Real Time Operating System (FreeRTOS) which is a real time operating system for embedded systems.

Currently there are two Nucleos running separate software, Interface and Sensor reading.

The sensor reading Nucleo is reading thee temperature sensors and broadcasting the data on the CAN bus.

The User Interface Nucleo handles buttons, indicators, Receives CAN messages, reads local temperature and displays data on the Liquid Crystal Display (LCD).

A menu system was developed for the LCD where the User can select what data to be displayed on the LCD using three buttons on the User Interface Panel.

Since we do not know yet exactly what features is required on the finished product, the Software is coded in a modular way for ease of modification and expansion.

E. Modular design

Author: Filip Starkenberg

The old base was a single surface where all the components was mounted directly to using zip-ties and screws which was not so easy to work with.

A new base for the prototype test bench was designed and manufactured to be modular and more expandable.

In the new design there are three types of modules; NucleoL476RG, LCD and breadboard module. Each module consists of a 3d printed base that the module is mounted on and the module it self. The base of the module is a rectangular surface with holes around the edges where the modules can connect to each other using a plastic connector.

Making a new module is very easy since all modules are using the same standard dimensions for the connector holes and the size in both axis are a multiple of 20 mm.

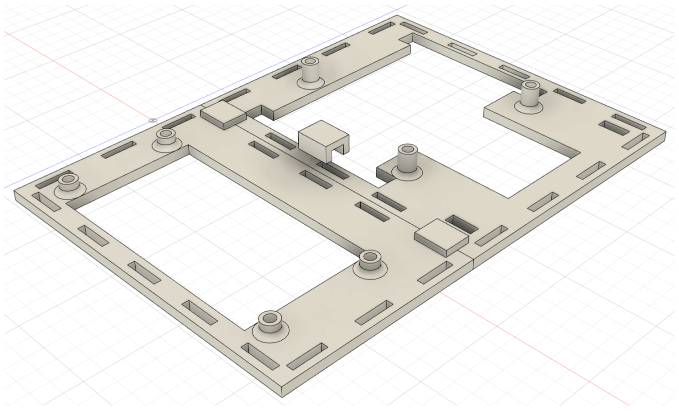


Figure 2. This picture shows an example configuration of the modular system consisting of a NucleoL476RG module attached to a LCD module. (The electronics are not shown in the picture.)

F. PCB implementation

Author: Mathias Strand

The purpose of the PCB implementation was to provide a easy to use and transport platform version of the user controller interface. The PCB in itself will enable easier software development and testing of the future solar car system, using modular blocks, new functionality can be connected and integrated via the CAN network.

The PCB functions are similar to dashboard controls and in the future implementation of the solar car it can be adapted and redesigned to serve as the steering wheel. The input devices are connected to the MCU and are used to control some aspect of the system. The output devices are controlled by the MCU and responds to the physical inputs that the MCU can receive via the input devices or via the CAN-bus.

As for the components, they were separated into analog and digital groups. The digital components were mostly grouped to the left and upper side of the PCB and the analog devices were placed as far to the right corner as possible.

Considerations were taken to accommodate the current consumption of visual and audible peripherals. The lighting and audio peripheral power budget was roughly estimated to 15 watts continuous use. This number was derived from the products used at the last Bridgestone World Solar Challenge (BWSC) race, and for safety this number was doubled when setting the trace width of the high current traces of the PCB. The trace width of the power supply traces was set to 1mm, if the calculated peak current of $2.5A$ is reached this will result in $10°C$ a temperature rise of the traces. The IPC-2221 was used to calculate the trace widths.

$$Area[mils^2] = \left(\frac{(Current[Amps]}{(k * (Temp\_Rise[°C])^b))}\right)^{(1/c)} \tag{1}$$

$$Width[mils] = \frac{Area[mils^2]}{(Thickness[oz] * 1.378[mils/oz])} \tag{2}$$

IPC-2221 external layers: k = 0.048, b = 0.44, c = 0.725 was used for all traces. k, b, and c are constants result of curve fitting to the IPC-2221 curves. [5]

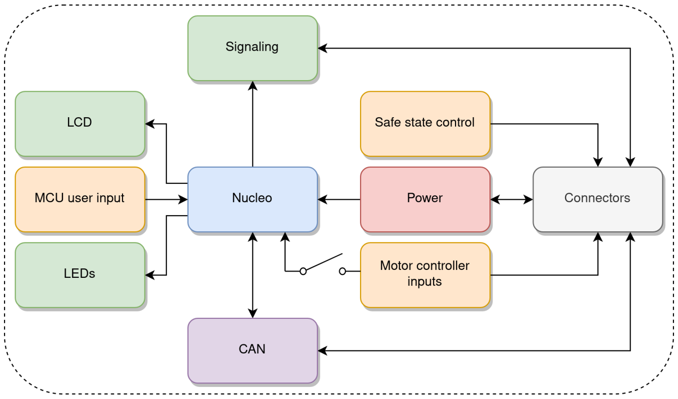


Figure 3. This picture shows the different hardware present on the PCB and the connections and information direction flow between each part. The Nucleo microcontroller is connected to commonly found inputs and outputs on a car dashboard as well as communication(CAN), power and connectors for external hardware used to control other sub systems in the car. Only one microcontroller were present on the PCB. If a network is to be set up then more microcontrollers can be connected via the CAN-bus connector.

