A Method for Reducing Simulation Timing Deviation in QEMU-Based Virtual ECU

Anna Yang
SDV Center
DRIMAES, Korea Aerospace Univ.
Seoul, South Korea
metamon@drimaes.com

Woo Hyun Seol
SDV Center
DRIMAES
Seoul, South Korea
truman@drimaes.com

Hyng Rae Kim
School of Electronics and Engineering
Kyungpook National University
Daegu, South Korea
hrsin95@knu.ac.kr

Jin Yong Kim
SDV Center
DRIMAES
Seoul, South Korea
red.kim@drimaes.com

Woo Jin Han
SDV Center
DRIMAES
Seoul, South Korea
wjhan@drimaes.com

Jeonghun Cho
School of Electronics and Engineering
Kyungpook National University
Daegu, South Korea
jcho@knu.ac.kr

***Abstract*— As support for a wider range of functions and services of vehicles, such as driver assistance systems, autonomous driving, and OTA services, has increased in recent years, the complexity of software embedded in vehicles has become increasingly complex. These changes are driving demand for simulation technologies that can help reduce development and validation time while ensuring vehicle safety and reliability. Among them, electronic control unit virtualization technology is steadily being researched and commercialized by many companies because it can consider the characteristics of ECUs from the development and validation stages of in-vehicle embedded software. In this paper, we discuss the virtualization method of STM32F407ZGT using QEMU as part of ECU virtualization technology. Furthermore, we propose a technique to reduce the simulation error by measuring the running time of the physical ECU and adjusting the unit time in the AUTOSAR OS on the virtual ECU. This is to reduce the simulation error caused by the dependence on the host computer environment and the generation of overhead in the case of highly computational in-vehicle embedded software, which have been pointed out as problems in ECU virtualization using QEMU. The experimental results of the proposed technique show that the simulation timing error rate of the virtual ECU compared to the physical ECU is reduced from an average of 2.40% to 0.02%. This shows that, just by reflecting the operational characteristics of the physical ECU in the QEMU-based virtual ECU, it is possible to improve the precision by reducing the timing error of the simulation.**

Keywords—virtual ECU, Simulation, QEMU, AUTOSAR, Operation Accuracy

I. INTRODUCTION

Recently, the automotive industry has witnessed significant advancements, with the integration of technologies such as ADAS(Advanced Driver Assist System), AD (Autonomous Driving), RDE(Real Driving Emission), and driver convenience systems into vehicles. These advancements have led to a significant increase in the number of E/E(Electrical/Electronic) systems integrated into vehicles, thereby escalating the overall system complexity. According to a survey on the productivity improvement of development compared to system complexity for in-vehicle embedded systems, it was found that the productivity of vehicle development increased by 25 to 35% when the vehicle system complexity increased fourfold from 2010 to 2020 [1]. This means that more manpower and time are required than ever before to ensure the safety and reliability of the vehicle during the development and validation stages. Therefore, measures to increase productivity in the development and validation stages are needed.

The development and validation processes for in-vehicle embedded systems following the traditional V-cycle require long lead times in the development and validation stages. In particular, when electronic control units (ECUs) and software are developed in parallel to meet vehicle launch schedules, it can be time-consuming and costly to solve the problems arising during the stage of integrating them. As a way of addressing this issue, methods have emerged to virtualize ECUs and perform simulations using virtual ECUs to validate systems with fast cycles in the early stage of development. Virtual ECUs allow for fast testing without the need to prepare physical ECU samples. Moreover, tests under complex conditions that are difficult to implement in real-world tests can be performed through simulations [2].

We have been conducting research on how to virtualize and simulate ECUs using QEMU(Quick EMUlator) to increase the efficiency of the development and validation stages of in-vehicle embedded systems. QEMU can be used to virtualize ECUs based on software implementation of specific hardware. In the absence of a physical ECU, using QEMU for ECU virtualization enables simultaneous development and validation of in-vehicle embedded systems, resulting in time and cost efficiency. However, since QEMU-based ECU virtualization is performed by software implementation of not only the ECU but also the connectivity of peripheral devices connected to the ECU, there is a problem that the precision of the operation time is reduced depending on the performance and resource utilization of the host personal computer (Host PC) [3][4]. In the case of in-vehicle embedded systems, real-time performance must be guaranteed, so it is also necessary to increase the precision of the operation time in simulation.

