18th IMEKO TC10 Conference
"Measurement for Diagnostics, Optimisation and Control to Support Sustainability and Resilience"
Warsaw, Poland, September 26–27, 2022

Card Level Management Solution for the BRAINE edge framework

Balázs Scherer

Department of Measurement and Information Systems
Budapest University of Technology and Economics
Budapest, Hungary
scherer@mit.bme.hu
+36 1 463-2066

***Abstract* – The objective of the BRAINE (Big data pRocessing and Artificial Intelligence at the Network Edge) project is to boost the development of an Edge framework focusing on energy efficient hardware and AI empowered software, capable of processing Big Data at the Edge, supporting security, data privacy, and sovereignty. The BRAINE's Edge framework is designed to be flexible and support many types of data processing and storage cards. Because of this flexibility and the wide range of heterogeneous cards a sophisticated low-level board/card management functionality is needed. The board management architecture of BRAINE is a hierarchical design, which consists of a central board management node (BMC: Board Management Controller) and a distributed low-level board management hardware present on all the nodes (BMMC: Board Management MicroController). This paper describes the functionalities and presents implementation details of the low-level board management controller the BMMC.**

Keywords – Diagnostics, Edge framework, Board Management, Ethernet over USB, Big Data Processing, AI on the edge.

I. INTRODUCTION

The goal of the Edge framework of BRAINE [1] is to leave behind cloud computing and place the intelligence for processing data closer to the sources. This offers many technical advantages, such as reduced latency, secure decentralized processing and storage, scalability at lower complexity, versatility to adapt the nodes to the underlying application to be serviced, etc... The Edge computing power of the BRAINE framework can dramatically boost services and applications by supporting artificial intelligence (AI) natively, instead of relying on AI in the cloud. Edge computing supporting AI (without cloud intervention) is the only technology that will enable many of the long-awaited game changers: Factory 4.0 and smart manufacturing, 5G, Internet-of-things, self-driving vehicles, remote robotics for healthcare, machine vision, among others.

To introducing AI to edge computing a specific hardware needs to be designed with big data processing and AI in mind. The BRAINE project's overall aim is to boost the development of such Edge frameworks and provide an example for such a framework.

II. THE BRAINE EDGE MICRO DATA CENTER

The development of the BRAINE Edge Micro Data Center is a complex project containing many sub projects (16 hardware topics and 18 software topics are identified). The hardware part consists of a design of a modular multi slot card holder, with sophisticated cooling features to ensure the operation of high powered computing cards (150W of possible power pro card, resulting an overall of a 1.5kW of backplane power for an 8 slot system), the cards are using the standard 3U rack size, with variable depth and they are individually replaceable or reconfigurable. A plan of the BRAINE Edge Micro Data Center physical layout is shown on Figure 1.

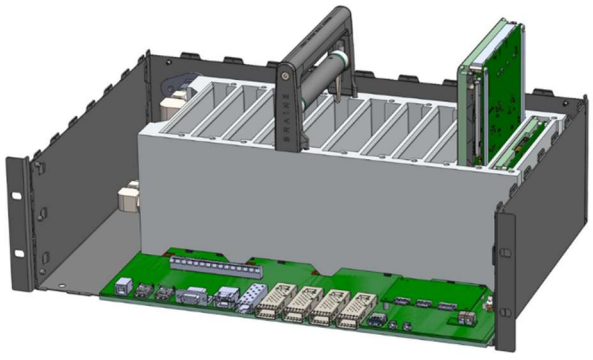

Fig. 1. A plan of an 8 slot BRAINE Edge Micro Data Center [2]

For data connection the BRAINE Edge Micro Data Center is using a Dual layered switching: Both (10/25/100G) Ethernet and PCIe Gen 4, shown on Figure 2.

