NTNU
Norwegian University of
Science and Technology

Project Thesis TFE4580

Automotive embedded system implementation using System-on-Module

Trym Sneltvedt

December 2019

Supervised by Bjørn B. Larsen

Unfinished version!
Notes and comments are colored red.

Contents

1 Introduction

1.1 Background

1.1.1 Revolve NTNU

Revolve NTNU is a student organization dedicated to constructing electric formula cars for the international Formula Student competitions. In the course of one year, an electric vehicle (EV) is designed, constructed and tested before entering multiple competitions, facing off against comparable vehicles and teams from other universities from around the world.

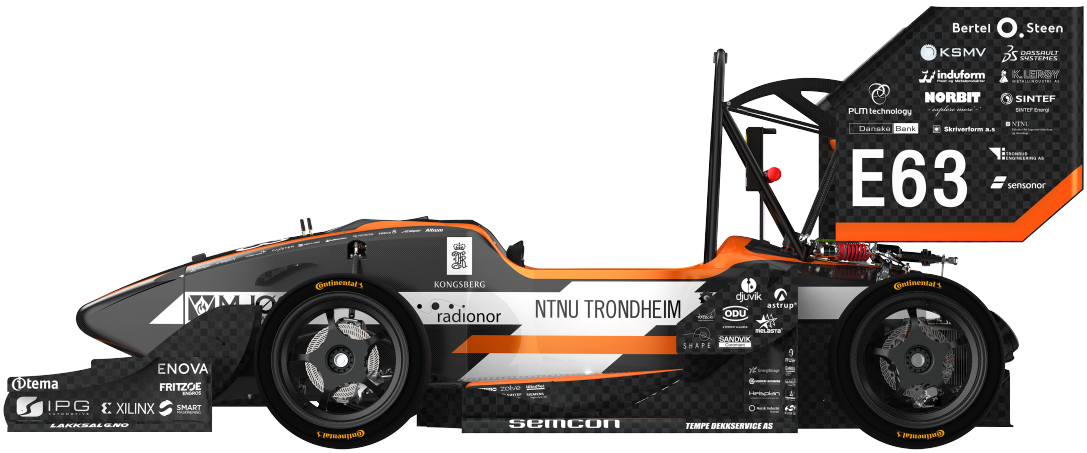

Figure 1: Revolve NTNU 2019's electric vehicle, Nova

The combination of short development time and a team consisting of mostly new, inexperienced members makes for several interresting problems. This report will focus on a specific embedded system, the vehicle control unit (VCU), and the issues encountered when developing it. As the EV is almost exclusively made from custom parts, the different systems on the car depends heavily on each other. This means that both changes done the VCU will invariably affect the rest of the vehicle and issues with other systems will most certainly affect the VCU.

To get a clearer picture of necessary changes to the VCU, we start with the system re-

quirements. analyze last year's VCU, VCU19.

1.1.2 The vehicle control unit

The vehicle control unit is the central control system of the electric vehicle. That is, it is responsible for inputting sensory data from the vehicle and the driver, and then output fitting data to electric motors connected to each wheel.

In addition, the VCU is to

- put the vehicle into Drive-Enable mode when certain criteria are met
- perform safety checks on inputted data and take appropriate action if errors are found
- interface with the inertial navigation system (INS)
- interface with the data logger provided by competition officials, as per EV 4.6 [4]

The team member responsible for the VCU is not responsible for the implementation of the control algorithms that takes vehicle and driver input and outputs motor set points. The control algorithms performs torque vectoring, a technique for intelligently varying the torque on each wheel (reference here), increasing the maneuverability of the EV. The VCU is responsible for providing an environment for the control algorithms to run.

Since the vehicle is capable of speeds exceeding 110 kilometres per hour [6], the control algorithms cannot use more time than necessary. As the deadlines the VCU has to uphold are critical for the survival of the vehicle and possibly the driver, it is per definition a *hard real-time embedded system*. However, the exact deadline is unknown. The rule of thmub used by members is that the control loop has to run in at no lower than 100 Hz. (Should really find some data on this.)