1) Motor controls: The motor control IO circuit has two modes of operation. The first one is a mode where the MC does not interface with the microcontroller at all, and the PCB provides the user with a clean and convenient setup for testing the functionality of the motor and MC. In this mode the MC is connected to the board and a series of Dual In-line Package (DIP) switches on the backside of the board is turned to the off position. The purpose is enabling an easy way to connect the MC directly to the hardware inputs it needs to be run the motor. The inputs the MC needs are: 2 potentiometers for the regenerative braking and acceleration as well as a direction switch, on/off switch and a power mode selection switch.

The other mode is for taking user inputs to the microcontroller and sending the data on the CAN-bus to a microcontroller that can output the reference signals which the MC needs. This is a solution that is much more flexible than the previous one, since the input devices for acceleration and braking can be of many different types as long as the input microcontroller and the output microcontroller can read and output the signals of the device. This way of delivering information to the MC through a microcontroller also enables a cruise control implementation that is not present in the motor controller itself but one that is severely needed in a car that is to participate in any long distance race.

2) PCB cover: A front cover was derived from an exported 3d model of the PCB. The model was imported into Solid Works and using multi-body design, the material of the PCB and the components was subtracted from a block of material to indicate where the hardware was positioned. The block was then shelled out and screw holes was added for fixation of the PCB to the cover. The production method used was 3d printing.

G. Sensors

Author: Henrik Särnblad

As the final layout of the SSV is not known, a microcontroller is added with the purpose of collecting and emulating sensor data that is not from within the cockpit. The plan with this microcontroller is to add different sensors that does not yet belong to a specific microcontroller and then later split it and assign it to the correct microcontroller as the SSV is developed. This microcontroller is a Nucleo-L476RG and is connected to the system through CAN bus.

The microcontroller can collect data in multiple ways such as I2C, 12-bit analog-to-digital converter (ADC) and SPI. There are currently no sensors connected through I2C but there are sensors connected through the ADCs and SPI. In the current stage of development there exists untested code to connect a potentiometer and the internal CPU temperature via the ADC to the sensor module. The SPI has three temperature probes connected that can be used to measure ambient temperature in for example in the electronic boxes.

Each sensor is assigned a unique global ID that is known within the entire car. This ID is used to distinguish between different sensors and set their priority on the CAN bus. When the microcontroller has a new sensor value it will send it to the CAN controller which in turn will attempt to send it on the CAN bus when a slot is available.

IV. Results

This section contains the result of the two prototypes.

A. Prototype

Author: Henrik Särnblad

The modules of the test bench does perform mostly as expected during individual testing but communication issues exists with the CAN bus causing data to not be received properly. Oscilloscope confirms that there are data being transmitted on the bus.

During testing some other issues are also observed and are as follows:

- LCD does not always initialise properly and sometimes displays random characters. (Requires reboot of microcontroller)
- Micro controllers hang if the CAN controller queue is full or the can transceiver is disconnected.

B. PCB

Author: Mathias Strand

The produced PCBs hardware functionality was confirmed to be working by conducting tests of the individual parts. The board could receive power both through the Nucleo micro controller as well as the external connector with no perceived differences. The LCD could display messages, the status LEDs could light up and be turned off, the button presses to the microcontroller were registered.

The 3d printed front cover was deemed suitable after 5 iterations of prints with redesign of text and screw holes.

The were faulty implementations and inconvenient ones. The faulty implementations were:

- One connector was of the wrong gender and didn't match the motor controller.
- The backlight of LCDs did not light up.

The inconvenient were:

- The silk screen was missing information about connector polarity and name as well as a name and revision for the PCB.
- Varying light intensities of the LEDs due to mismatched resistor values.
- The connector to the Nucleo micro controller was placed at a bad position, directed inwards to the PCB, making it a bit hard to get access to without flipping it over.

V. Discussion

A. Prototype

Author: Henrik Särnblad

The modules of the test bench are mostly finished but due to limited number gatherings the modules are not fully integrated with each other. The modules has been individually tested and show promising results for their intended purposes but limited testing has been performed on the CAN bus connection and its stability.

In the current stage of development the modules are connected using breadboard connectors which might pop out and are hard to get a overview of. A better solution for this would be to chain all modules using two standard connectors on each module to pass through the CAN bus. This would also make it easier to add new modules to the system.