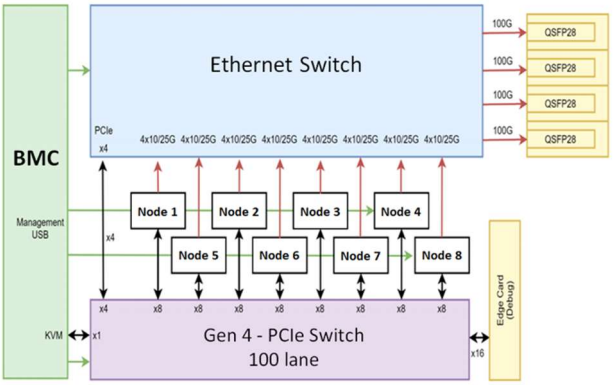

Fig. 2. Architecture of the BRAINE Edge Micro Data Center [2]

As described above, Edge framework designed during the Braine project must be flexible and support many types of data processing and storage cards. The so called Compute node slots (there are 8 of such slots in the current architecture represented as Node X on Figure 2), can support the following card options:

COMe CPU card: Support ComExpress Type VII CPU modules or AMD (EPYC) or Intel (Xeon D) modules. It has at least 64 GByte memory, and on-board NVMe storage. It has minimum 2x10 G Ethernet interface, and a minimum of x8 Gen3 PCIe connection.

GPU card: Support Xavier AGX (32GB) AI SoC module with 512-Core Volta GPU Tensor Core, and 8-Core ARM v8.2 64-Bit CPU. It also has NVMe storage and at 32 GB of system memory. It provides at least an 8xGen3 PCIe host interface and a 2x 10 G (25 G) Ethernet interface.

NVMe card: Provides storage capacity of at least 8 TB/card capacity and at least 8x Gen 3 PCIe host interface.

ARM card: It should provide a power efficient processing capability. It is currently under specification.

FPGA card: It is intended to provide a highly flexible reconfigurable hardware acceleration. It is currently under specification.

Currently there are working prototypes of the COMe CPU card and the GPU card, the other card types are under development or manufacturing.

Because of this flexibility and the wide range of heterogeneous cards shown above a sophisticated and hierarchical low level card management functionality is needed.

This hierarchical management design consists of a central board management node (BMC: Board Management Controller) shown on Figure 2., and a distributed low-level board management hardware present on all the nodes (BMMC: Board Management MicroController). The goal of this paper is to describe the functionalities and present implementation details of the low-level board management controller the BMMC of the BRAINE Edge Micro Data Center framework. Details of the BMC (Board Management Controller) functionality will be included in a separate paper.

III. FUNCTIONALITY OF THE BMMC

The main goal of the BMMC is to support the operation of its Compute node slot card (COMe CPU card, GPU card etc.), and provide management interface functionality to the card for the BMC.

The functionalities of the BMMC can be divided to two parts: the first part contains the ***Common functionality*** which is same for every Compute node slot card of the BRAINE Edge Micro Data Center framework. The second part is the ***Card specific functionality***, which is needed because of the heterogeneous cards supported by the firmware.

The ***Common functionality*** of the BMMC include the following:

- Providing communication interface to the higher-level Board Management (BMC)
- Node startup management: power supply enabling and switching (the functionality is card independent, but the execution is card dependent)
- Temperature monitoring
- Current consumption monitoring
- Handling of the TPM (Trusted Platform Module) chip of the board
- Providing debug interface for the card's computation unit during the development. This has card dependent parts, but every computation module has some kind of debug feature, usually an UART like interface which need to be redirected to the BMC.

The *Card specific functionality,* of the BMMC as it is included in its name highly depends on the given card. For example, such functionality can be the HDMI interface management for the NVIDIA GPU card.

IV. HARDWARE OF THE BMMC

The BMMC is based on a low-cost, low-power microcontroller, the STM32L4R5VGT6. Figure 3. shows the high-level block diagram of the microcontroller.

The selected microcontroller has a wide range of peripherals (I2C, SPI, UART, USB), up to 16 analog and up to 81 digital pins, so it offers the necessary connectivity to be used as a management device. The high clock frequency (120 MHz), the large flash (1 MB) and RAM size (640 kB) allows the implementation of complex and

high-speed communication protocols like USB CDC Ethernet and TCP-IP based on it. It was also beneficial that a NUCLEO-L4R5ZI development board is available for this processor series, so that software development can begin long before the actual hardware panels are manufactured. This was critical because of the current chip crisis, which caused delays in hardware production.

