APP4MC RaceCar: A Practical ADAS Demonstrator for Evaluating and Verifying Timing Behavior

Anand Prakash
IDiAL Institute
Dortmund University of Applied Sciences and Arts
44227 Dortmund, Germany
anand.prakash@fh-dortmund.de

Lukas Krawczyk
IDiAL Institute
Dortmund University of Applied Sciences and Arts
44227 Dortmund, Germany
lukas.krawczyk@fh-dortmund.de

Carsten Wolff
IDiAL Institute
Dortmund University of Applied Sciences and Arts
44227 Dortmund, Germany
carsten.wolff@fh-dortmund.de

***Abstract*—The computational demands of safety-critical ADAS applications on autonomous vehicles have been ever-increasing. As a result, high performance computing nodes with varying operating frequencies, and inclusion of different sensors have been introduced, which has resulted in introduction of heterogeneous architectures with high complexity. This complexity has led to challenges in analyzing a system's timing behavior, such as determining the end-to-end response time of high-level functionality as well as a real-time application's latency. Although several approaches to tackle this issue have been proposed, their practical verification on real-life applications is still an open issue. Accordingly, this work proposes an automotive demonstrator that will be used in evaluating the timing behavior of ADAS applications in a real-life environment using methodologies such as tracing, profiling and static analysis. The APP4MC RaceCar is a work in progress four-wheel drive demonstrator built on a Traxxas 1/10 scale RC car platform. It is equipped with state-of-the-art sensors like LiDAR, ZED2 stereo camera and hosts multiple heterogeneous on-board computers such as Nvidia AGX Xavier to replicate a full size autonomous vehicle. In this paper, we describe the need for making such a demonstrator with an overview of the heterogeneous components used in it. Moreover, we further describe the system architecture as well as the data flow through event-chain task model for the ADAS application which is based on Waters Challenge 2019 industrial case study.**

***Index Terms*—Heterogeneous System, Radio-Controlled Cars, Electronic Speed Controller, RT-Linux Kernel.**

I. Introduction

From an abstract point of view, a typical Advanced Driver Assistance System (ADAS) has to perform three tasks - perception, planning, and control. As part of the perception task, various sensors are used in an ADAS application. The vehicle status is provided as a feedback, which along with the processed sensor data, plans and controls the path of an autonomous vehicle. Each of these tasks are computationally expensive, which requires high performance processing cores. At the same time, these applications are safety-critical in nature and must follow hard real-time constraints. Adaptive cruise control, anti-lock braking systems, lane keep assistance, obstacle detection/avoidance systems, and traffic sign recognition are just a few examples of an ADAS application with high computational requirements.

Automotive OEMs are gradually moving towards higher levels of driving automation, thereby increasing the complexity of these applications. The number of different sensor, such as LiDARs, stereo cameras, and Radars, installed on a vehicle has also increased significantly in recent times. This accounts for the huge amount of data from different sensors that needs to be processed in real time, which leads to a computational bottleneck. In order to deal with the computational bottleneck, heterogeneous platforms are used to improve the overall performance. Hardware accelerators such as GPUs and FPGA are used in co-ordination with CPUs to increase the computational power. Semiconductor companies provide various hardware platforms for such ADAS application. For example, Renesas R-Car-H3 [1] System-on-Chip (SoC) is a high performance platform specifically designed for In-Vehicle infotainment and driving safety support. The NVIDIA Drive [2] platform provides a range of developer kits for Autonomous vehicle along with sensor suite.

From the architecture point of view, an ADAS application can be divided into two categories. In a centralized computing architecture, the raw data from sensors is passed to a centrally located high performance computer that performs data processing. However, the cost, performance, and power requirements for such a processing unit are usually very high. Generally, the sensors have their own processing cores where the initial data is filtered and then sent to the main processing unit for further computation. This kind of distributed computing architecture is the more conventional approach. The application does not rely on a single main processing core,

which provides system redundancy for functional safety along with reduced processing requirements.

Each task in an ADAS application has different computational demands. For example, the computational requirement for an object detection algorithm from camera input is much higher than that of an object range detection from ultrasonic sensors. Therefore, in order to reduce the latency of the overall application, it becomes necessary to map these tasks to optimal processing cores. The latency of these applications can be further reduced by using parallel programming models such as MPI, CUDA, etc. These tasks should not only follow hard-real time constraints but must also be deterministic at the same time. Using a Real-Time Operating System (RTOS) for such applications provides a better control over the handling of tasks based on the scheduling and preemption model used in it.

Development of autonomous driving applications [3] poses several challenges in terms of timing analysis, efficient mapping and scheduling of tasks, and maintaining their deterministic behavior. These applications undergo rigorous testing and must pass all related safety standards such as Automotive Safety Integrity Level (ASIL) before going into production. An efficient way to conceptualize a new feature or a functionality is to implement it in early design phases on an RC (Radio-Controlled) Car model. A model based approach can be used to design a heterogeneous application on these platforms. This not only reduces the development cost and risk factor but also helps in determining the feasibility of applications. Besides, the RC platform provides the flexibility to evaluate and benchmark various performance metrics such as latency, end-to-end response time, memory contention, execution time, and so on. Since there are multiple sensors and ECUs interacting with each other over different communication interfaces, there is a possibility of communication interference adding to the latency which affects the overall performance of the application in real-life scenarios. These metrics can be analyzed, to further improve the software/hardware design of the system.