1.1.3 Analysis of VCU19

To fulfill the requirements of the control algortihm, VCU19 was designed around a new processing platform, the Xilinx Zynq-7000. It is a SoC with a dual core ARM Cortex A9 processor and embedded FPGA [5]. There were however issues with this implementation. Advanced processing units like the Zynq-7000 are commonly only available in ball grid array (BGA) packages, meaning packages with all pins placed in a grid pattern on the bottom. This makes hand soldering impossible, a reflow method has to be used instead. This makes assembly and debugging harder, a serious concern considering the rapid development methodologies necessary in Revolve NTNU. The high complexity of design also compelled the previous designer to opt for the smallest available package, a 225 pin BGA package. This limits the available peripherals, but was deemed a necessary compromise during the 2019 season.

1.2 Scope

This report covers the design of VCU20. The main focus of the new design is to reduce the complexity of the system while keeping the performant Zynq-7000 SoC platform.

2 Theory

2.1 Ball grid array PCB layout

Ball grid array (*BGA*) refers to a packaging type used for surface mount IC's. BGA packages feature all interconnects on the bottom of the package and allows for a higher number of connected pins than with pins on the edges of the package, see figure 2.

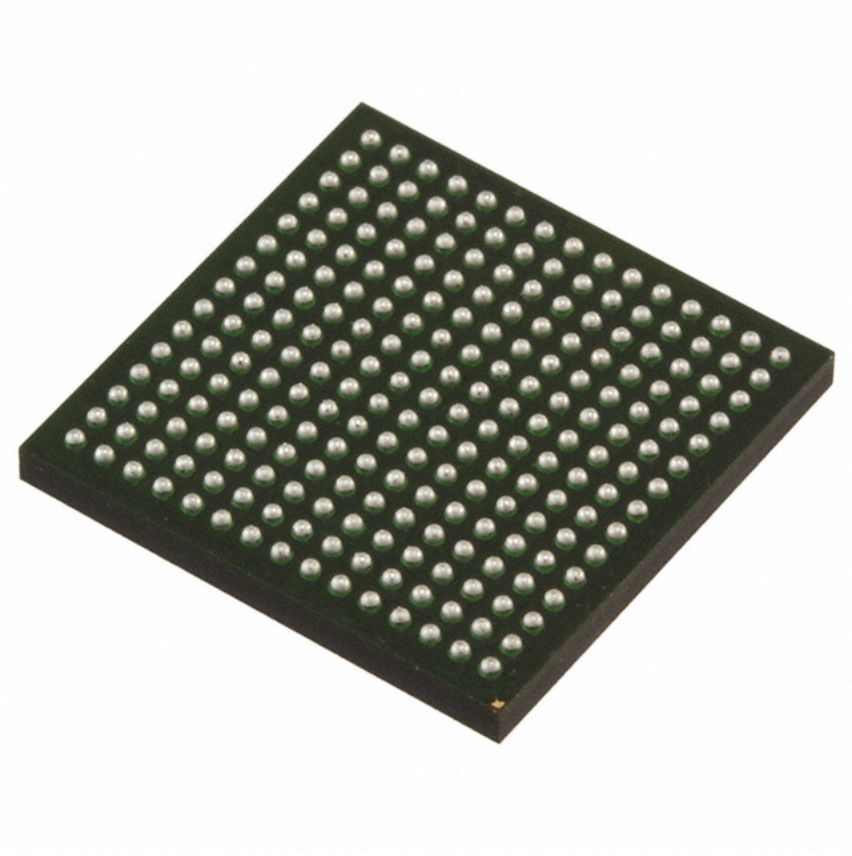

Figure 2: CLG255, a 15-by-15 pin BGA package used by the Zynq-7000 series SoC's [7]
Do I need special permission to use an image from Xilinx?

As the pins are hidden when the package is mounted on a PCB, BGA packages are must be soldered by other methods than classical hand-soldering. Reflow soldering is a popular method for soldering BGA's where the entire package and adhering PCB are heated to the point where all the solder melts. This is typically performed in a dedicated reflow soldering oven, but can also be performed using a hot-air gun or similar.

2.2 CAN-FD

The different embedded systems on the vehicle communicate with each other using CAN-FD. CAN, or *controller area network* is a network bus developed by Bosch. It has been the de-facto standard communication platform for automotive applications since the mid-1990's. An improved version, CAN-FD (flexible data-rate) was released by Bosh in 2012, it differs from CAN in that the body of each message (called *frames* for CAN) are transmitted at a higher frequency than the header and the allowed frame size increases from 8 to 64 bytes. This is the communication bus currently used at Revolve NTNU.

CAN and CAN-FD is a one-to-many bus system where each frame is given an identifier which determines the priority of the frame. When the bus is free, all connected devices can begin transmitting their frame, but the frame with the highest priority will take precedence on the bus. When a frame is transmitted on the bus, any connected device are able to read it. When the frame has successfully been transmitted the bus is again free and a new transmission can begin.