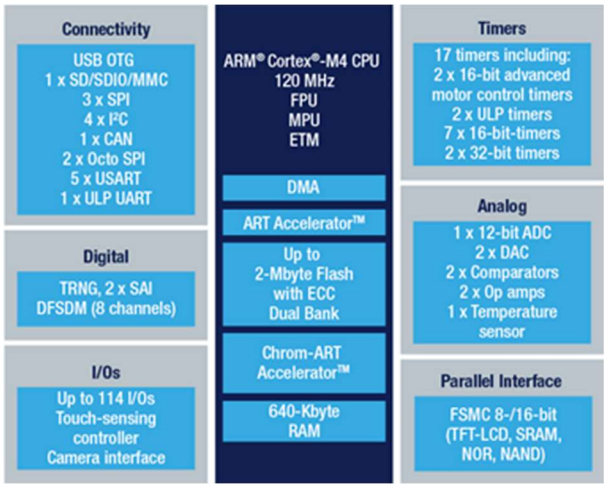

Fig. 3. Internal architecture of the Board Management Microcontroller [3]

An example for the real hardware a CPU Node with AMD EPYC, Xeon D module integrating a BMMC module is shown on Figure 4.

Fig. 4. Manufactured CPU Node

The CPU card is supplemented with a debugger PCB enabling its development. This debugger PCB can be seen on the upper side of the picture together with the external debugger enabling the software development of the STM32L4R5VGT6 microcontroller. At the bottom of the picture the card interface connector is visible, with a development connector which can give power supply to the card and enables the simulation and testing of the USB communication between the BMMC and the BMC.

V. SOFTWARE ARCHITECTURE OF THE BMMC

The software of the BMMC has a layered architecture according to the industrial trends (Figure 5). This software architecture consists of the following main layers and blocks.

A. Manufacturers SW Library:

The software architecture of the BMMC is based on the microcontroller vendor's firmware library, the ST Cube. *ST Cube HAL* [4]: The ST cube provides high level APIs the so-called HAL (Hardware Abstraction Layers), which can be generated using a graphical configuration system the ST Cube Mx. This high-level APIs can provide great functionalities, but the high-level approach also has many drawbacks (like common event handling of peripheral blocks, etc.). *ST Cube LL*: Besides the high-level APIs the ST also provide a low-level library called ST Cube LL, which can be useful to separate and encapsulate some functionalities of the system.

B. MCU level API

The MCU level API layer provides the BMMC hardware based peripheral configurations, including peripheral property setups and microcontroller PIN specifications. This layer has two parts, the Common driver configuration, and the Card Specific I/O.

The *Common driver configuration* is based on the ST Cube software stack and provides a microcontroller independent abstraction for the main common peripherals. Its purpose is to provide an API for handling the common peripherals of all BMMCs. These common peripherals are providing the interface to the BMC, the TPM chip, the power controls and the BMMC debug. This software part is not subject to any change if a new BMMC type is developed.

The *Card Specific I/O abstraction* implements the BMMC type specific peripheral handlings.

C. Board Level API

The purpose of the board level API is to provide a function support layer to the application based on the peripheral interfaces. The common functionality block of the Board Level API integrates the BMMC's card independent functionality, like BMMC - BMC communication management, power setups, temperature monitoring and TPM chip function interface. The card level functionality is the BMMC card dependent part providing card specific boot procedures and other card specific measurements and setups. The Communication part consists of the Ethernet over USB communication (CDC Ethernet) support and USB DFU (Device Firmware Update) based Firmware update support for the BMMC itself.