In this work, we design and implement an ADAS application on a 1/10 scale RC car. The demonstrator model will provide a platform to measure the response time of the implemented ADAS application, memory contention, latency caused due to communication interference in real-life environments. The remainder of the paper is structured as follows: Section II provides an overview of related work. Section III describes the existing challenges with respect to the timing analysis of an ADAS. Section IV gives an overview of the heterogeneous components and sensors used in the proposed demonstrator model. The system architecture is explained in Section V. Section VI provides a brief overview of evaluating the timing behavior based on the generated trace data. Finally, Section VII draws the conclusion and road map for the future work.

II. Related Work

Numerous RC cars have been developed for education and research purpose, with their complexity depending upon their particular use case and purpose. A typical RC car consists of a chassis, motor controller, actuators, processing units, sensors, and power supply. However, if a system is designed for a specific purpose such as parallel parking, the platform may consist of a minimum required components. Automatic Parallel Parking of RC Car platform [4] hosts a custom-made circuit board consisting of an IC chip, amplifier and radio receiver along with electric motors and antenna. It consists of a simple parallel parking application developed on a single IC chip with no requirement of high performance processing nodes. A more complex application would require a heterogeneous system. For instance, Duckietown [5] provides minimal autonomy and basic features such as lane following by utilizing e.g. a Raspberry Pi that is attached to a monocular camera. Another example of a low cost, low power autonomous robot can be found in Wolfbot[6] which is based on Beaglebone Black development platform. Even though a realistic computer vision pipeline is implemented on these platforms, the GPU capability available on the on-board computer is not utilized. At the same time, the sensor technology used in these platforms are not very advanced.

Higher the complexity of ADAS application, more is the computational requirement. Such applications require dedicated accelerators to process the data in real-time. The RC platform Go-CHART [7] makes use of external GPU capability to overcome the computational bottleneck. The sensor data is transmitted for further processing to Jetson TX2 board using wireless communication. For better performance and reliable ADAS application, it is recommended to have an on-board high performance computer on the RC platform. Several open source self-driving RC car platforms such as MuSHR [8], JetRacer [9], and Donkey Car [10] come with onboard Jetson Nano processors and advanced sensors. MuSHR has an additional Electronic Speed Controller (ESC) component, which provides a better control over the actuators. MuSHR, JetRacer, and Donkey Car are based on centralized compute architecture where there is only one processing node which is responsible for processing all sensor data as well as controlling the actuators. The performance of NVIDIA Jetson Nano is sufficient for applications involving computer vision algorithms. However, relying on a single processing node might not be efficient for an application having multiple tasks with real-time constraints.

Some autonomous miniature car models use two processing nodes, one with the capability of GPU is dedicated for executing machine learning algorithms and the other processing node for motor control. AutoRally [11] is a high-end RC car built on a 1/5-scale platform which uses Intel processing unit along with Nvidia GTX accelerator for scaled autonomous driving. It is based on distributed compute architecture where each processing node is responsible for a specific task. However, the RC car platform does not include a LiDAR sensor. Instead of hosting a stereo camera, it comes with two monocular cameras which makes the system more complex. MIT Racecar[12] platform houses state-of-the-art sensors and computing hardware, placed on top of a powerful 1/10-scale mini race car.

A model based approach can be taken in designing a system

based on Operator-Controller Module (OCM) [13] architecture. This helps in better analysis of the performance metrics at each architectural level. It is worth mentioning that none of the above mentioned RC mini cars platforms are based on the OCM architecture. The Industrial Waters Challenge 2019 [14] provides a case study on a prototypical ADAS application modelled on a heterogeneous platform using Amalthea [15]. The implementation of this model can be used to determine the performance metrics of a heterogeneous ADAS application. MIT Racecar platform satisfies the requirements for designing an ADAS application based on Waters Challenge 2019. However, its compute capabilities can be further enhanced by using next-generation and more powerful development boards. The heterogeneity of the system can be further enhanced by designing the system based on OCM architecture. This has led us to the development of a new RC car platform for applying the methods to measure the end-to-end latency of an ADAS application on a heterogeneous system.

III. Problem Statement

Researchers have come up with many novel solutions to analyze and benchmark the performance of ADAS application on the basis of several performance metrics. The existing work in this area can be categorized on the basis of architecture, high computation algorithms, sensor fusion, scheduling and mapping of tasks on processing cores. New functions in an ADAS require access to different communication interfaces, which makes the system more complex. AUTOSAR (AUTomotive Open System ARchitecture) is based on OSEK specifications, and provides a three layer architecture to develop an automotive application.

Since most ADAS applications rely on computer vision and image processing algorithms which require parallel processing, General Purpose GPUs (GP-GPUs) have started playing an important role. Parallel portions of an application are executed on GP-GPUs in terms of e.g. kernel programming model [16]. Therefore, the response time of an application can be determined by the execution time of a kernel [17] on a given computing unit. However, GPUs are proprietary systems, which restricts the knowledge of their internal working. This makes the prediction of latency caused due to GPUs uncertain. Another way to determine the execution time of a task on a GPU is presented in [18]. In recent times, reconfigurable platforms such as FPGAs are also being exploited as accelerators for image processing algorithms. PYNQ [19] is a FPGA based platform that hides the underlying hardware details and exposes a python interface to use any computer vision framework such as OpenCV [20].

Typically, an ADAS application consists of multiple tasks executing on dedicated cores. These programs consist of multiple event chain sequences where the input of next task depends on the previous task's output. Each task has its own execution time. Varying speed of processors must be taken into account while scheduling a task on that processor. At the same time, an optimum scheduling sequence of these tasks needs to be determined to make the overall application more efficient. The availability of a processor at a given point of time must be considered while scheduling these tasks. This makes scheduling of these tasks non-trivial. This can affect the hard real time constraint and the efficiency of the algorithm. For example, Wang et al. [21] provided an insight of how effective parallelism can improve the performance of a LDA (Lane Detection Algorithm) compared to a naive parallel approach. RTOS such as QNX Neutrino, SAFERTOS uses different scheduling techniques in executing the tasks, which affects the latency of the application [22].