B. PCB

Author: Mathias Strand

Overall the implementation was a success since the final PCB demonstrated that the schematics design and layout were overall functional. The shortcomings were a few features that was only partially working or needed processing post manufacturing or the components needed to be swapped.

The mismatched connector belonged to the motor controller and the hardware functionality could be tested and confirmed on a separate PCB where a connector of the right type was soldered to the opposite side of the PCB, since the pin mapping of that component was mirrored compared to the initial design.

There was limited work on implementing the software from the breadboard test bench onto the PCB version, instead previous code from the ASC summer project was used and slightly altered to verify the hardware functions.

VI. Conclusion

A. PCB

Author: Mathias Strand

The purpose of the implementation was to create a sturdy, compact and mobile platform with a modular design, with CAN connection for receiving information from other embedded systems and easier testing of software and hardware functionalities for the future solar car system. A common problem with previous implementations was that they either were bulky or that they were fragile and transport could easily damage them, they didn't have connectors for additional hardware or the connections were only through a breadboard and not very sturdy. The previous platforms also were also hard to get insight into how they operated and because they were implemented on breadboards, they were difficult to maintain and hand over to new users. Turning previous iterations into a compact format reduces this threshold significantly.

VII. Future work

There are a couple of aspects of the test bench that can be improved and functionality that needs to be implemented for it to reach the initial goals. A final goal does not exist for the test bench as the purpose is that it should constantly be developed and improved as the SSV is being developed.

A. Prototype

Author: Henrik Särnblad

For the test bench to fulfill the initial goals the following actions has to be done.

- Connect the battery through CAN bus and collect their values in the user interface and data logging unit.
- Develop the propulsion section with speed readout, warnings and cruise control.
- Integrate the communication section to the rest of the test bench.
- Integrate the PCB implementation of the user interface to the rest of the test bench.
- When the SSV is costructed the sensor module can be split in to parts that correspond to the electrical layout.
- Add two CAN connectors to each board with standardised connectors to easily connect new modules to the test bench and pass through the CAN bus. Terminate the endpoints using a connector with a resistor bridge.

B. PCB implementation

Author: Mathias Strand

Upon testing the factory produced PCB, these hardware faults were discovered that needs to be fixed in the next iteration of the test bench.

- Change D-sub 37 connector gender.
- Change the MOSFET to a signal transistor in the LCD design.
- Adjust the resistors to match the LEDs
- Route the CAN traces as a differential pair.
- Add silk screen descriptions. It should include Name of PCB, name and polarity for each connector.

The PCB can also be made smaller by reducing space between buttons and reiterating the original design and placement of the other components. The Nucleo should also be rotated so that the programming connector faces to the edge of the test bench and not inwards to enable easier access to the programming port. Additionally these tasks should be pursued to continue the modular system development.

- Standardise use of specific connectors and cables for CAN. E.g ethernet.
- Design PCB module that collects sensor data and communicates via CAN.
- Design PCB module for BLE communication, data storage and CAN.
- Design CAN connections to enable linking multiple modules.

C. Testing

Author: Henrik Särnblad

To confirm functionality following tests should be performed.

- Maximum load on CAN bus before missing deadlines.
- Maximum load on local and remote data storage before it is saturated.
- Maximum load on BLE before the connection is saturated.
- Behaviour for disconnected CAN bus cables.
- Consequences from a sudden power loss. (Will data logging be corrupted?)

Acknowledgment

We thank Emil Persson (Student MDH) for his great contribution as a project manager / team leader and helping to keep the motivation up. We thank Martin Ekström (Program coordinator / Senior lecturer MDH), Henrik Falk (Reaserch engineer MDH) and Lennie Carlèn Eriksson (Assistant professor MDH) for their contribution of knowledge within the field of electronics. We would also like to say a big thanks to our sponsors from Volvo providing significant amount of money, Würth for knowledge and components. And last we would like to give a big thanks to Mikael Ekström (Associate professor MDH) for sharing knowledge and enabling this project.

References

[1] R. Bannatyne, "Microcontrollers for the automobile," accessed: 25-04-2021. [Online]. Available: https://www.mcjournal.com/articles/arc105/arc105.htm