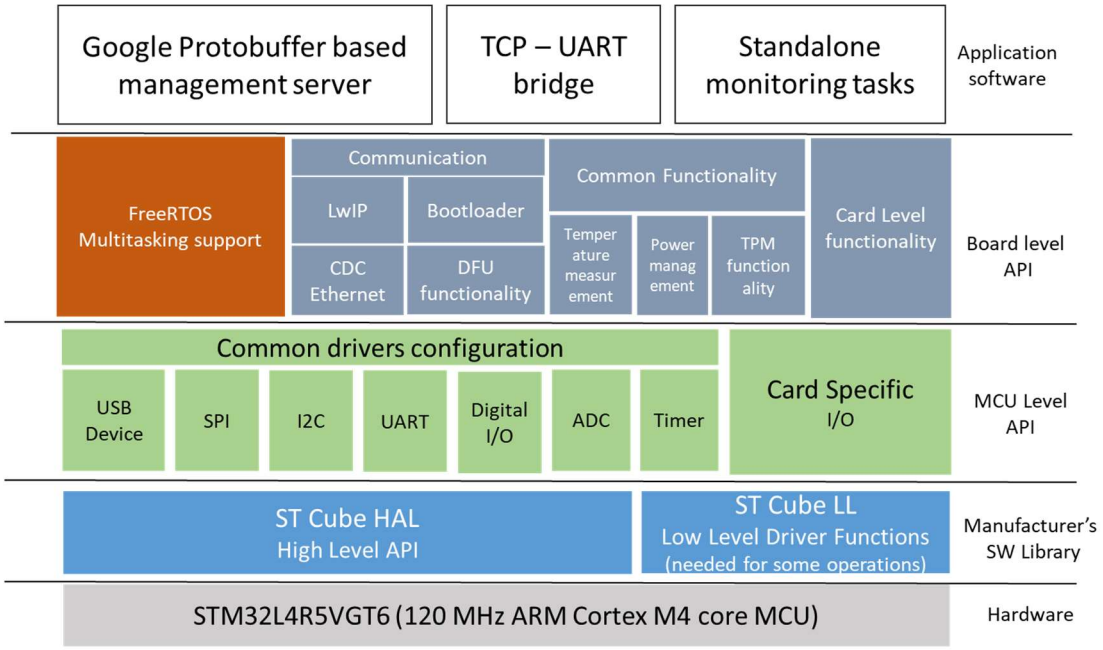

Fig. 5. Software architecture of the Board Management Microcontroller

D. Multitasking support

The complex BMMC functionality requires multitasking: TCP-IP communication, data acquisition, Google Protocol buffer server functionality, UART over TCP bridge etc. need their own threads. This multitasking functionality is supported by the FreeRTOS kernel [5]. To provide future compatibility we use the FreeRTOS kernel through the CMSIS-OS API abstraction.

E. Application Software

The application software consists of three separate parts, the Google Protocol buffer server tasks, the standalone monitoring task, and the UART over TCP bridge functionality.

The Google Protocol buffer server [6] is used to provide an easily extendable, high abstraction level, command-based interface for managing the BRAINE card. The Google Protocol buffer uses a language-neutral, platform-neutral, description for the possible communication messages in a client server communication. By using this abstract description, the communication message encoding and processing can be generated automatically for the BMMC microcontroller and the BMC side (using Linux) too. After the code generation only the hardware specific part needs to be handwritten, therefore this approach provides a highly extendable command interface.

The Standalone monitoring task performs the data acquisitions needed by the card management. This task provides the data for the Google Protocol buffer based communication.

The UART over TCP bridge functionality is used to provide a debug interface for the card's main computation unit. For example, for the "CPU" card the main CPU-s UART console is connected through this bridge to the BMC, which can make this console remotely accessible.

VI. BMC – BMMC COMMUNICATION

The most complex part of the BMMC containing many novelties is the communication between the BMC and the BMMC.

A. CDC Ethernet

The CDC Ethernet USB class was selected to implement communication between the BMC and BMMC. This choice has several advantages from the BMC's Linux point of view: on the one hand, since CDC Ethernet is an existing, widely known USB device class, so it does not require individual driver development. On the other hand, since it sees the connected device through the Linux Ethernet interface, the TCP/IP protocol suite support such as TCP sockets can be used, and thus there is no need to deal with the development of low-level communication protocols.

However, from the side of the STM32L4R5VGT6 microcontroller, this implementation requires significantly more complicated development. The STM32 Cube system has CDC (Communication Device Class) device support, but it only includes a ready solution for VCP (Virtual Com Port) communication. Therefore, for the CDC Ethernet solution, it is necessary to prepare the corresponding USB description files, as well as to manage the special control / setup commands and functions, and to ensure the data transfer. For this otherwise very complicated part, we use a sample unofficially supported by STMicroelectronics as

a framework, and we have expanded it further.

The choice of communication over Ethernet also means that we must also provide TCP/IP support on the STM32L4R5VGT6 microcontroller side. For this, STMicroelectronics proposes and integrates the LwIP protocolstack implementation into the CubeIDE environment. LwIP (LightWeight IP) is a popular TCP/IP protocol stack used in microcontroller environments, which supports low-level transport layer protocols such as UDP and TCP, as well as some application layer protocols such as TFTP, http server, SNMP. In our case, the LwIP support provided for CubeIDE can only be used to a limited extent, because STMicroelectronics provides an example of its simple use only through the Ethernet peripheral of microcontrollers. To use this protocol stack over USB, a so-called network interface is needed, which implements the sending and receiving of TCP/IP packets via the CDC Ethernet interface.