Another major aspect of an ADAS application is efficient mapping of tasks to the processing units. The performance of an application is highly dependent on optimum utilization of the resources available on a heterogeneous platform. The Waters Challenge 2019 [14] focused on developing an initial model based on Amalthea [15] which can be further used in deriving the performance metrics of the application. It is also worth mentioning that the model is derived for a specific hardware platform (NVIDIA Jetson TX2 SoM). To further optimize the application, evolutionary optimization approaches such as genetic algorithm can be used for allocation of tasks [23]. In order to determine the worst case response time of an application, the event chain in the critical path must be considered. Tracing format such as BTF (Best Trace Format) [24] can be used to analyze the timing, performance, and reliability of the system. Similar approach can be used for mapping of tasks, calculation of the response time in real life environment for an application running on RC car platform.

IV. Overview of Heterogeneous Components

In order to precisely replicate the real life scenario, the demonstrator must be modeled similar to existing vehicles with autonomous functionality. Therefore, it has been designed with multiple sensors and processing units. The remainder of this section briefly describe the architecture of the components used in the system.

A. NVIDIA Jetson AGX Xavier

The NVIDIA Jetson AGX Xavier SoM [25] includes a compact carrier board and Jetson Xavier module. It is a powerful AI computer designed for autonomous machines. It provides performance to handle sensor fusion, localization and mapping, obstacle detection, and path planning algorithms critical for autonomous driving. A 40-pin expansion header supports some standard communication interfaces such as I2C, UART, SPI and CAN. A M.2 Key E slot can be used to add WiFi/LTE capability to the board. An overview of the components of the NVIDIA Jetson AGX Xavier is illustrated in Figure 1, followed by an in-depth description in the following subsections.

1) Processing Unit: The CPU complex (CCPLEX) is divided into four clusters. Each cluster contains two identical 64-bit Carmel processors that are compliant to ARM's v8.2 ISA architecture. A high performance System Coherency Fabric (SCF) connects all CPU clusters, thus enabling simultaneous

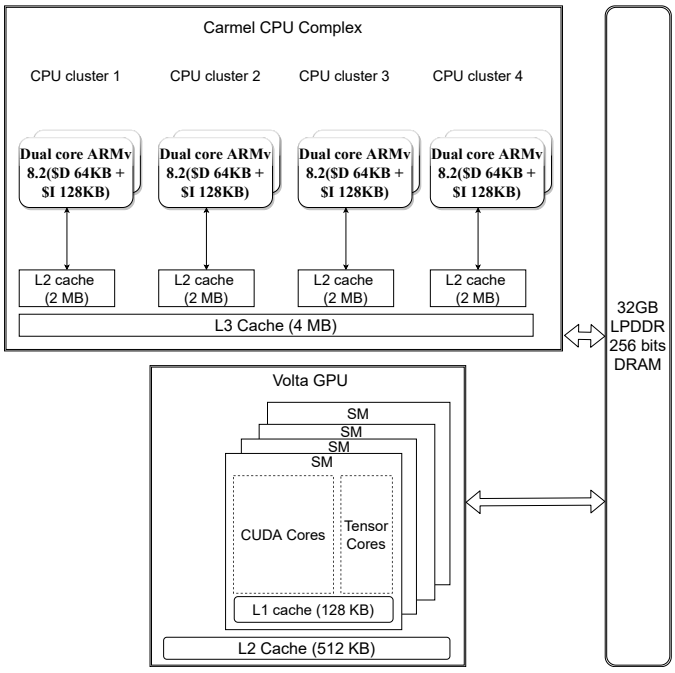

Fig. 1. NVIDIA Jetson AGX Xavier Architecture.

operation of all CPU cores for a true heterogeneous multi-processing (HMP) environment. The SCF also connects the CPU clusters to DRAM through a Memory Controller Fabric (MCF) and I/O blocks in the Memory Mapped I/O (MMIO) space through an ARM Advanced eXtensible Interface (AXI).

2) Hardware Accelerators: The NVIDIA GPU GV10B is based on Volta Architecture which features 512 shading units, 32 texture mapping units, and 16 Render output units (ROP). It also includes 64 tensor cores, which help to improve the speed of machine learning applications. Additionally, the Volta GPU architecture features a new Streaming Multiprocessor (SM) which allows an energy efficient, high performance computation of tasks that have processing requirements of large and complex data streams. Each SM is partitioned into four separate blocks referred as Streaming Multiprocessor Partitions(SMPs). Each SMP contains its own instruction buffer, scheduler, CUDA cores, and Tensor cores. GPUs core graphics functions are performed inside the Graphics Processing Cluster (GPC), which is a dedicated hardware block for computation, rasterization, shading, and texturing. It is also augmented with an image signal processor (ISP), a multimedia engine, a programmable vision accelerators (PVAs), and a pair of NVIDIA deep-learning accelerators (NVDLAs). These accelerators can be used in parallel or in conjunction with CPU and GPU cores.

3) Memory: The Xavier platform features a distributed shared memory architecture. It comes with a system memory of 32GB 256-Bit LPDDR4x and provides a eMMC storage of 32GB. CPU as well as GPU have direct access to system memory. Each CPU core includes 128 KB Instruction (I-cache) and 64 KB Data (D-cache) Level 1 caches, whereas a 2 MB L2 cache is shared by both cores in a single cluster. All clusters share a common 4MB L3 cache. Each SM in GPU has an additional L1 cache of 128KB as well as access to a common 512KB L2 cache that is shared by all SMs.