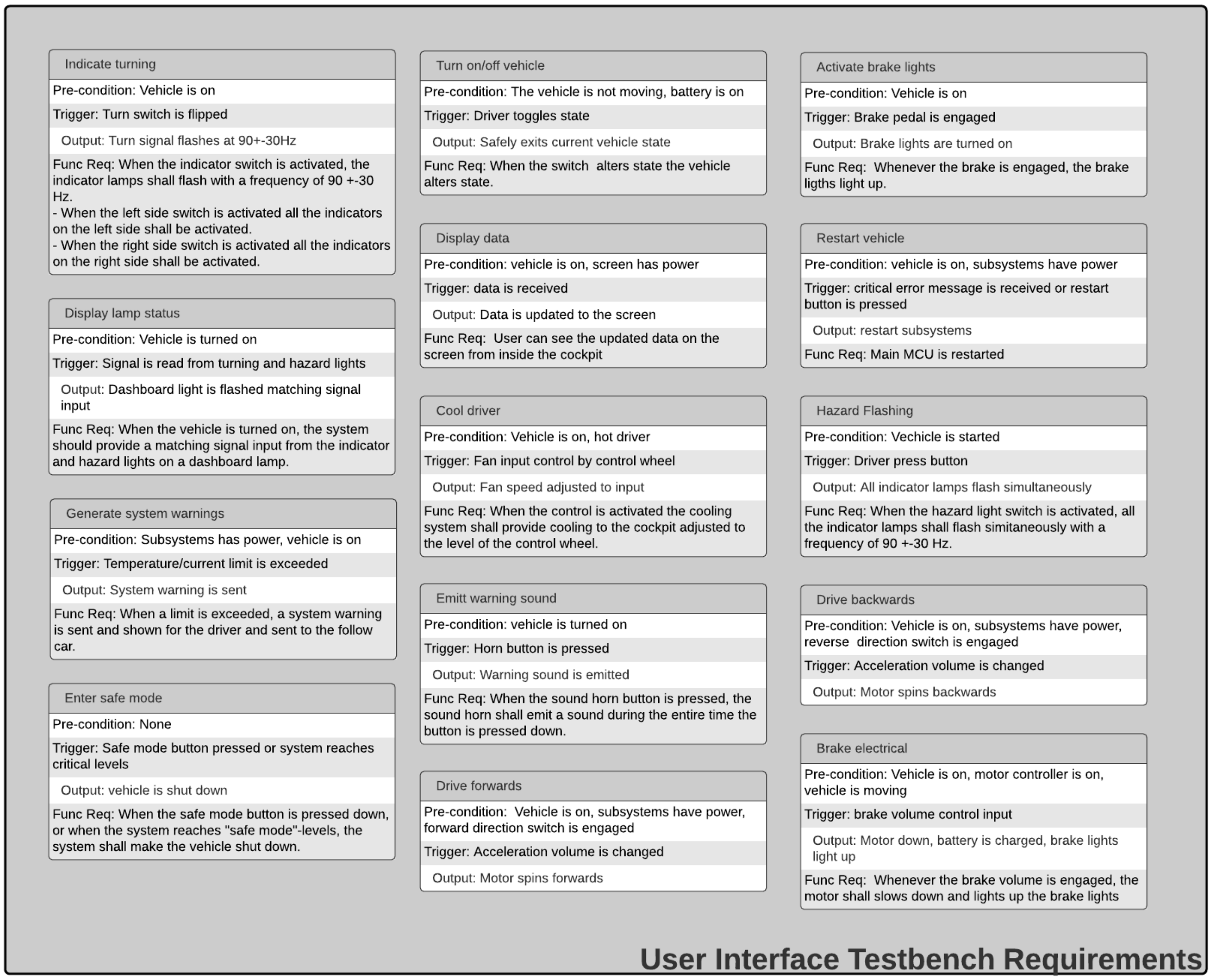
[2] Bridgestone, "Bridgestone world solar challange," accessed: 25-03-2021. [Online]. Available: https://www.worldsolarchallenge.org/

[3] Mitsuba, "Driving motor products," accessed: 25-03-2021. [Online]. Available: https://www.mitsuba.co.jp/scr/products/m0548-%E2%85%A2%EF%BC%8Fm1096-%E2%85%A2

[4] TryQa, accessed 19-04-2021. [Online]. Available: http://tryqa.com/what-is-v-model-advantages-disadvantages-and-when-to-use-it/

[5] "Ipc-2221a," accessed: 24-03-2021. [Online]. Available: https://www.ipc.org/TOC/IPC-2221A.pdf

VIII. Appendix



Indicate turning
Pre-condition: Vehicle is on
Trigger: Turn switch is flipped
Output: Turn signal flashes at 90+-30Hz
Func Req: When the indicator switch is activated, the indicator lamps shall flash with a frequency of 90 +-30 Hz.
- When the left side switch is activated all the indicators on the left side shall be activated.
- When the right side switch is activated all the indicators on the right side shall be activated.

Display lamp status
Pre-condition: Vehicle is turned on
Trigger: Signal is read from turning and hazard lights
Output: Dashboard light is flashed matching signal input
Func Req: When the vehicle is turned on, the system should provide a matching signal input from the indicator and hazard lights on a dashboard lamp.

Generate system warnings
Pre-condition: Subsystems has power, vehicle is on
Trigger: Temperature/current limit is exceeded
Output: System warning is sent
Func Req: When a limit is exceeded, a system warning is sent and shown for the driver and sent to the follow car.