In this paper, we propose a technique to improve the precision of virtual ECU operation time by virtualizing ECU using QEMU and reflecting physical ECU operation time. In the case of ECU virtualization, we virtualize a Cortex-M4-based STM32F407ZGT [5] and run a software package developed according to AUTOSAR (AUTomotive Open System Architecture) Classic, a representative standard for in-vehicle embedded software. To improve the operation time precision of the virtual ECU, we propose a method to improve the operation time precision of the virtual ECU by adjusting the unit time of the AUTOSAR OS running on the virtual ECU based on the operation speed of the real ECU and show the results of the implementation.

In this paper Section 2 describes the results of previous studies, and Section 3 describes the ECU virtualization method for to run AUTOSAR. Section 4 describes the technique for improving the operation time precision of virtual ECUs and shows its implementation results, and Section 5 provides the conclusion.

II. Previous works

AUTOSAR is an organization that aims to develop and establish an open standard software architecture for ECUs installed in vehicles [6]. The main platforms are AUTOSAR Classic and AUTOSAR Adaptive. The former is an embedded real-time standard based on OSEK (Offene Systeme und deren Schnittstellen für die Elektronik in Kraftfahrzeugen), and the latter utilizes an operating system based on the POSIX standard. In this paper, only AUTOSAR Classic will be discussed. Fig. 1 roughly shows the architecture of AUTOSAR Classic.

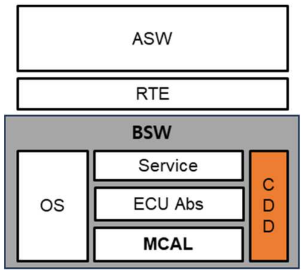


Fig. 1. Architecture of AUTOSAR Classic Platform

ASW (Application Software) refers to the various applications defined by the OEM. For the RTE (Runtime Environment), a VFB(Virtual Function Bus) is provided to enable independent operation between ASW and BSW (Basic Software). BSW consists of Service Layer, ECU Abstraction Layer, MCAL (Microcontroller Abstraction Layer), and CDD (Complex Device Driver) as the core structure of AUTOSAR. Each layer of BSW operates independently of each other and exists to pass data up or down based on each layer. The hierarchical division of AUTOSAR plays a major role in increasing reusability and development efficiency for software. For AUTOSAR Classic, development is performed by generating ARXML(AutosaR XML), the configuration information for in-vehicle embedded systems, utilizing ARXML to generate source code for ASW and BSW, and then generating executable.

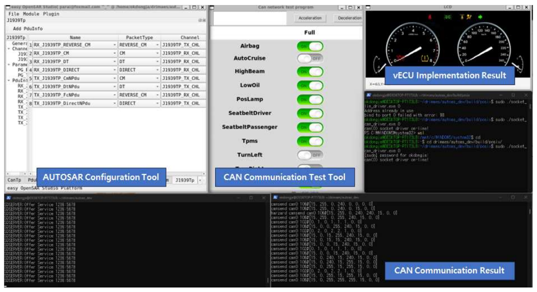


Fig. 2. Result of previous works

We have been conducting research on how to develop AUTOSAR Classic software that runs on a virtual ECU based on open software [7]. As a result, we have been able to generate source code for AUTOSAR Classic based on ARXML generated by the ARXML Configuration Tool [8] and generate it as an executable file. Fig. 2 shows the AUTOSAR Classic software running on a general-purpose virtual ECU provided by QEMU as a result of the research and development. We have also developed a CAN communication tool [9][10] to facilitate CAN communication tests of the virtual ECU. In previous works, experiments and implementations were conducted targeting the versatile board provided by QEMU to verify the operation of AUTOSAR Classic, but the versatile board of QEMU has the drawback of poor usability in the automotive industry. Therefore, virtualization will be performed for STM32F4W07ZGT based on Cortex-M4, which is highly usable in the automotive field, and AUTOSAR Classic will be operated and tested on the virtualized STM32F407ZGT. In the case of AUTOSAR Classic, only the MCAL of the BSW has dependencies on the ECU, so existing AUTOSAR Classic's MCAL must be implemented appropriately based on the STM32F407 to enable operation on the physical ECU. The developed AUTOSAR Classic software is then utilized as the software to run on the virtual ECU in Section 3. Fig. 3 shows the result of modifying the MCAL to run on the STM32F407 and running the above developed AUTOSAR Classic software on the physical ECU [11].

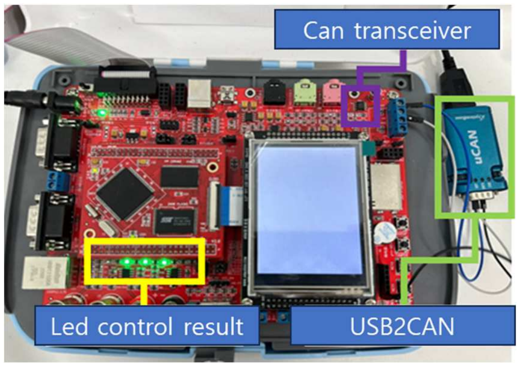


Fig. 3. The result of modification of MCAL on real ECU (STM32F407ZGT6)

III. Virtualization of STM32F407 For AUTOSAR

A. *Architecture of QEMU for ECU virtualization*

QEMU is a type of virtualization software, which has the characteristic that the entire software stack created for device types other than x86 can be executed on a virtual machine [12].