2.2.1 CAN-FD bus utilization

Almost all messages sent over the CAN-FD bus are sent at a regular interval. The size of the different messages are also known, this means that we know how long each message will occupy the bus for. These facts means that the frames on the bus can be modelled in the same way as tasks in a rate-monotonic scheduled (RMS) system, known from real-time theory. This means the standard formula for utilization can be used when analyzing CAN and CAN-FD buses. The equation for utilization U on a bus is given in equation 1.

$$U = \sum_{i=1}^{N} \frac{C_i}{T_i} \tag{1}$$

C_i is the execution time and T_i is the execution period for task i. In our case this is translated to the time frame i is active on the bus and the period at which it should be transmitted.

2.2.2 Maximum utilization

We can use the *Utilization-based schedulability test* to determine whether all deadlines are met for a set of tasks. The criteria can be seen in equation 2. It is important to note that this test is sufficient but not necessary. This means that a passing test guarantees all deadlines are met but a failing test does not guarantee missed deadlines.

$$U = \sum_{i=1}^{N} \frac{C_i}{T_i} \leq N \left(2^{\frac{1}{N}} - 1\right) \tag{2}$$

N denotes the number of different tasks being scheduled. Since the number of different

messages on the CAN-FD buses is quite high, we might as well look at what happens to the criteria when N goes towards infinity.

$$\lim_{N\to\infty} N\left(2^{\frac{1}{N}} - 1\right) = \ln 2 \approx 0.693 \tag{3}$$

From equation 3 we can deem that as long as the bus load on each of the CAN-FD buses are below 69.3%, no messages are lost. This does of course not mean that the vehicle becomes unusable, figure 6 shows that the load was above the the limit for both buses, and the vehicle worked fine. The embedded systems are naturally made to be resilient to frame loss.

However, the limit serves as a good target for max bus load, and we will work towards lowering bus load to below this limit.

2.2.3 CAN-FD utilization

From the RMS utilization equation we know that we need the execution time C and period T for each task. While the period for each CAN-FD message is easy to determine, the execution time takes slighly more effort.From now on we will call the execution time for a message m the *transmission time* and we will denote it C_m. For a standard CAN message it would be sufficient to divide the number of bits in the message by the operating frequency of the bus. This is made slightly more difficult as CAN-FD operates on two different frequencies, one for header transmission and one for data transmission. We can split look at the total transmission time as a sum of the transmission time for each of the frequencies , see equation 4.

$$C_m = C_s + C_f \tag{4}$$

C_s is the transmission time for the message header. It is independent of frame size and the calculation can be seen in equation 5.

$$\begin{aligned} C_s = &\frac{(SOF + ID + r1 + IDE + EDL + r0 + \frac{BRS}{2} + \frac{CRCdel}{2}) \cdot 1.2}{t_x} \\ &+ \frac{ACK + DEL + EOF + IFS}{t_x} \end{aligned} \tag{5}$$

C_f if the transmission time for the frame data, sent at a higher speed than the header. It is calculated as in equation 6 where D_f is the data size in bits.

$$C_f = \frac{(D_f + \frac{BRS}{2} + ESI + DLC + \frac{CRCdel}{2}) \cdot 1.2 + CRC + BS}{t_y} \tag{6}$$

Because of error detection mechanisms defined in the CAN-FD protocol, the transmission time for payloads larger than 16 bytes is different from payloads smaller than 16 bytes.

For payloads smaller than 16 bytes the CRC (Cyclic Redundancy Check) is 17 bits and BS (Bit Stuffing) is 5 bits. For payloads larger than 16 bits, CRC rises to 21 bits and BS to 6 bits.

More information about calculating the worst case transmission time (WCTT) is needed

2.3 Differential signalling

When dealing with high-speed, low-power signals in electronic systems, minimizing noise is of the utmost importance. One technique for dealing with this is *differential signalling*. Simply put, it means driving two conductors instead of just one. Single-ended signalling, the conventional method, means that signalling lines are referenced to a common ground. In differential signalling, two lines are used for each signal line. These lines are referenced to each other instead of to a ground shared by all signals. This serves to remove noise that might be present on the ground plane and, when the two conductors are placed close to each other, excellent protection against external interference. When the lines are close to each other, electromagnetic fields that normally would introduce noise to the signal will optimally affect both conductors by the same amount, and since we only look at the difference between the two lines, the noise is elegantly subtracted away.