Enter safe mode
Pre-condition: None
Trigger: Safe mode button pressed or system reaches critical levels
Output: vehicle is shut down
Func Req: When the safe mode button is pressed down, or when the system reaches "safe mode"-levels, the system shall make the vehicle shut down.

Turn on/off vehicle
Pre-condition: The vehicle is not moving, battery is on
Trigger: Driver toggles state
Output: Safely exits current vehicle state
Func Req: When the switch alters state the vehicle alters state.

Display data
Pre-condition: vehicle is on, screen has power
Trigger: data is received
Output: Data is updated to the screen
Func Req: User can see the updated data on the screen from inside the cockpit

Cool driver
Pre-condition: Vehicle is on, hot driver
Trigger: Fan input control by control wheel
Output: Fan speed adjusted to input
Func Req: When the control is activated the cooling system shall provide cooling to the cockpit adjusted to the level of the control wheel.

Emitt warning sound
Pre-condition: vehicle is turned on
Trigger: Horn button is pressed
Output: Warning sound is emitted
Func Req: When the sound horn button is pressed, the sound horn shall emit a sound during the entire time the button is pressed down.

Drive forwards
Pre-condition: Vehicle is on, subsystems have power, forward direction switch is engaged
Trigger: Acceleration volume is changed
Output: Motor spins forwards

Activate brake lights
Pre-condition: Vehicle is on
Trigger: Brake pedal is engaged
Output: Brake lights are turned on
Func Req: Whenever the brake is engaged, the brake ligths light up.

Restart vehicle
Pre-condition: vehicle is on, subsystems have power
Trigger: critical error message is received or restart button is pressed
Output: restart subsystems
Func Req: Main MCU is restarted

Hazard Flashing
Pre-condition: Vechicle is started
Trigger: Driver press button
Output: All indicator lamps flash simultaneously
Func Req: When the hazard light switch is activated, all the indicator lamps shall flash simitaneously with a frequency of 90 +-30 Hz.

Drive backwards
Pre-condition: Vehicle is on, subsystems have power, reverse direction switch is engaged
Trigger: Acceleration volume is changed
Output: Motor spins backwards

Brake electrical
Pre-condition: Vehicle is on, motor controller is on, vehicle is moving
Trigger: brake volume control input
Output: Motor down, battery is charged, brake lights light up
Func Req: Whenever the brake volume is engaged, the motor shall slows down and lights up the brake lights

User Interface Testbench Requirements

Figure 4. The Functional requirements of what the testbench hardware shall support. This was derived from through stakeholder requirements and the systems engineerings V-model. Each block represents a subsystem and inside is contained what is needed to trigger the subsystem function and the subsystem output.