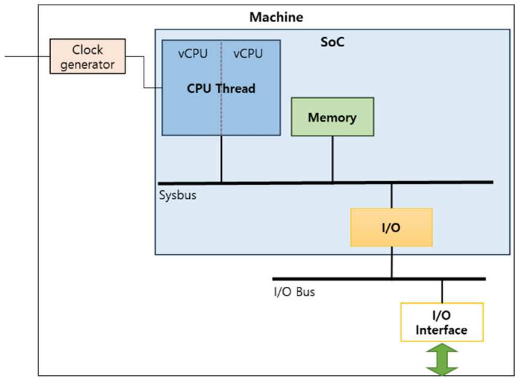


Fig. 4. Conceptual architecture of QEMU

Fig. 4 shows the architecture of the virtual ECU for implementing the STM32F407ZGT as a virtual ECU based on

QEMU. The Machine of the virtual ECU supports clock generation of the virtual ECU's SoC (System-on-Chip) and CPU, and supports data exchange between the Host PC and the virtual ECU through the I/O BUS and I/O interface. SoC is composed of vCPU, Memory, Sysbus, and I/O. The vCPU supports a binary translator that can run the code of the virtual ECU on the Host PC through the TCG Tiny Code Generator), and Memory consists of RAM and Flash. Sysbus is a bus that allows vCPU and I/O to access Memory. I/O has the role of simulating the operation of various peripherals supported by the virtual ECU and managing registers corresponding to the peripherals.

B. *Virtualization of STM32F407*

STM32F407ZGT6 is a development board that has a Cortex-M4 32-bit RISC core with ARMV7 architecture, and it supports CAN communication, which is the most widely used in automotive communication networks. To implement the STM32F407ZGT6 as a virtual ECU using QEMU, it is necessary to implement the Machine and SoC corresponding to the STM32F407ZGT6.

1) *Implemantion of Machine*

First, a Machine is added in the configuration, as shown in Table 1, so that the STM32F407ZGT can be used on QEMU. Based on this, the STM32F407ZGT can be selected as a virtual ECU when running QEMU, and the operations shown in Table 2 are performed by calling stm32f-407zg.c when the STM32F407ZGT is selected.

TABLE I. CONFIGURATON FOR MACHINE

Add machine: hw/arm/meson.build

```
arrm_ss.add(when: 'CONFIG_STM32F_407ZG', if_true: files('stm32f-407zg.c'))
```

Map SoC into machine: hw/arm/Kconfig

```
config STM32F_407ZG
   bool
   select STM32F405_SOC
```

Enable machine: configs/devices/Kconfig

```
CONFIG_STM32F_407ZG=y
```

Table 2 shows the implementation of the Machine's functionality. In stm32f_407zg_machine_init, ARM_CPU_TYPE_NAME specifies the type of CPU to use in the SoC. Stm32f_407zg_init is for assigning the clock generated by clock_new to the SoC using qdev_connect_clock_in. Sysbus_realize_and_unref is for creating Sysbus, and armv7m_load_kernel is for loading the firmware.

TABLE II. IMPLEMENTATION OF MACHINE

Implementation : stm32f-407zg.c

```
static void stm32f_407zg_init(MachineState *machine)
{
   Clock *sysclk;
/* Create clk */
   sysclk = clock_new(OBJECT(machine), "SYSCLK");
/* Set SoC */
   dev = qdev_new(TYPE_STM32F405_SOC);
   …
/* Set clk for SOC */
   qdev_connect_clock_in(dev, "sysclk", sysclk);
/* Create & Init Sysbus to connect various devices */
   sysbus_realize_and_unref(SYS_BUS_DEVICE(dev), &error_fatal);

/* Load firmware*/
   armv7m_load_kernel(ARM_CPU(first_cpu),                    machine->kernel_filename, 0, FLASH_SIZE);
}
…
static void stm32f_407zg_machine_init(MachineClass *mc)
{
  /* Call function for Init */
   mc->init = stm32f_407zg_init;
  /* CPU type */
   mc->default_cpu_type = ARM_CPU_TYPE_NAME("cortex-m4");
…
}
```

2) *Implementation of SoC*

Since QEMU does not support the STM32F407, we use the SoC of the STM32F405, which has the most similar architecture. Therefore, when running STM32F407ZGT, STM32F405_soc.c is executed as the SoC. However, it can have the same architecture as STM32F407 if additional peripherals not supported by STM32F405 are registered. Table 3 shows the configuration for this, in which the STM32F4XX_CAN option is added to support the CAN communication of STM32F407.