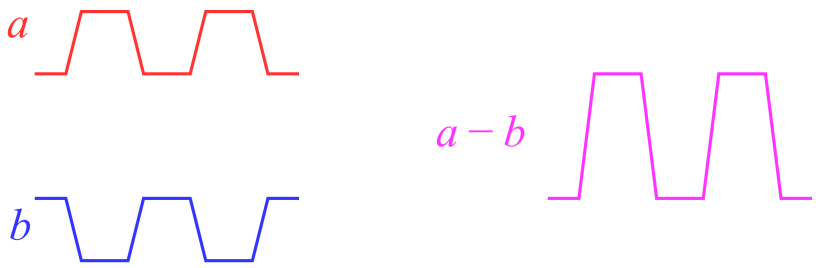

Figure 3: Differential signalling illustrated. The signal is the difference between the signals on line a and b.

3 Method

This section covers the choices made during the design of VCU20. Each subsection takes on a different challenge encountered in the 2019 season.

3.1 System-on-Module

The Zynq-7000 platform provided a sufficient platform for running the VCU control loop and the TV algorithm. However, the Zynq-7000 SoC product line are only available in BGA packages. This introduced unwanted complexity to the design, both in term of requirements to the PCB manufacturing, and to the assembly of the PCB, which is a process that normally is performed by team members at the electronic workshop at Revolve NTNU's headquarters.

In regards to the PCB production process, the BGA package of the SoC adds two kinds of complexity to the design. The first is the need for very tight tolerances. To be able to properly extract all necessary signals from the SoC, a trace width of 0.1mm is needed. Secondly, a higher number of PCB layers are needed. This is mainly because the SoC requires multiple different supply voltages, 1V, 1.8V and 3.3V in the case of Zynq-7010. Both these requirements increase the price of PCB production drastically. Although PCB production houses have increased in number over the last few years, thereby driving the manufacturing cost down, special production techniques like more than 4 layer PCB's and 0.1mm trace widths still introduces high costs to production runs. A quick comparison on pcbway.com shows that 5 PCBs with 4 layers and a tolerance of 8mils $\approx$ 0.20mm and 4 layers costs 49USD to produce, while the same amount PCBs with tolerances of 4mils $\approx$ 0.10mm and 8 layers costs 388 USD. Additionally, the lead time increases from 4-5 days to 7-8 days. While these numbers might be acceptable for a large production run by a large company, it is most definitely an issue for a voluntary student organization like Revolve NTNU.

The solution to this issue is to employ as System-on-Module. Instead of mounting the SoC directly on the VCU20 PCB, we can purchase off-the-shelf SoC modules. These modules consists of a small PCB featuring a SoC and some peripherals like external DDR RAM and non-volatile flash memory. The modules interconnects with the VCU PCB by one or more PCB connectors which are easier to route. The downsides to using these modules is mainly the cost. The module utilized for VCU20, Enclustra Mercury ZX5 with Zynq-7015 (figure 4), has a unit price of 318USD. However, as they are modular, they can easily be reused between seasons. Reusing soldered ICs is not common practice at Revolve NTNU, so this should mitigate the high entry cost at least to some extent.