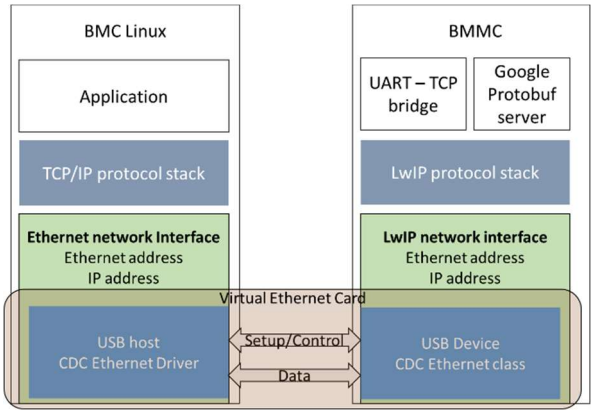

Fig. 6. CDC Ethernet communication between the BMC and BMMC

Communication based on the CDC Ethernet interface is partly difficult because the STM32L4R5VGT6 microcontroller (BMMC) must show itself as an Ethernet card to the Linux operating system (BMC) with its own Ethernet address and Ethernet-related packet filter and link management capabilities, and on the other hand, it must also function as a TCP/IP node. The details of this relationship are presented in Figure 6.

B. DHCP support

It is important for BMMC connected to BMC Linux to have a unique IP address to implement / emulate TCP/IP communication. It is advisable to implement this unique address in way, when the BMC Linux recognize the BMMC as USB CDC Ethernet card, then it start using the DHCP (Dynamic Host Configuration Protocol) server protocol to give a unique IP address to the connected BMMC. This implementation has been done using the LwIP's DHCP client functionality.

C. Google Protocol buffer server

High-level control and management communication between BMC and BMMC is provided through the Google Protocol buffer command serialization and interpretation framework. The Google Protocol buffer framework is able to format messages containing concise binary information from commands specified in a .proto file with special syntax, and can interpret responses also specified in a .proto file from binary response messages. An example for such a .proto file part is presented in Figure 7.

```
message cmdGetTemperature{
            int32 tempChannel = 1;
}

message Command {
   oneof cmd_oneof{
   cmdDebug            debug = 1;
   cmdGetStmId         getStmId = 2;
   cmdGetSwVersion     getSwVersion = 3;
   cmdGetControlState  cmdGetControlState = 4;
   cmdControl          cmdControl = 5;
   cmdGetTemperature   cmdGetTemperature = 6;
   cmdGetPower         cmdGetPower = 7;
   cmdGetVoltage       cmdGetVoltage = 8;
}
}
```

Fig. 7. Google Protocol buffer .proto file example

Binary Protocol buffer messages in BMC – BMMC communication's case are embedded in a TCP stream (TCP port 6000) and supplemented with a simple framing. The simple framing is needed, because the Protocol buffer interpreter itself does not have protection against "slipping" of commands, so if for some reason the message does not start at the beginning of the command, the interpretation cannot encode it.

The Protocol buffer server created on the BMMC runs in a separate thread. Figure 7. illustrates the relationship between the Protocol buffer thread and its environment.

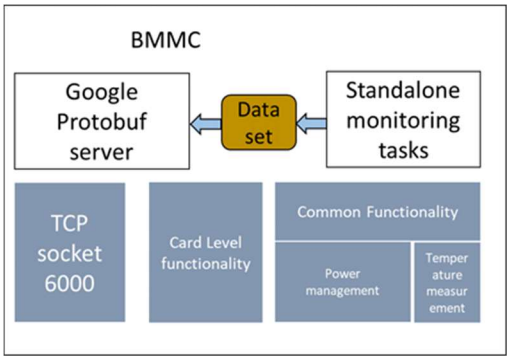

Fig. 8. Google Protocol buffer server in the BMMC software architecture

As shown in the Figure 8., the server thread reads the continuously monitored data collected by the *Standalone monitoring thread*, as well as the static information, such as the version number of the BMMC software and the unique identifier of the microcontroller, from a data table.