TABLE III. CONFIGURATON FOR THE ADDING SOC

Add SoC: hw/arm/meson.build

```
arrm_ss.add(when:     'CONFIG_STM32F_405_SOC',     if_true: files('stm32f-405_soc.c'))
```

Map devices into SoC: hw/arm/Kconfig

```
config STM32F405_SOC
          bool
          select ARM_V7M
          select OR_IRQ
          select STM32F4XX_SYSCFG
          select STM32F4XX_EXTI
          select STM32F4XX_GPIO
          select STM32F4XX_CAN
```

However, since STM32F405 and STM32F407 have different system configurations, it is necessary to register the status information for the supported peripherals, as shown in Table 4, so that the status information for each device can be updated based on the system information of STM32F407. In this paper, to support the scenario for improving the timing accuracy of virtual ECU operation using the physical ECU's synchronous timing, it is necessary to support an additional LED. Therefore, STM32F4XXGpioState is added to indicate the status of GPIO. Furthermore, STM32F4XXCanState is added to update the status information of CAN, and CanBusState is added to check the status of CAN communication bus.

TABLE IV. IMPLEMENTATION OF SOC

hw/arm/stm32f405_soc.h

```
struct STM32F405State {
   STM32F4XXGpioState gpio[STM_NUM_GPIOS];
   STM32F4XXCanState can[STM_NUM_CANS];
   CanBusState *canbus[STM_NUM_CANS];
};
```

When QEMU is executed, the peripheral list is registered using the information in TypeInfo. Then, when the Machine is executed, the peripheral devices registered in the Machine are initialized through class_init, and the instances of the devices are created and initialized through instance_init. Stm32f405_soc_realize uses memory_region_init_xxx, memory_region and add_subregion to initialize various memories and allocate memory hierarchically. DEVICE()

contains information about the peripherals added at the time of SoC addition. The peripherals can access specific addresses on the sysbus through sysbus_mmio_map. Sysbus_connect_irq sets the IRQ (Interrupt Request) so that a specific peripheral can receive interrupts on the Sysbus. In order for a device registered in the virtual ECU to communicate with the outside world, information about the interface must be registered in stm32f405_soc_properties, so CAN BUS was added through DEFINE_PROP_LINK.

3) *Implementation results*

Figure 5 shows results of running the AUTOSAR software package for LED control on the physical STM32F407ZGT6 and the virtual ECU. The virtual ECU was run in the environment shown in Table 5.