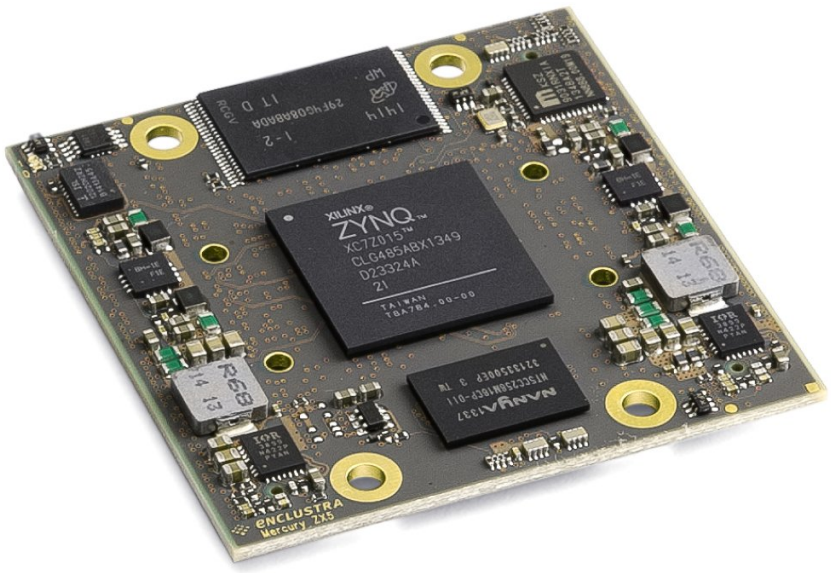

Figure 4: Enclustra Mercury ZX5 SoM

but at the cost of high design complexity. By utilizing a Zynq-7000 based System-on-Module (SoM), we avoid the hard parts of PCB layout, and simply design the VCU as a *breakout board* for the SoM. Using a SoM also gives us access to peripherals that would

have been difficult to implement by ourselves, specifically Ethernet, external flash memory and external double data-rate (DDR) memory.

3.2 Partitioning interfaces

As the VCU is an integral part of the EV, issues regarding the vehicle are also issues for the VCU. This section is dedicated to some of the issues encountered during last season, which hopefully can be solved by improving the design of the VCU.

In Nova, the main communication channel for the embedded systems were two CAN-FD buses running at a base frequency of 1MHz and an data transfer frequency of 4MHz. A system overview can be seen in figure 5.

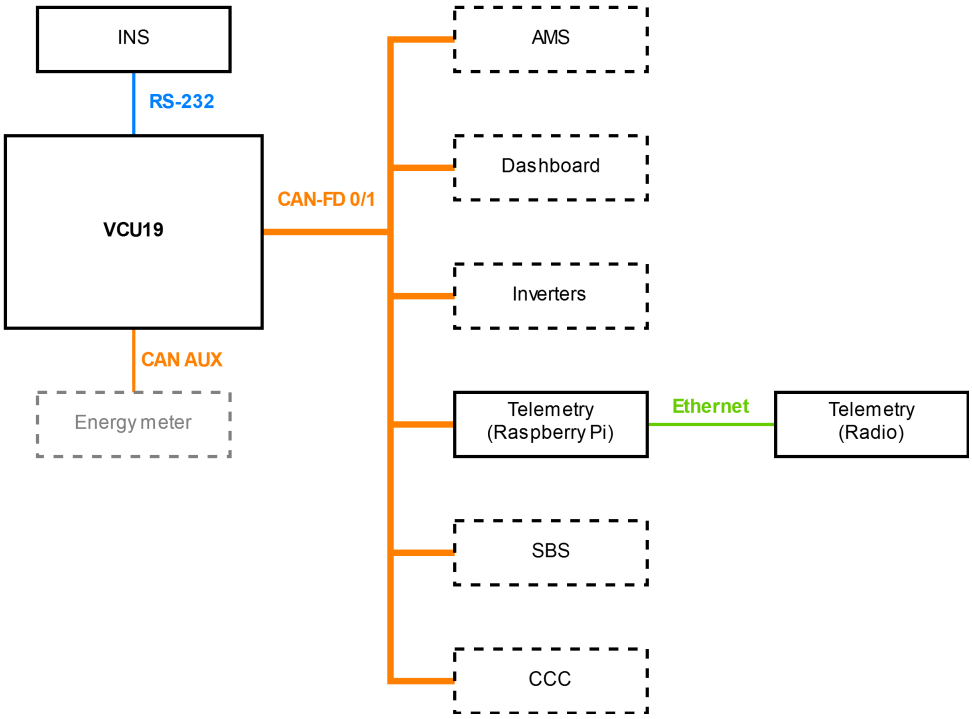

Figure 5: VCU19 interface overview. Systems developed by other team members have dotted borders.

There are several points where this system can be improved. They will be discussed in detail in the following subsections.

3.3 Partitioning CAN-FD buses

A trace of the load on both CAN-FD buses during normal operation can be seen in figure 6. Note the increase in load on bus 1 halfway into the trace. This is caused by the car entering "Drive-enable mode" where the VCU starts transmitting set points to the inverters.