4) Speed: Each CPU can operate at a maximum frequency of 2265 MHz. The GPU is operating at a frequency of 854 MHz, which can be boosted up to 1377 MHz. The GPU provides workstation-class performance with up to 32 TeraOPS (TOPS) of peak compute and 750 Gbps of high-speed I/O. The maximum system memory bandwidth is 137GB/s providing a low latency in accessing it.

5) Power utilization: Jetson AGX Xavier enables new levels of power efficiency. Users can configure operating modes for their applications at 10W, 15W, or 30W.

B. Beaglebone AI

The Beaglebone AI [26] is used as secondary processing unit in our RaceCar platform. The board is built around a Texas Instruments (TI) AM5729 system-on-chip (SoC). Its computing capabilities enable the user to develop machine learning applications with ease. It supports all standard communication interfaces over an 46-pin header on either side of the board. Additionally, it also supports a 16-bit LCD interface. The board was developed specifically for AI applications and has an integrated neural engine that processes complex algorithms on the hardware level. The block diagram of AM5729 SoC is depicted in Figure 2.

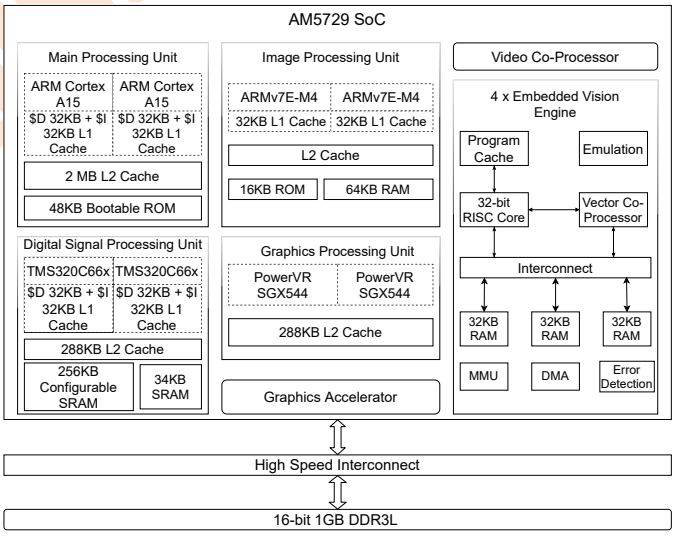

Fig. 2. AM5729 SoC Architecture.

1) Processing Unit: The Beaglebone AI comes with an ARM-A15 based dual-core processor. It supports the standard ARM instruction set with hardware virtualization support. Two dual-core Programmable Real-Time Units (PRU) are present to provide ultra low latency. Access to these PRUs are enabled via expansion headers. These dual-core PRUs are based on ARMv7E-M architecture for general purpose usage, particularly real-time control.

2) Hardware Accelerators: The platform comes with a 3D graphics processing unit (GPU) subsystem based on dual POWERVR SGX544 cores to support general embedded applications. The GPU can process different data types simultaneously, such as: pixel data, vertex data, video data, and general-purpose data. Additionally, the platform is also equipped with a Vivante based 2D graphics accelerator. There is a separate dedicated hardware for the machine learning libraries called Embedded Vision Engine (EVE) which functions as a programmable image and vision processing engine. Additionally, the platform also has two DSP subsystems for audio processing as well as general purpose image and video processing.

3) Memory: Similar to AGX platform, Beaglebone AI has a distributed shared memory architecture. It comes with a 16GB eMMC device and features a SD card slot. The system memory is 16-bit 1GB DDRL device. Each core of the main processing unit has a 32KB instruction and 32KB data L1 cache along with shared 2MB L2 cache. The unit also features a 48KB bootable ROM. Finally, the platform provides 32KB data and 32KB instruction L1 cache for the DSP unit, 32KB shared L1 cache memory for the PRUs, and a system level cache of 128KB for the GPUs.

4) Speed: The system memory operates at a frequency of 553MHz yielding an effective rate of 1066Mb/s on the DDR3L bus allowing for 4GB/s of DDR3L memory bandwidth. The main processing unit operates at a frequency of 1.5MHz, whereas the GPU operated at a maximum frequency of 532MHz.

C. Vedder Electronic Speed Controller

Most modern Electronic Speed Controller (ESC) consist of a microcontroller, which take input signals to regulate the speed of an electric motor. VESC [27] is an open source ESC that enables advanced customization options with multiple interface support. The VESC 6 MKIV uses STM32F4 microcontroller chip. It operates within the voltage range of 11.1V to 60V and provides a continuous current of 80A, which can reach up to 120A in burst mode. Moreover, it can read its 3D orientation in space, 3 axis acceleration values, and directions via the inbuilt Inertial Measurement Unit (IMU).

It is equipped with multiple communication interfaces and sensor ports. Hall sensors allow precise and powerful rotation of motor rotors from a random position. Single Wire Debug (SWD) provides an interface for debugging, diagnosis of real-time data on STM controller. Along with the connectors for Brushless DC (BLDC) motor and servo motor, it also provides standard interfaces such as I2C, UART which allows its integration with other micro-controllers such as the Beaglebone AI. An additional CAN Bus interface allows integrating multiple VESC devices into an array.

D. ZED2 Camera

Cameras serve as a crucial component in enabling machine vision and surroundings awareness. Based on the vision, camera can be classified as monocular vision or stereo vision. In automated driving, monocular vision camera can detect only the classified objects, whereas stereo cameras replicate human vision, thus allowing accurate extraction of depth information, such as the distance of a moving object.