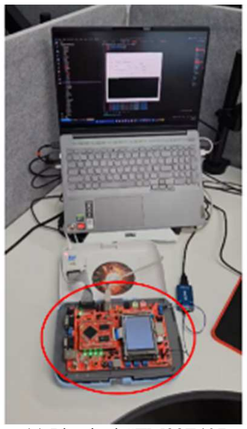
TABLE V. IMPLEMENTATION OF SOC

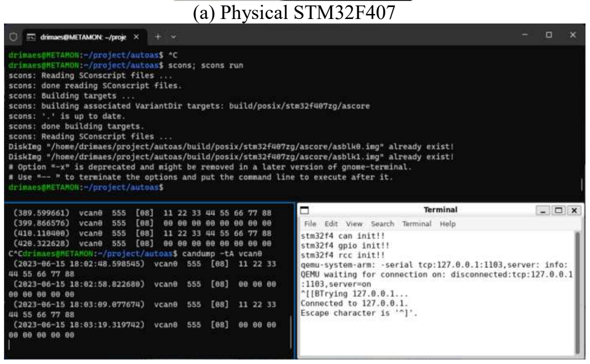
hw/arm/stm32f405_soc.c

```
static const TypeInfo stm32f405_soc_info = {
    …
    .instance_init = stm32f405_soc_initfn,
    .class_init    = stm32f405_soc_class_init,
};
…
static void stm32f405_soc_realize(DeviceState *dev_soc, Error **errp)
{
    STM32F405State *s = STM32F405_SOC(dev_soc);
    MemoryRegion *system_memory = get_system_memory();
    DeviceState *dev, *armv7m;
    SysBusDevice *busdev;

    /* Configure clock */
    clock_set_mul_div(s->refclk, 8, 1);
    clock_set_source(s->refclk, s->sysclk);

/************* [Memory allocation] *************/
/* Flash Memory & RAM & Cache*/
    memory_region_init_rom(&s->flash, OBJECT(dev_soc),
"STM32F405.flash",FLASH_SIZE, &err);
    memory_region_add_subregion(system_memory,
FLASH_BASE_ADDRESS, &s->flash);
…
/************* [CPU] *************/
    …
/************* [Peripheral] *************/
    …
    /* GPIO devices */
    for (i = 0; i < STM_NUM_GPIOS; i++) {
        busdev = SYS_BUS_DEVICE(dev);
        sysbus_mmio_map(busdev, 0, gpio_addr[i]);
        sysbus_connect_irq(busdev, 0, s->gpio[i].irq);
    }
    /* Attach CAN and CAN controllers */
    for (i = 0; i < STM_NUM_CANS; i++) {
        dev = DEVICE(&(s->can[i]));
        busdev = SYS_BUS_DEVICE(dev);
        sysbus_mmio_map(busdev, 0, can_addr[i]);
            sysbus_connect_irq(busdev, 0, qdev_get_gpio_in(armv7m,
can_irq[i]));
    }
}
/* External Interface */
static Property stm32f405_soc_properties[] = {
    DEFINE_PROP_STRING("cpu-type", STM32F405State, cpu_type),
    DEFINE_PROP_LINK("canbus0", STM32F405State, canbus[0],
TYPE_CAN_BUS, CanBusState *),
    DEFINE_PROP_LINK("canbus1", STM32F405State, canbus[1],
TYPE_CAN_BUS, CanBusState *),
    DEFINE_PROP_END_OF_LIST(),
};
```


(a) Physical STM32F407



```
drimaes@METAMON:~/project/autoas$ ^C
drimaes@METAMON:~/project/autoas$ scons; scons run
scons: Reading SConscript files ...
scons: done reading SConscript files.
scons: Building targets ...
scons: building associated VariantDir targets: build/posix/stm32f407zg/ascore
scons: '.' is up to date.
scons: done building targets.
scons: Reading SConscript files ...
DiskImg "/home/drimaes/project/autoas/build/posix/stm32f407zg/ascore/asblk0.img" already exist!
DiskImg "/home/drimaes/project/autoas/build/posix/stm32f407zg/ascore/asblk1.img" already exist!
# Option "-x" is deprecated and might be removed in a later version of gnome-terminal.
# Use "-- " to terminate the options and put the command line to execute after it.
drimaes@METAMON:~/project/autoas$
```

```
 (389.599661)  vcan0  555  [08]  11 22 33 44 55 66 77 88
 (399.866576)  vcan0  555  [08]  00 00 00 00 00 00 00 00
 (410.110400)  vcan0  555  [08]  11 22 33 44 55 66 77 88
 (420.322628)  vcan0  555  [08]  00 00 00 00 00 00 00 00
C^Cdrimaes@METAMON:~/project/autoas$ candump -tA vcan0
 (2023-06-15 18:02:48.598545)  vcan0  555  [08]  11 22 33
44 55 66 77 88
 (2023-06-15 18:02:58.822680)  vcan0  555  [08]  00 00 00
00 00 00 00 00
 (2023-06-15 18:03:09.077674)  vcan0  555  [08]  11 22 33
44 55 66 77 88
 (2023-06-15 18:03:19.319742)  vcan0  555  [08]  00 00 00
00 00 00 00 00
```

```
stm32f4 can init!!
stm32f4 gpio init!!
stm32f4 rcc init!!
qemu-system-arm: -serial tcp:127.0.0.1:1103,server: info:
QEMU waiting for connection on: disconnected:tcp:127.0.0.1
:1103,server=on
^[[BTrying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
```

(b) Virtual STM32F407

Fig. 5. Operation of AUTOSAR-based Application on STM32F407-based Target Board (a) real STM32F407;(b) virtual STM32F407

TABLE V. IMPLEMENTATION OF SOC

PC	**Processor** : Intel® Core™ i7-1160G7 @ 1.20GHz 2.11 GHz
	Memory : 16GB
	OS : Windows 11, x64

IV. ENHANCING OPERATIONAL ACCURACY FOR V ECU

A. *Scenario for Measuring Operation Time Error Between Physical and Virtual ECUs*

As a scenario to check whether there is an error in the precision of the operation time between the physical ECU and the virtual ECU, a CAN message is sent to the Host PC every time the state of each LED changes while turning on/off four LEDs every 200ms. The difference between the time when the CAN message arrives (t_{curr}) and the time when the previous message was sent (t_{prev}) is the time it takes for the ECU to turn on/off the LED, which can be considered as the ECU operation time. It is repeated over a number of times (N) to calculate the average operation time of the ECU (ECU_{avg}). Here, $\mathrm{rECU_{avg}}$ denotes the average operation time of the physical ECU, and $\mathrm{vECU_{avg}}$ denotes the average operation time of the virtual ECU. Therefore, the error rate ($deviation$) of the operation time between the physical ECU and the virtual ECU can be obtained as shown in Eq. (1). If the deviation is less than 0, it means that the virtual ECU is operating slower than the physical ECU, and if it is greater than 0, it means that the virtual ECU is operating faster than the physical ECU.