From the calculations in section 2.2.2, we know that to ensure that no frames are dropped

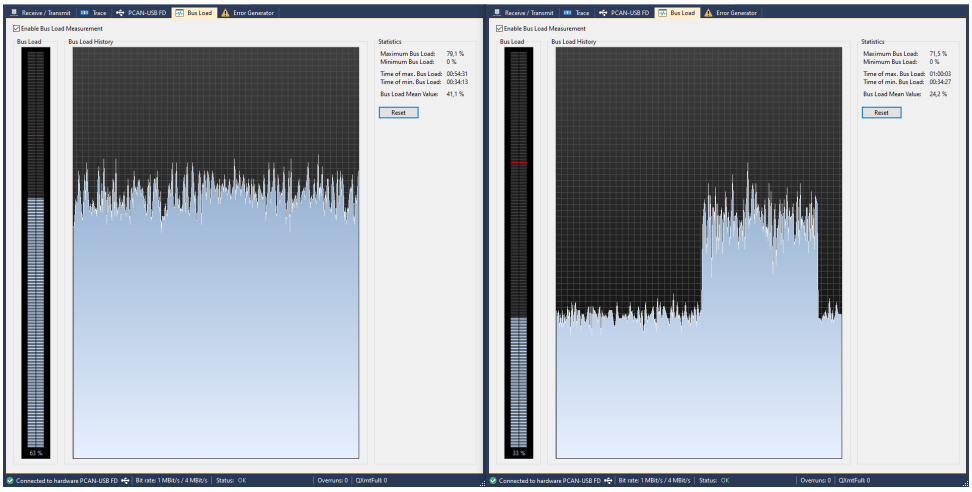

Figure 6: CAN-FD bus load during typical operation

(i.e. all deadlines are met) the bus utilization should be no more than 69%. For bus 1, this seems to be in order. The trace tells us that the peak utilization is 71.3%, but this is an outlier and probably not an issue. However, bus 0 seems to have a somewhat high bus load continuously, staying around 70% load with peaks as high as 79.1%. This is an area that can be improved.

To start of, we will take a closer look at the frames that are transmitted on each bus. Last season, the team started the process of transitioning from an in-house protocol on top of CAN/CAN-FD (called the Revolve Protocol), to a standardized alternative, UAVCAN. One of the benefits of using UAVCAN is a simplified message definition process. Messages are defined in document schema definition language (DSDL) files and can be converted to C for use in embedded systems and high level data formats like JSON for use in analytics software. As a part of this transition, a simple Python script for calculating the load on each bus based on message size and period was written by Åsmund Eek, a team member from last year that was responsible for the CAN/CAN-FD systems. This code was slightly modified to fit the purposes of this report.

Calculations on bus load for different systems and different CAN-FD partitioning

Summing up, a system overview of VCU20 can be seen in figure 7.

3.4 Ethernet for telemetry

It is necessary to retrieve data on the vehicle and the different systems on it during races. This is both for later analysis and to indicate to the team if there might be issues with the vehicle before things go wrong. To achieve this, Nova utilized a wireless communication solution from Radionor, the CRE2-144-LW [1]. UDP is used to interface with the radio and the physical layer is Ethernet. As no other system on Nova was equipped with Ethernet, the solution was to place a Raspberry Pi 3 B+ in the vehicle, and connect it to the CAN-FD buses with two PCAN-FD USB dongles. The Raspberry Pi would simply gather all available data from the CAN-FD buses and transmit it to the radio over Ethernet.