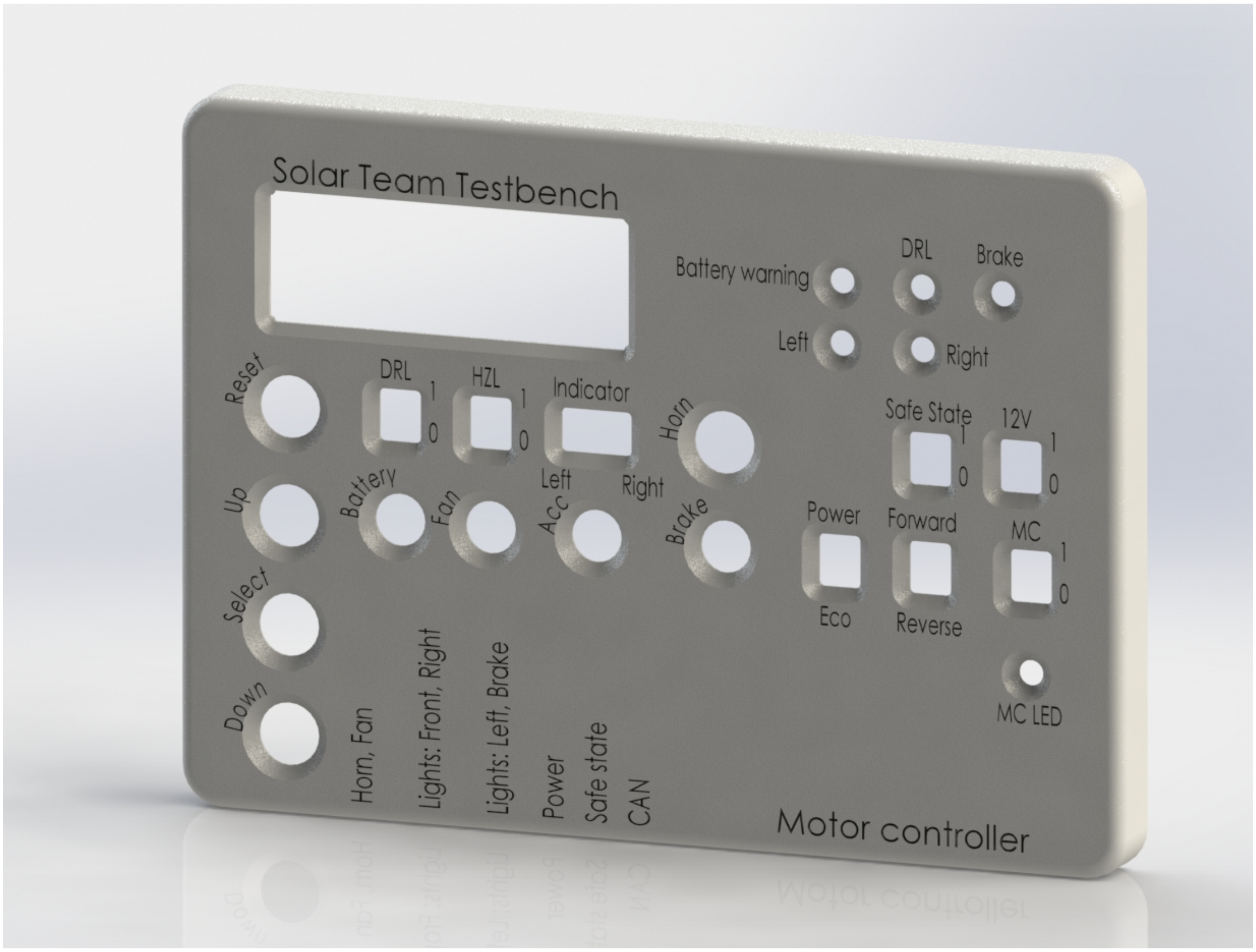


Figure 5. Final render of the PCB testbench cover.

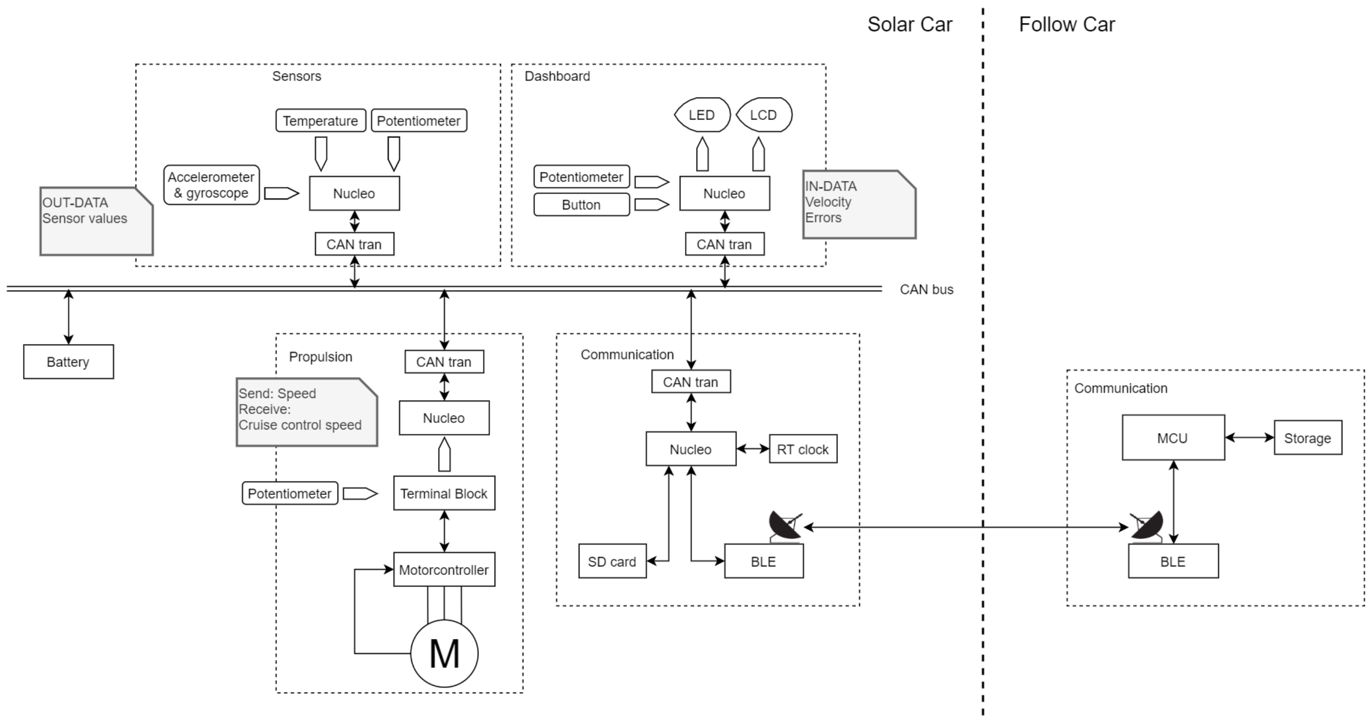


Figure 6. Overview of the prototype divided into modules.