$$deviation(\%) = \left(1 - \frac{\mathrm{rECU_{avg}}}{\mathrm{vECU_{avg}}}\right) * 100 \qquad (1)$$

$$ECU_{avg} = \frac{\Sigma\,(t_{curr} - t_{prev})}{N}$$
t_{curr} : $Timestamp\ of\ the\ current\ message(ms)$
t_{prev}: $Timestamp\ of\ the\ previous\ message$(ms)
ECU_{avg} : $Average\ ECU\ operation\ time\ (m/s)$
N : $Total\ number\ of\ transmitted\ can\ message$

Table 6 shows the experimental results, and we can see that, in general, the virtual ECU operates slower than the physical ECU. This confirms that there is a temporal error in the operation of ECU between the virtual and physical environments. These errors can also be influenced by the availability of the Host PC or the complexity of the application running on the ECU.

TABLE VI. RESULTS OF COMPARING THE TIME BASED ON TRANSMISSION IN PHYSICAL AND VIRTUAL ENVIRONMENTS (BEFORE CALIBRATION, N=30)

Set	$vECU$	$rECU$	$deviation$(%)
1	2055.00	1995.05	-2.98
2	2052.97	1995.00	-2.91
3	2046.13	1995.43	-2.54
4	2041.43	1997.10	-2.22
5	2041.50	1997.6	-2.20
6	2039.60	1996.57	-2.16
7	2039.23	1995.50	-2.19
8	2039.20	1995.57	-2.12
9	2036.63	1994.43	-2.20
10	2038.27	1994.43	-2.37
Average(ms)	2043.00	1995.71	-2.40

B. Method to Improve the Operation Time Precision of Virtual ECU by Reflecting Physical ECU's Operation Time

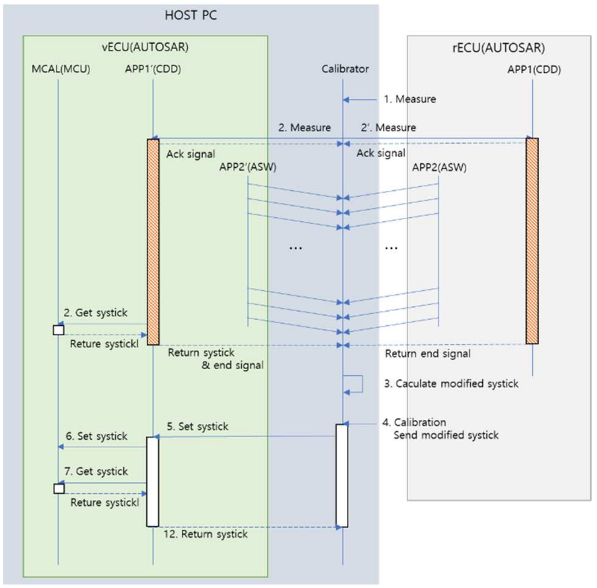


Fig. 6. Operation sequence diagram for correcting errors in the virtual ECU by reflecting the physical ECU

In this paper, to improve the operation accuracy of the virtual ECU based on the physical ECU, we propose a method of modifying systick, the unit time of ECU, by implementing a separate app in the AUTOSAR CDD area, as shown in Fig. 6. To improve the accuracy of the virtual ECU based on the same operation on the virtual ECU and the physical ECU, we ran same AUTOSAR software on the virtual ECU and the physical ECU. APP1 is a Systick calibration app that runs in the CDD area. When a measurement start command is sent from the outside, the ACK message is returned to account for potential variations in data transmission time($T_{\text{communication}}$ and $T_{real\_communication}$) by the communication circuit's configuration. Subsequently, the Systick value ($systick\_val$) set in the individual ECU with a measurement termination signal after a specified duration (T_{send} and T_{real_send}) from the relevant timestamp (T_{receive} and $T_{real\_receive}$).

As for the AUTOSAR ASW, we ran the LED application provided, which is APP2 in the diagram. Calibrator is a separate PC application that measures the operation time of the virtual ECU $((T_{\text{receive}} - T_{\text{send}}) - T_{\text{communication}})$ and that of the physical ECU $(T_{real\_receive} - T_{real\_send}) - T_{real\_communication})$. It calculates the calibrated Systick ($cal\_systick\_val$) using Eq. (2) and delivers it to the virtual ECU.