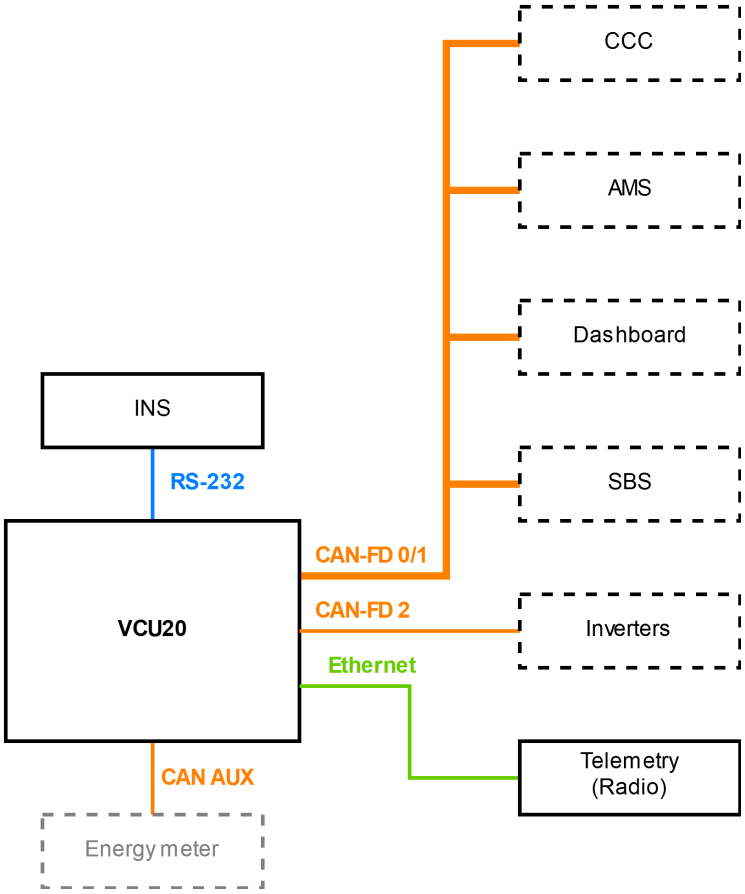

Figure 7: VCU20 interface overview

As the SoM chosen for VCU20 is equipped with an Ethernet-PHY interface, it is an opportunity to greatly simplify the telemetry system. By connecting the VCU directly to the radio directly using Ethernet, the total weight and complexity of the vehicle can be reduced. The downside to this is that the complexity is moved to the VCU, as it now has to communicate with the radio in addition to everything else.

It is important that the telemetry system is able to perform at least as well as in Nova. As the previous telemetry system gathered data from both CAN-FD buses and sent the information over Ethernet to the radio, we must consider the bandwidth of both CAN-FD and Ethernet. The buses on Nova ran with a maximum data rate of 4MHz, although it does not use this data rate all the time, we can simplify and say it has a bandwidth of 4Mbps. This means a total bandwidth of both CAN-FD buses of 8Mbps. The Raspberry Pi 3 B+ has a maximum Ethernet bandwidth of 300Mbps [3]. This means the Ethernet interface on VCU20 has to achieve a minimum bandwidth of 8Mbps. According to the specifications, the Mercury ZX5 module is capable of Gigabit Ethernet (1Gbps) [2]. However, the actual speed of the interface is heavily dependent on the quality of the PCB layout. Gigabit Ethernet consists of four differential pairs, each of which should be matched in length to each other and within the pair to achieve maximum bandwidth.

4 Results

4.1 Hardware design

The electronic design automation *(EDA)* software used for schematics and pcb layout, *Altium Nexus* supports multiple features which shortened the design process significantly. Firstly, since VCU19 used *hierarchical design* (i.e. project is composed of multiple sheets and sub sheets), large parts of the schematics developed during last years design period could be reused.

Simple sub circuits like CAN-FD transceivers could be directly copied without issue. In addition, the sub sheets are modular in the sense that they appear as an electrical component with a specific set of interfaces, or *ports*. This means that sheet using the same set of ports can be interchanged freely. The modularity of the sheets was heavily used during the development of VCU20. The new processing unit features a super set of the VCU19 SoC interface, meaning they are perfectly compatible.

There is also the possibility of reusing sheets in the same project. VCU20 has 4 CAN-FD transceivers, but since the transceiver schematic is identical for all 4 interfaces, the sheet can simply be repeated.

There are also features for reuse in the PCB layout part of the design. Altium Nexus has a feature called *rooms* which is a grouping of component footprints and how they are connected. It is possible to copy the layout if a room to other rooms. The designer chooses how rooms are generated, and by default they are created per sheet. This means that the layout for one CAN-FD transceiver can be perfected and then copied over to the other 3 transceiver rooms.

The PCB was produced by Simpro AS as they are sponsoring Revolve NTNU. The finished, soldered PCB can be seen in figure 8.

Must mention CAN-FD interfaces (whether they work or not)

4.2 Ethernet bandwidth

Benchmark Ethernet interface bandwidth. Calculations showing whether it is able to transmit all data from CAN-FD buses over telemetry

4.3 Further work

The proposed solution of an extra CAN-FD bus solely used by the inverters and VCU has room for improvement. CAN-FD buses are designed for communication between several embedded systems. When there are only two embedded systems connected to the bus a point-to-point communication system might be more suitable, like the Ethernet link between the VCU and the telemetry radio. It would be best to utilize one of the unused interfaces on the module as this would reduce the required amount of supporting circuitry. The module is equipped with 4 general purpose Gigabit Transceivers, these would probably