The ZED2 [28] is a stereo camera that provides high definition 3D video and neural depth perception of the environment. It has been designed for a variety of challenging applications, such as autonomous navigation and mapping to augmented reality and 3D analytics. It supports video streaming with a maximum field of view of 120 degrees and a maximum resolution of 2.2K at 15 frames per second (fps). For applications which require higher fps, the camera can provide a maximum rate of 100 fps with the resolution getting compromised relatively. Any object within the depth range of 0.3m to 20m can be detected through the camera. It has a built-in Inertial Measurement Unit (IMU), barometric pressure sensor, and magnetic sensor, and can acquire inertial, elevation, and magnetic field data in real time.

ZED2 camera is compatible with NVIDIA GPU platforms. Therefore, the computation power of Jetson AGX platform can be leveraged in creating a real-time application.

E. Slamtech Lidar Sensor

Although cameras provides much of the sensing capabilities for an autonomous vehicle, they suffer various limitations when dealing with e.g. shadows or bright lights, which may cause confusion in taking decisions. Moreover, calculating an object's distance from raw images usually comes at high computation cost and requires correspondingly powerful computers. A viable solution to reduce these limitations is using other sensing technologies such as LiDAR or RADAR. The RaceCar uses the capability of LiDAR along with stereo camera for sensing the environment in decision-making.

A LiDAR uses lasers to sense the surrounding environment. The concept for distance determination remains similar to RADAR, where the distance is calculated based on the duration between the transmitted signal and the reflected signal received from the object. LiDARs have extremely fast response times which gives the processing units on the autonomous cars ample amount of time to react to the changing environment. One of its primary advantage is precision and accuracy.

The RPLIDAR A3M1 [29] used in RaceCar is the next generation low cost 360 degree 2D laser scanner (LIDAR) developed by SLAMTEC. It can take up to 16000 samples of laser ranging per second with high rotation speed. The system can perform 2D 360-degree scan within a 25-meter range. It must be noted that the distance range for dark or less reflective objects is limited to 10m. The generated 2D point cloud data can be used in mapping, localization, and object/environment modeling. The typical scanning frequency of LiDAR is 10Hz (600rpm), and the frequency can be freely adjusted within a range from 5 to 20Hz according to the specific requirements. With the 10Hz scanning frequency, the sampling rate is 16kHz and the angular resolution is 0.225 degree. It is worth mentioning that it provides a rotation speed detection and adaptive system as it adjusts the angular

resolution automatically according to the actual rotating speed. The LiDAR is augmented by a DSP unit which takes the input from the vision acquisition system, processes the sampled data and provides the output distance, angle values between the object and LiDAR.

The RPLiDAR A3M1 can either be operated in *enhanced mode* or *outdoor mode*. The enhanced mode is meant for indoor environments and provides greater performance compared to outdoor mode, whereas the outdoor mode comes at an increased reliability. The RPLiDAR needs a 5V supply for powering the range scanner core and motor system.

V. System Architecture

The system architecture is coarsely based on the concept of Operator-Controller-Module (OCM) [13]. The OCM architecture helps in realization of a self-optimizing complex system with adaptive behavior over changing environmental conditions. It can be structured into three levels - Controller, Reflective, and Cognitive Operator. The block diagram in Figure 3 depicts the overall system architecture. The VESC acts as a controller operator as it has direct access to the actuators and operates under hard-real time conditions. The Beaglebone AI acts as Reflective operator. It receives the information about motor speed, steer, and acceleration value from the VESC. It does not have direct access to the actuators, but regulates and supervises the VESC. At the same time, it acts a communication interface between the VESC and NVIDIA Jetson AGX Xavier. The Jetson AGX Xavier board is part of Cognitive Operator as it deals with processing the sensor data such that it adapts to the changing environment conditions.

A. Hardware Architecture

The Traxxas chassis comes with a built-in ESC. However, for the purpose of better control and customization, the built-in ESC is replaced with the VESC.

a) Power Management: The platform consists of two power sources - 3S LiPo battery and power bank. The LiPo battery is dedicated to provide power supply to VESC using XT-90 connector. The Jetson AGX Xavier board is powered with a 19V power supply from Patona power bank. An active USB is also connected to the power bank. The active USB hub consists of four USB ports, one is used to power up Beaglebone AI. The power supply to RPLiDAR is also provided by the USB hub.

b) Sensing: The sensing technology installed on the platform consists of ZED2 stereo camera and RPLiDAR A3M1. Both the sensors provide a USB interface and are directly connected to Jetson AGX board. Additionally, the VESC as well as ZED2 camera comes with an in-built IMU sensor which can be used to determine the orientation, acceleration of the RC car.

c) Actuators: The Traxxas platform already comes with a high-speed performance Velineon brushless DC motor. The DC motor has a 3.5mm bullet connection interface to integrate