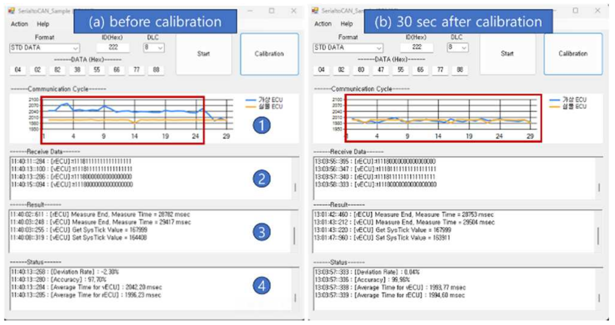
$$cal_{systick_{val}} =$$

$$\frac{((T_{real\_receive} - T_{real\_send}) - T_{real\_communication})}{((T_{\text{receive}} - T_{\text{send}}) - T_{\text{communication}})} * systick\_val \qquad (1)$$

$systick\_val$: Virtual ECU's current systick value
T_{receive}: Time at which the test result of the virtual ECU CDD module is received
T_{send} : Time at which the virtual ECU CDD module transmitted the measurement signal
$T_{\text{communication}}$: Time delayed due to communication of the virtual ECU CDD module
$T_{real\_receive}$: Time at which the physical ECU CDD module received the test result
T_{real_send}: Time at which the physical ECU CDD module transmitted the measurement signal
$T_{real\_communication}$: Time delayed due to communication of the physical ECU CDD module

In Fig. 7, (a) and (b) show the results before and after ECU's systick calibration, respectively. In the figure, ① shows a graph of the communication cycle, which is the result of CAN communication based on AUTOSAR ASW of the virtual ECU (blue) and the physical ECU (orange). The x-axis represents the number of CAN messages received by Calibrator, and the y-axis represents $t_{curr} - t_{prev}$. ② displays the timestamps along with the messages received from the virtual ECU and physical ECU. ③ displays the step-by-step progress of the operation to correct the time error of the virtual ECU by utilizing the physical ECU. ④ displays the error of the operation time between the virtual ECU and the physical ECU in real-time. In the Communication Cycle Graph in Fig. 9, the communication error gap between the ECUs is reduced in (b) than in (a).

In Table 7, the operation time error rate between the physical ECU and virtual ECUs was calculated after performing the calibration based on the same method as Table 6. It shows that the error rate of 2.40% before performing the calibration was reduced to 0.02%.



```
------Receive Data------
11:40:11::284 : [rECU]:t1118111111111111111
11:40:13::100 : [vECU]:t1118111111111111111
11:40:13::286 : [rECU]:t1118000000000000000
11:40:15::094 : [vECU]:t1118000000000000000

------Result------
11:40:02::611 : [rECU] Measure End, Measure Time = 28782 msec
11:40:03::248 : [vECU] Measure End, Measure Time = 29417 msec
11:40:03::255 : [vECU] Get SysTick Value = 167999
11:40:08::319 : [vECU] Set SysTick Value = 164408

------Status------
11:40:13::268 : [Deviation Rate] : -2.30%
11:40:13::280 : [Accuracy] : 97.70%
11:40:13::284 : [Average Time for vECU] : 2042.20 msec
11:40:13::285 : [Average Time for rECU] : 1996.23 msec
```

```
------Receive Data------
13:03:55::395 : [rECU]:t1118000000000000000
13:03:56::347 : [vECU]:t1118111111111111111
13:03:57::340 : [rECU]:t1118111111111111111
13:03:58::333 : [vECU]:t1118000000000000000

------Result------
13:01:42::460 : [rECU] Measure End, Measure Time = 28753 msec
13:01:43::212 : [vECU] Measure End, Measure Time = 29504 msec
13:01:43::220 : [vECU] Get SysTick Value = 167999
13:01:47::960 : [vECU] Set SysTick Value = 163911

------Status------
13:03:57::333 : [Deviation Rate] : 0.04%
13:03:57::336 : [Accuracy] : 99.96%
13:03:57::338 : [Average Time for vECU] : 1993.77 msec
13:03:57::339 : [Average Time for rECU] : 1994.60 msec
```

Fig. 7. Calibrator execution screens: (a)before calibration; (b)30 sec after calibration

TABLE VII. RESULTS OF COMPARING THE TIME BASED ON TRANSMISSION IN PHYSICAL AND VIRTUAL ENVIRONMENTS (AFTER CALIBRATION, N=30)