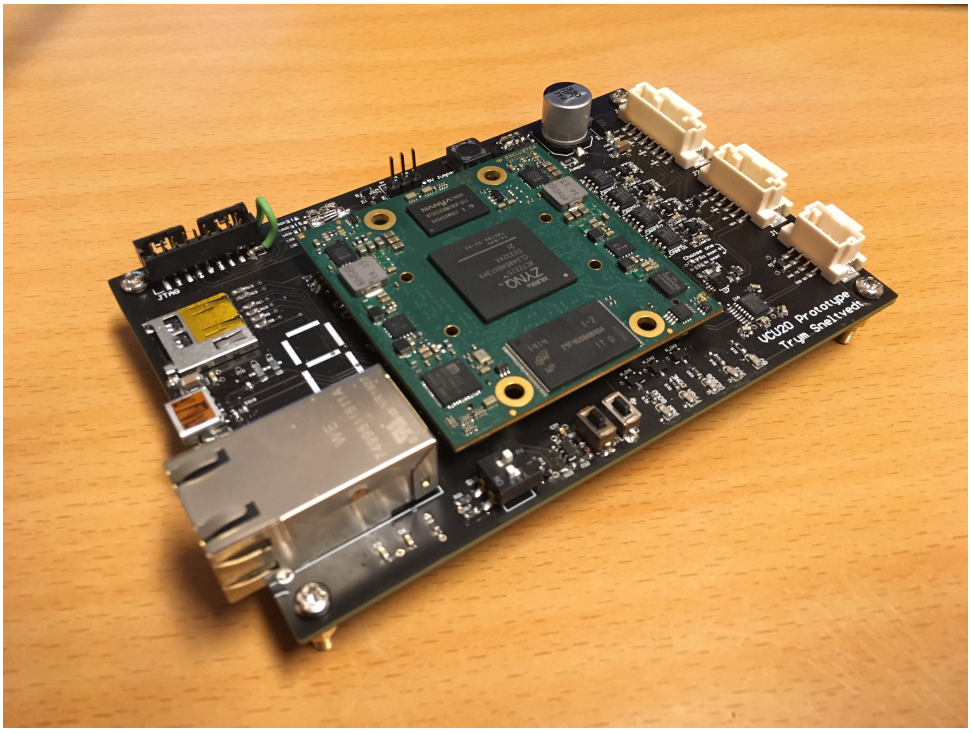

Figure 8: VCU20 PCB with all components and Mercury ZX5 module attached

be a good fit.

Another solution that the author would recommend to next year's team is to consider merging the VCU and the inverters. The VCU is currently overpowered when considering the relatively simple tasks it performs and it should at least be researched whether the Zynq-7000 could run the inverter control loops in addition to the torque vectoring algorithms. If the ZX5 module proves to be too weak, the team should consider transitioning to the Xilinx Zynq UltraScale+ MPSoC platform. They are equipped with one quad/dual-core ARM Cortex A53 application processors, one dual-core ARM Cortex R5 real-time processor and a FPGA. This should be more than sufficient to run both systems. Possible pitfalls are competition rules, they must be examined closely. The Ultrascale platform is available as modules as well, although at a higher cost compared to the Zynq-7000 series.

5 Discussion

Reflect on results.

References

[1] Radionor Communications AS. *CRE-144-LW product information page*. URL: `https://radionor.no/product/cre2-144-lw/`. (accessed: 05.12.2019).

[2] Enclustra. *Mercury ZX5 product overview*. URL: `https://www.enclustra.com/en/products/system-on-chip-modules/mercury-zx5/`. (accessed: 11.12.2019.

[3] Raspberry Pi fundation. *Raspberry Pi 3 B+ product overview*. URL: https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/. (accessed: 11.12.2019.

[4] Formula Student Germany. *FSG rules 2020*. URL: https://www.formulastudent.de/fileadmin/user_upload/all/2020/rules/FS-Rules_2020_V1.0.pdf. (accessed: 05.12.2019).

[5] Xilinx Inc. *Zynq-7000 product brief*. URL: https://www.xilinx.com/support/documentation/product-briefs/zynq-7000-product-brief.pdf. (accessed: 04.12.2019.

[6] *Revolve NTNU 2019 Nova information page*. URL: https://www.revolve.no/electric-vehicle/. (accessed: 04.12.2019).

[7] *Xilinx CLG225 BGA package*. URL: https://media.digikey.com/Photos/Xilinx%20Photos/122;225CSBGA-1.5-13x13;CLG;225.JPG. (accessed: 09.12.2019).

Need more references

Appendix

Layout

Schematics

CAN-FD calculations

Also including source code

Benchmarks