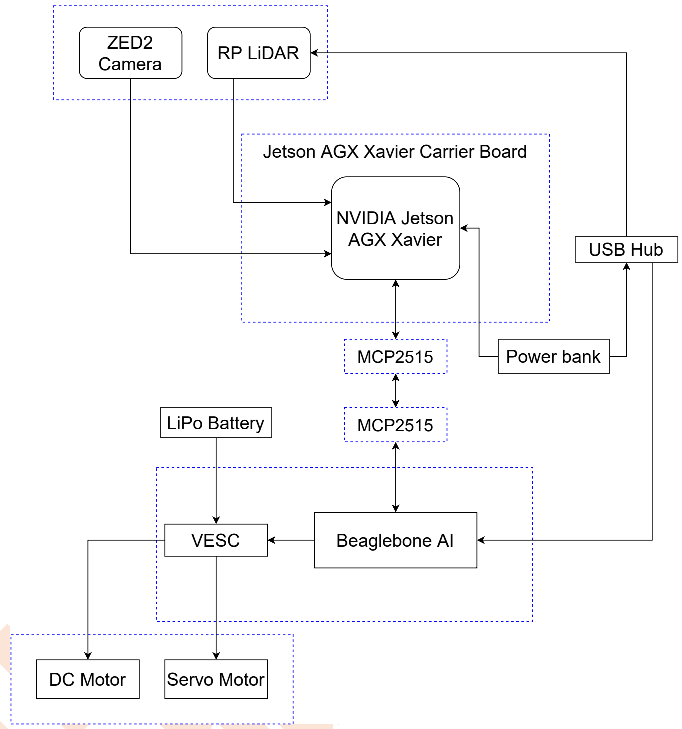

Fig. 3. APP4MC RaceCar Block diagram

with any standard ESC. Additionally, it also provides a high-torque digital steering servo with Futaba connectors. The Traxxas 2075 steering servo has a transit time of 0.16 seconds which delivers a responsive steering.

d) Communication Interfaces: Our demonstrator uses some standard communication protocols to interact with different components on the RaceCar. The data transfer between the Beaglebone AI and VESC must be in full-duplex mode. Since the amount of data exchanged between them is not very high, a UART interface with a baud rate of 115200 is sufficient for the application. CAN interface is one of the standard protocol used in automotive applications. The data transfer between the Beaglebone AI and VESC must also be in full-duplex mode. A CAN bus over SPI interface is established between them for data exchange. For this purpose, MCP2515 breakout board is used, which provides a CAN Bus transceiver over SPI interface.

B. Software Architecture

A modular software stack for individual components is depicted in Figure 4.

1) Jetson AGX Xavier Software Stack: To address the real-time constraints of the application we have enabled RT-Linux to the NVIDIA Jetson Xavier platform. This signifies that among all the threads ready for execution, the one with the highest priority will be executed. The Linux kernel provides two real-time scheduling policies (`SCHED_FIFO` and `SCHED_RR`) that apply an individual arbitration in case of tasks having same priorities. Non-real-time tasks are scheduled following the `SCHED_NORMAL` policy.

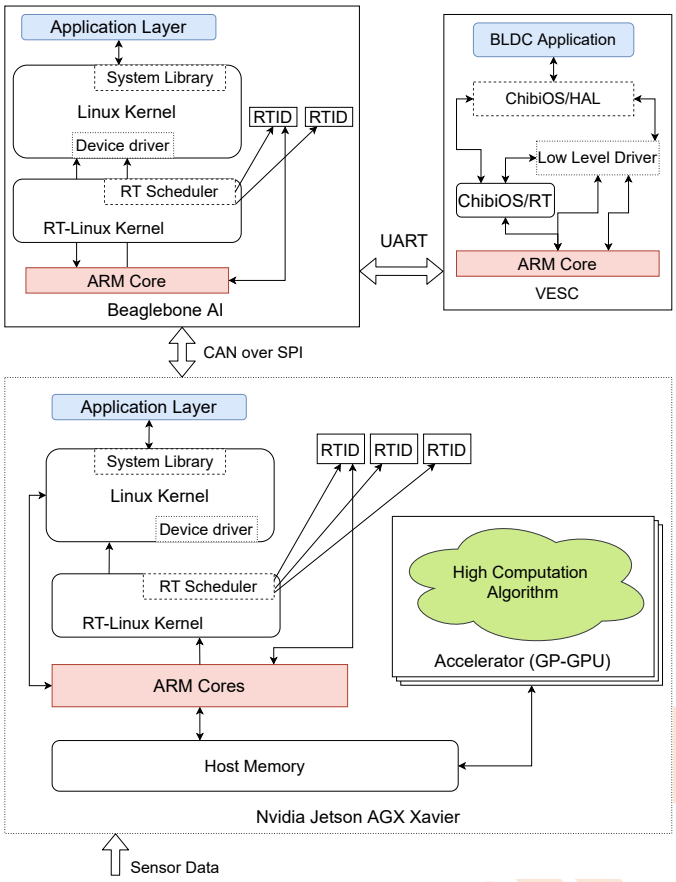

Fig. 4. APP4MC RaceCar Software Architecture

The application is implemented on top of the Operating System (OS) layer, further broken into several tasks, and mapped on different cores. Those tasks that have high computational demands are offloaded to GP-GPUs. CPUs apply a fully preemptive fixed priority scheduling policy, whereas GPUs follow weighted round-robin scheduling. The system memory is shared between the CPU cluster and GP-GPUs for better performance. Figure 5 depicts the task model along with the data flow from sensors to actuators. The task definition of ADAS application defined in this paper is mainly derived from the Waters Challenge 2019 [14].

- *Localization* - The localization task is responsible for determining the relative position of the RC car on a given environmental map. It takes the point cloud data from LiDAR input and merges it with the RC car motion status to estimate the demonstrator's position.
- *Can Polling* - This task gets the key information about the demonstrator motion parameters from the on-board CAN bus and sends it to the Localization and Planner task.
- *Structure From Motion* - This task is responsible for estimating the depth of an object based on the stereo vision camera images. The distance of the object is passed to Planner task for further processing.
- *Lane Detection* - This task provides accurate locations of the road boundaries and the shape of each lane. The output of this task is a matrix of points representing the lane boundaries within the road, which is sent to the Planner task.
- *Detection* - The detection task is responsible for detecting and classifying the objects within the visual range of the camera. The output of this task is sent to the Planner task.
- *Planner* - The main purpose of this component is to calculate and follow a vehicle trajectory. The targeted vehicle motion parameters are passed to the CAN Controller task.
- *Car Controller* - The main purpose of this task is to get the steering angle, speed, and acceleration value from the Planner task and provide it to the reflective operator over CAN Bus.