Set	$\boldsymbol{vECU}$	$\boldsymbol{rECU}$	$\boldsymbol{deviation}$(%)
1	1996.57	1996.03	-0.03
2	2000.60	1995.47	-0.26
3	1988.50	1994.97	0.18
4	1999.73	1996.60	-0.16
5	1992.23	1995.50	0.16
6	1994.27	1997.07	0.14
7	1995.70	1995.47	0.01
8	1996.53	1993.93	-0.13
9	1997.63	1993.93	-0.19
10	1993.87	1995.53	0.08
Average(ms)	1995.56	1995.45	0.02

V. CONCLUSION

In this paper, we conducted research on virtualizing STM32F407ZGT6 and reducing operational discrepancies in both virtual and physical environments using AUTOSAR. We demonstrated the addition of the machine and SoC in QEMU for virtualizing STM32F407ZGT6, as well as the inclusion of various peripherals. As a result, we confirmed that the newly implemented STM32F407ZGT6 is capable of running AUTOSAR Classic and CAN communication. Moreover, by adjusting the virtual ECU's systick based on the operational time of the physical and virtual ECUs, we were able to reduce the temporal discrepancies in the operation. This approach overcomes the limitations of traditional QEMU, which performs complete virtualization of the ECU, by incorporating the characteristics of the physical hardware to calibrate and mitigate potential errors. Additionally, in a mixed environment utilizing both virtual and physical ECUs, we expect to enhance the precision of communication based on the ECU's operation, enabling more accurate validation of vehicle embedded systems. In the future, we plan to conduct further research on adjusting the systick to achieve hardware-independent operation in virtual ECUs, simulating similar time discrepancies as hardware operation based on the workload of the App, even in the absence of physical ECUs.

ACKNOWLEDGMENT

This work was supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIP) (No.1711160343, Development of virtual ECU-based vehicle-level integrated simulation technology for vehicle ECU application software development and verification automation)

REFERENCES

[1] O. Burkacky, J. Deichmann, D. Hepp, S. Frank, and A. Rocha, “When code is king: Mastering automotive software excellence,” 17-Feb-2021. [Online]. Available: https://www.mckinsey.com/industries/automotive-and-assembly/our-insights/when-code-is-king-mastering-automotive-software-excellence?cid=eml-web

[2] T. Kim. "Vehicle Test and Validation in Virtual Environment" AUTO JOURNAL : Journal of the Korean Society of Automotive Engineers 40, 8 (2018) : 66-68.

[3] R. Misbin and A. George, “QEMU-Based Emulation-in-the-Loop for the Simulation of Small Satellite Flight Software,” presented at the - 2023 IEEE Aerospace Conference, 2023, pp. 1–8, doi: 10.1109/AERO55745.2023.10115569.

[4] S. B. Oh and J. H. Kim, “An Analysis on Interrupt Latency of Hypervisor for Automotive Software Integration,” vol. 30, no. 11. The Korean Society of Automotive Engineers, pp. 901–907, 2022

[5] STM32F407ZGT User Manual V1.0 ,” 2012 [Online]. Available: https://kazus.ru/forums/attachment.php?attachmentid=40217&d=1352233087

[6] AUTOSAR [Online]. Abailable: https://ko.wikipedia.org/wiki/AUTOSAR

[7] AUTOAS [Online]. Available: https://github.com/autoas/as

[8] J. H. Lee, W. J. Han, A. Yang, “A Designing of Automotive Embedded Software for Virtual ECU” , The Korean Institute of Communications and Information Sciences Winter Conference 2023, Pyeongchang, GangWon, South Korea, 2023, pp. 313-313.

[9] J. S. Kong, W. J. Han, A. Yang, “A Utilization Automotive Embedded Software for Virtual ECU”, The Korean Institute of Communications and Information Sciences Winter Conference 2023, Pyeongchang, GangWon, South Korea, 2023, pp. 315 - 316

[10] I. H. Lee, W. J. Han, A. Yang, “A Structure of CAN/CAN-FD for Automotive Embedded Software based on Virtual ECU” , The Korean Institute of Communications and Information Sciences Winter Conference 2023, Pyeongchang, GangWon, South Korea, 2023, pp. 317-318

[11] J. S. Kong, W. J. Han, A. Yang, “Implementation of MCAL CAN for Applying Validated AUTOSAR in a Virtual Environment to STM32F” , The Korean Institute of Communications and Information Sciences Summer Conference 2023, Jeju, South Korea, 2023

[12] WiKi, “QEMU.” [Online]. Available: https://ko.wikipedia.org/w/index.php?title=QEMU&oldid=34819587.