Background from the report of the ASC summer project which covered CAN-bus thoroughly.

2. Background

2.1. Environmental and communication analyses

Author: Johan Bergelin

The relevant background can be found in the corresponding appendixes at the end of the report.

2.2. CAN Protocol

Author: Mathias Strand

Controller Area Network (CAN) is a robust, half duplex, differential signaling system, serial bus standard for communication between various embedded systems called Electronic Controller Unit (ECU).[2] The development of CAN started in 1983 at Robert Bosch GmbH [3] and was officially released in 1986 at Society of Automotive Engineers (SAE).

The goal was to develop a fast and robust inter-embedded systems communication. The ECU's is referred to as nodes in the CAN network and can be various Progammable Logic Device (PLD)'s or Micro Controller Unit (MCU)'s. [4] Each node is connected in parallel to each other through a two-wire bus and the end of the network is denoted with a 120 Ω resistor at both sides. The two wires are referred to as CAN high and CAN Low and are often twisted together or printed on a Printed Circuit Board (PCB). The differential nature of the communication makes CAN more robust against inductive spikes, electrical fields and other noise. [2]

The protocol is a message based and priority oriented to be almost completely collision-free. It is multi-master and no single node has total control over the network, though this demands that all nodes have to listen to the network even as themselves try to send messages. The collision-free environment is achieved through dominant/recessive bits and ID filtering.

A Controller Area Network can be comprised of two or more nodes. The maximum number of nodes is constricted by delay time and the electrical load of the bus. The total bus length for CAN varies with the signalling speed. Lower speeds allow longer bus length. This is because of propagation delays that the medium exhibits when sending signals over a long distance. It is easy to forget that nothing happens instantaneously. As well as higher transfer rates are more sensitive to environmental noise and possible reflections. The abstraction layers that the CAN protocol comprises are Physical layer, Data-Link layer, and Application layer, where the Physical layer is the two-wire multi-master bus.

Total bus length (m)	Signaling speed (Kbit/s)
40	1000
100	500
200	250
500	100
1000	50

2.2..1 Can Frames

The Data-Link abstraction layer handles the work of getting the messages away safely and ensuring that errors are contained. The standard CAN frame from pre 2015 [5] consists of 44 bits or 4,5 bytes excluding the actual data which can range from 0 to 64 bits(0 to 8 bytes). The extended frame consists of 64 bits excluding the data field. The biggest difference is the Identifier field that is extended with the second part of 18 bits. The standard frame enables 2048 different message IDs while the extended frame enables 536 870 912. [6]

2.2..2 ID handling

The Data-Link layer at last deals with message filtering. The nodes that are trying to communicate start with transmitting their message ID. 0's act as the dominant bits and 1's as recessive. [5] When a node notices a lower ID than what their message has, the node ceases to transmit and only listens. In short terms it is a struggle for the lowest numbered ID, a lower ID means higher priority.

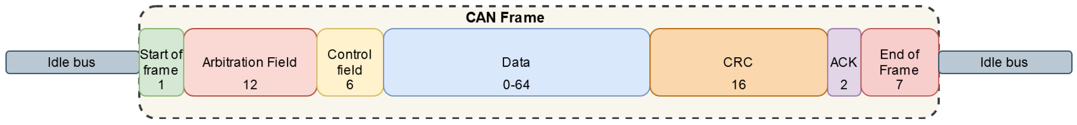


Figure 1: The start of the frame is signalled by a dominant bit. Then the message identifier follows that is part of the Arbitration field and is concluded with a Remote Transmission Request(RTR). The RTR decides if the frame is of type "Remote" or "Data". "Remote" is a data frame request, while "Data" is an ordinary message frame. The "Remote" frame contains no actual message. The control field consists of an Identifier extension bit, which indicates if the message has the standard 11-bit identifier or the extended 29 bit. The reserved bit is always set to dominant but accepted as either dominant or recessive. The "data length code" tells how big the message is. After that comes the actual message and then follows a Cyclic Redundancy Check (CRC). Then the message Acknowledgement (ACK) follows, which always is recessive, and any node can assert a dominant bit to get the sender to resend the frame. Finally is the End of Frame (EoF) which is 7 recessive bits. The type of message in the CAN network can have both the standard ID and the extended ID. There are also additional types of frames such as the "Error" and "Overload" frame.

2.2..3 Error detection

There several redundant error checking mechanisms of the CAN network. In total 5, 2 of which is at the bit level and 3 at the message level. At the bit level, there is the "data bit" checking mechanism. Where the messenger is fact-checking that the bit is written is the same as the bit read, except for the message identifier. The other one is "bit stuffing". Bit stuffing is executed when five of the same consecutive logic level bits is read. A complement bit is then inserted on the bus. The bit stuffing enables the network to synchronize the nodes.