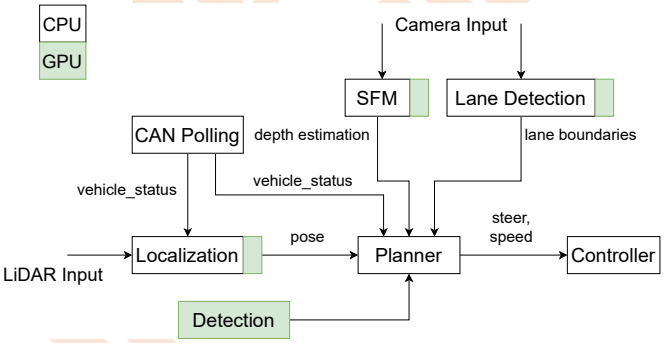

Fig. 5. APP4MC RaceCar Task Model with dataflow

2) Beaglebone AI Software Stack: The RT-Linux Kernel has been ported on Beaglebone AI.

The application layer consists of two threads - one interacting with VESC to configure and send the commands to it, the other to communicate with Jetson AGX Xavier board. The Beaglebone AI provides the feedback regarding the current speed, steering angle, acceleration, and orientation of the RaceCar to Jetson AGX Xavier board over the CAN bus.

3) VESC Application: As the VESC directly interacts with the actuators, it must also conform to the real-time constraints required for controlling the actuators. The VESC firmware is built using RTOS ChibiOS. It is a light weight operating system providing deterministic behavior of real-time multi-threaded applications. The scheduling of threads on VESC is possible in two ways - Round Robin scheduling and Cooperative Scheduling.

A brushless DC (BLDC) motor application is implemented on top of ChibiOS RTOS. This application receives commands from Beaglebone AI over UART interface and performs the respective operation.

VI. Timing Analysis using Tracing Framework

Tracing the software application is one of the efficient approaches in determining its timing behavior. BTF [24] is a CSV (Comma-Separated Values) based format used in

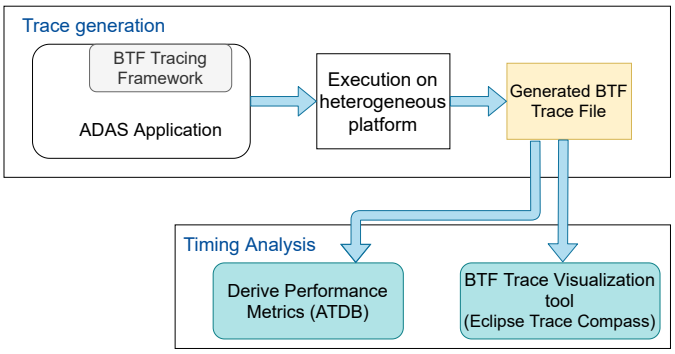

Fig. 6. Timing Analysis Workflow using BTF Trace.

recording events that are triggered on entities in a chronological order, on a system level. The integrated BTF tracing framework can be utilized to capture the events generated on each task at runtime in real-life scenarios. An overview of the workflow for timing analysis of the application using tracing capabilities is illustrated in Figure 6. The generated trace file can be viewed on any standard BTF trace visualization tool, for example Eclipse Trace Compass [30]. The timing performance metrics can be derived from the generated trace file by converting it to the Eclipse APP4MC Amalthea Trace Database (ATDB) [31] format. The ATDB file determines the execution time of each task and runnables which includes the average, best-case and worst-case execution time on a specific core. The event-chain metrics in ATDB provides the latency of all the event-chain tasks in the application. At the same time, the trace data also provides the information about the resource utilization of the processing unit, thereby assisting in efficient mapping of the tasks on processing cores.

VII. Future Work and Conclusion

The paper describes the state-of-the-art work on RC car platform and identifies the need for developing a new demonstrator. The APP4MC RaceCar provides a practical prototype of a full size autonomous vehicle with its heterogeneous architecture and sensing capabilities. The paper briefly describes the architecture of the heterogeneous components and sensor used in our platform. It discusses the system architecture and the data flow event-chain task model for an ADAS application on a heterogeneous platform. The computing capabilities of GP-GPUs can be used to implement and test an ADAS application in a real-life environment.

Future work involves working on the mechanical design and assembly of the RaceCar components. Further implementation involves integrating each component as well as developing the Amalthea task model based on the described architecture. In addition, porting a deterministic RTOS such as QNX on Beaglebone AI and Jetson AGX Xavier will further enhance the real-time capability of the system. Finally, we implement a BTF tracing framework that allows us to use the demonstrator to verify timing analysis results and efficient mapping of tasks on the processing nodes.

References

[1] Renesas r-car-h3. [Online]. Available: https://www.renesas.com/jp/en/products/automotive-products/automotive-system-chips-socs/r-car-h3-m3-starter-kit

[2] Nvidia drive. [Online]. Available: https://developer.nvidia.com/drive

[3] R. Okuda, Y. Kajiwara, and K. Terashima, "A survey of technical trend of adas and autonomous driving," in *Technical Papers of 2014 International Symposium on VLSI Design, Automation and Test*, 2014, pp. 1–4.

[4] B. Xiao, C. Xu, and L. Xu, "Notice of violation of ieee publication principles: Automatic parallel parking of rc car using distance sensors," in *2009 Second International Conference on Future Information Technology and Management Engineering*, 2009, pp. 525–528.