Therefore, during frequent data collection requests, there is no low level communication which delays the response. At the same time, in the case of the command arriving for example for the power supply control functions, since the successful or unsuccessful execution is immediately indicated in the response of the command, the Protocol buffer server thread directly accesses the card-level API and executes the necessary commands.

The set of commands transferred in the Protocol buffer is under continuous development, currently the most necessary commands are supported by the BMMC like power up and down, read BMMC software version, read card temperature, and power consumption etc.; but the environment can be expanded very quickly with new commands thanks to the generation of the command interpreter based on the .proto file. A sample Linux based test of BMMC's protocol buffer communication can be seen on Figure 9.

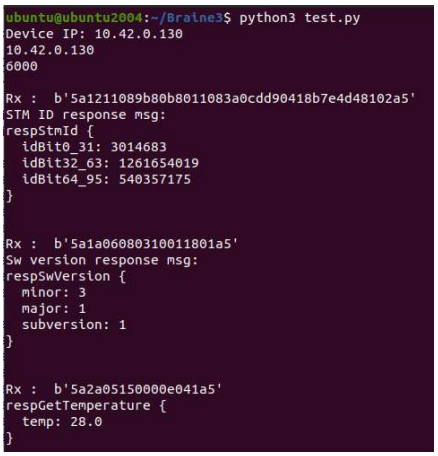

Fig. 9. Sample Protocol buffer communication during BMMC software test

VII. CONCLUSIONS

This paper has presented the goals of the BRAINE Big data pRocessing andArtificial Intelligence at the Network Edge project and its novel approach to provide high performance computation power close to the final application instead of the remote cloud. This approach can provide infrastructure for many of the long-awaited game changers: Factory 4.0 and smart manufacturing projects like [7],[8],[9], 5G, Internet-of-things, self-driving vehicles, remote robotics for healthcare, machine vision, among others.

The paper described in details one of the key components of this framework the BMMC module. Every so called Compute node slot card encapsulates a BMMC microcontroller, which is responsible for the low level card functionality: monitoring and management. The paper focused on the description of one of the main novelties of the BMMC implementation: the communication method between the BMC and the BMMC. The prototype implementation of the BMMC functionality has successfully been done.

VIII. ACKNOWLEDGMENTS

This work was partly funded under the BRAINE project ('Edge computing and AI technology for big data processing'- Horizon 2020 framework Grant Agreement 876967).

REFERENCES

[1] **BRAINE Big data pRocessing andArtificial Intelligence at the Network Edge**: https://www.braine-project.eu/

[2] **ECSEL Research and Innovation Action**.: BRAINE - Big data pRocessing and Artificial Intelligence at the Network Edge, *Second project report on the status of WP2*. 10 April 2022.

[3] **STMicroelectronics,** STM32L4+ SERIES, Ultra-low-power and more performance. https://www.st.com/en/microcontrollers-microprocessors/stm32l4r5vg.html

[4] **STMicroelectronics,** STM32CubeMX & CubeHAL Basics MOOC *(Massive Open Online Courses)*. https://www.st.com/content/st_com/en/support/learning/stm32-education/stm32-moocs/stm32cubemx-and-cubeHhal-basics.html

[5] **FreeRTOS™,** Real-time operating system for microcontrollers, *https://www.freertos.org/*

[6] **Google Protocol Buffer,** Developpers guide: https://developers.google.com/protocol-buffers/docs/overview

[7] **Ildikó Bölkény:** AI Based Detection of Gas Hydrate Formation, *17th IMEKO TC 10 and EUROLAB Virtual Conference "Global Trends in Testing, Diagnostics & Inspection for 2030" October 20-22, 2020 pp.: 202-207*

[8] **Zsolt János Viharos, Richárd Jakab:** Reinforcement Learning for Statistical Process Control in Manufacturing, *17th IMEKO TC 10 and EUROLAB Virtual Conference "Global Trends in Testing, Diagnostics & Inspection for 2030" October 20-22, 2020 pp.: 225-234*

[9] **Eckart Uhlmann, Julian Polte, Claudio Geisert:** Condition Monitoring Concept for Industrial Robots, *17th IMEKO TC 10 and EUROLAB Virtual Conference "Global Trends in Testing, Diagnostics & Inspection for 2030" October 20-22, 2020 pp.: 253-257*