At the Data-Link level, there is a CRC which checks that the message contains the right information through a 16-bit checksum generated from the data field. The ACK consists of two recessive bits. One is the actual ACK and the other delimits the ACK. Finally, there is the form check. The form check checks that the pre-decided recessive bits of the frame are always present. If a dominant bit is in the wrong place then an error is generated. The recessive bits that always must be in a frame is Start of Frame (SoF), EoF, ACK delimiter and the CRC delimiter bits. [5] A frame is considered valid if all checks have been passed and the EoF is confirmed.

2.2..4 Message handling

The application layer can decide if a message is interesting or not. It is possible to only "subscribe" to certain message priority levels and thus making sure that only the relevant information is received. [7] If any fault is detected in lower layers then error messages are generated to get the other nodes to resend their frames.

2.2..5 Bit Timing

Since the communication is asynchronous the recipients have to know when to sample the line. The communications smallest building blocks are called Time Quantum. This unit is decided by the clock frequency and the Baud Rate Prescaler.

$$TQ = 2 * BRP * T_{osc} = \frac{2 * BRP}{F_{osc}} \tag{1}$$

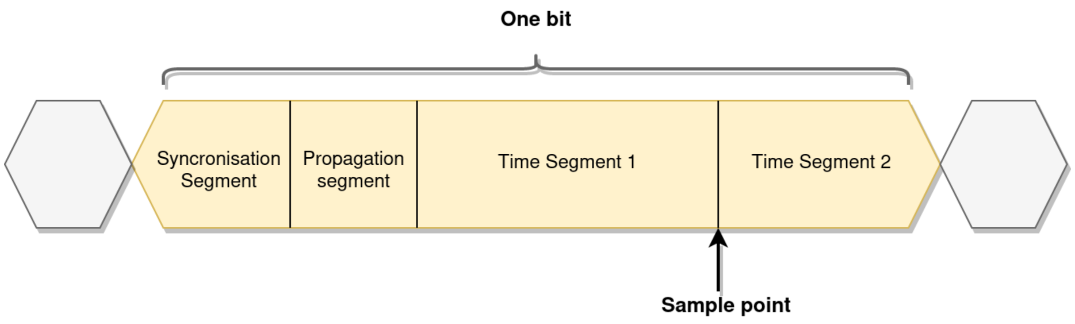


Figure 2: Each bit is comprised of four parts. which are a sync part, a propagation part and two time segments. The sync part is used for nodes to synchronize and is fixed to 1-time quanta. [8] The propagation part is compensation for the delay on the bus and can be programmatically set. [8] The length of the time segments is also up to the programmer to decide. The point of sampling lies between the two-time segments.

The CAN timing settings is stored in the Bit Timing Register (BTR). Each node has to agree on the timing format of messages as opposed to the different message formats. Faulty settings will cause faulty reads of the messages and a lot of error frames will be generated. [5]

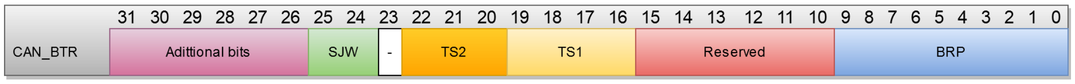


Figure 3: btr - Bit Timing Register (BTR), additional bits, SJW - Synchronization Jump Width, TS1 - Time Segment 1, TS2 - Time Segment 2, reserved, Baud Rate Prescaler (BRP). The timing register defines how the CAN bus is read by the devices and BRP defines how fast the communication is conducted. SJW is adjusted accordingly in order to maintain synchronization

2.3. Micro controller

Author: Hubert Stepien

2.3..1 General Overview

microcontrollers [9] are small computers that are used in many diverse applications such as embedded systems. These processors are often much smaller in size when compared to a standard home computer making them ideal to use in size constrained environments. Unlike a home computer, a microcontroller is not designed to be an all in one computing unit. Instead before use, they have to be programmed for them to be able to do what the programmer wants it to do, what this means is that they are most often optimized for outstanding performance at one or a few tasks such as controlling a coffee machine in an office. MCU's are built with the following core hardware [9]:

1. Central Processing Unit (CPU)
2. Random Access Memory (RAM)
3. Flash Memory
4. Input/Output Ports (I/O ports)

The above table describes the bare minimum of what needs to be on board of a microcontroller for it to be called a microcontroller. Nowadays MCU manufacturers can equip their MCUs with diverse other features such as Analog to Digital and Digital to Analog converters, timers, counters, crystals as well as communication protocols that can be useful in the development of an embedded system [10]. Communication protocols included on an MCU can be but are not confined to support for Serial Peripheral Bus (SPI)[11], I^2C[12], CAN[13], and Universal Serial Bus (USB)[14].