[5] L. Paull, J. Tani, H. Ahn, J. Alonso-Mora, L. Carlone, M. Cap, Y. F. Chen, C. Choi, J. Dusek, Y. Fang, D. Hoehener, S. Liu, M. Novitzky, I. F. Okuyama, J. Pazis, G. Rosman, V. Varricchio, H. Wang, D. Yershov, H. Zhao, M. Benjamin, C. Carr, M. Zuber, S. Karaman, E. Frazzoli, D. Del Vecchio, D. Rus, J. How, J. Leonard, and A. Censi, "Duckietown: An open, inexpensive and flexible platform for autonomy education and research," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 1497–1504.

[6] J. Betthauser, D. Benavides, J. Schornick, N. O'Hara, J. Patel, J. Cole, and E. Lobaton, "Wolfbot: A distributed mobile sensing platform for research and education," in *Proceedings of the 2014 Zone 1 Conference of the American Society for Engineering Education*, 2014, pp. 1–8.

[7] S. Kannapiran and S. Berman, "Go-chart: A miniature remotely accessible self-driving car robot," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 2265–2272.

[8] S. S. Srinivasa, P. Lancaster, J. Michalove, M. Schmittle, C. Summers, M. Rockett, J. R. Smith, S. Choudhury, C. Mavrogiannis, and F. Sadeghi, "Mushr: A low-cost, open-source robotic racecar for education and research," 2019.

[9] JETRACER. [Online]. Available: https://github.com/NVIDIA-AI-IOT/jetracer

[10] DONKEYCAR. [Online]. Available: https://www.hackster.io/wallarug/donkey-car-with-jetson-nano-robo-hat-mm1-e53e21

[11] B. Goldfain, P. Drews, C. You, M. Barulic, O. Velev, P. Tsiotras, and J. M. Rehg, "Autorally: An open platform for aggressive autonomous driving," *IEEE Control Systems Magazine*, vol. 39, no. 1, pp. 26–55, 2019.

[12] S. Karaman, A. Anders, M. Boulet, J. Connor, K. Gregson, W. Guerra, O. Guldner, M. Mohamoud, B. Plancher, R. Shin, and J. Vivilecchia, "Project-based, collaborative, algorithmic robotics for high school students: Programming self-driving race cars at mit," in *2017 IEEE Integrated STEM Education Conference (ISEC)*, 2017, pp. 195–203.

[13] J. Gausemeier, U. Frank, J. Donoth, and S. Kahl, "Specification technique for the description of self-optimizing mechatronic systems," *Research in Engineering Design*, vol. 20, pp. 201–223, 11 2009.

[14] F. Wurst, D. Dasari, A. Hamann, D. Ziegenbein, I. Saudo, N. Capodieci, M. Bertogna, and P. Burgio, "System performance modelling of heterogeneous hw platforms: An automated driving case study," in *2019 22nd Euromicro Conference on Digital System Design (DSD)*, 2019, pp. 365–372.

[15] Eclipse APP4MC. [Online]. Available: https://www.eclipse.org/app4mc/

[16] D. A. Jamsek, "Designing and optimizing compute kernels on nvidia gpus," in *2009 Asia and South Pacific Design Automation Conference*, 2009, pp. 224–229.

[17] R. Saussard, B. Bouzid, M. Vasiliu, and R. Reynaud, "Optimal performance prediction of adas algorithms on embedded parallel architectures," in *2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems*, 2015, pp. 213–218.

[18] C. Widerspick, W. Bauer, and D. Fey, "Latency measurements for an emulation platform on autonomous driving platform nvidia drive px2," in *ARCS Workshop 2018; 31th International Conference on Architecture of Computing Systems*, 2018, pp. 1–8.

[19] K. Haeublein, W. Brueckner, S. Vaas, S. Rachuj, M. Reichenbach, and D. Fey, "Utilizing pynq for accelerating image processing functions in adas applications," in *ARCS Workshop 2019; 32nd International Conference on Architecture of Computing Systems*, 2019, pp. 1–8.

[20] OpenCV. [Online]. Available: https://docs.opencv.org/master/index.html
[21] X. Wang, M. Cui, K. Huang, A. Knoll, and L. Chen, "Improving the performance of adas application in heterogeneous context: A case of lane detection," in *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, 2017, pp. 1–6.
[22] M. Hammond, G. Qu, and O. A. Rawashdeh, "Deploying and scheduling vision based advanced driver assistance systems (adas) on heterogeneous multicore embedded platform," *2015 Ninth International Conference on Frontier of Computer Science and Technology*, pp. 172–177, 2015.
[23] L. Krawczyk, M. Bazzal, R. P. Govindarajan, and C. Wolff, "An analytical approach for calculating end-to-end response times in autonomous driving applications," 06 2019.
[24] V. I. GmbH, "Best trace format (btf)technical specification v2.2.0," 2020.
[25] NVIDIA Jetson AGX Xavier. [Online]. Available: https://www.nvidia.com/de-de/autonomous-machines/embedded-systems/jetson-agx-xavier/
[26] Beaglebone AI System Reference Manual. [Online]. Available: https://github.com/beagleboard/beaglebone-ai/wiki/System-Reference-Manual
[27] Vedder Electronic Speed COntroller. [Online]. Available: https://vesc-project.com/
[28] ZED2 Camera Specifications. [Online]. Available: https://cdn.stereolabs.com/assets/datasheets/zed2-camera-datasheet.pdf
[29] RPLIDAR A3. [Online]. Available: https://www.slamtec.com/en/Lidar/A3
[30] Eclipse trace compass. [Online]. Available: https://www.eclipse.org/tracecompass/
[31] Eclipse app4mc amalthea trace database. [Online]. Available: https://www.eclipse.org/app4mc/help/latest/index.html#section4.9