International Journal of Modern Electronics and Communication Engineering (IJMECE) ISSN: 2321-2152
Volume No.-7, Issue No.-3, May, 2019

Memory and Runtime Optimization for Automotive Embedded Systems

[1]K. M. Shweta
Department of Electronics and Communication Engineering
Bangalore Institute of Technology
Bangalore, India
kmshweta52@gmail.com

[2]J. C. Narayanaswamy
Department of Electronics and Communication Engineering
Bangalore Institute of Technology
Bangalore, India
jcnswamy@gmail.com

Abstract: Automotive Embedded System consists of ECU (Electronic Control Unit), the heart of the system and number of microcontrollers, actuators, sensors and other peripheral devices. The PCM (Power-train Control Module) is a type of ECU, consists of TC1793 microcontroller, focused in this project for optimization. As advances in Automotive Embedded System increases, it demands the addition of new features and functionalities in the software. The microcontroller's memory and runtime start to take a hit and its optimization becomes prevalent. In this paper, an attempt is made to optimize the load on both the CPU and memory by using various optimizing techniques, inturn improves the efficiency of the microcontroller. The major challenge is to maintain the runtime of the CPU below 92%, in spite of enhance in the memory.

Keywords: ECU, Memory optimization, PCM, RAM, TC1793 Microcontroller.

I. INTRODUCTION

In today's automotive embedded systems, ECU's (Electronic Control Unit) play a vital role in controlling the various functions of a vehicle. There are number of ECU's present in today's cars, with up to 50 - 60 ECU's in high end cars. Various functions can be achieved using these ECU's such as engine control, transmission control, brake control, speed control, infotainment etc. ECU's are controlled with the help of microcontrollers to perform multiple tasks. The 32 bit TC1793 [1] is the microcontroller used in this project. In the ECU software development cycle, there are repeated changes to the software set due to many reasons, such as addition of new functionalities, fixing the bugs in earlier versions of the software, and making the code more compliant with industry standards.[2] The PCM ECU being used in this project has reached its maximum RAM, Flash and Runtime limits due to already existing features such as Flex Fuel algorithm,[3] Gasoline Particulate Filter (GPF)control, UniAir intake valve control and many more. Upon adding new features to the software, the resources start to exceed its limit and its optimization becomes inevitable. In this project, addition of new functionalities in the software such as Advanced Driver Assistance Systems (ADAS), Flashing over the Air (FOTA) and Cyber security leads to unavailability of RAM and Flash as the memory is already full and this leads to incorporate various optimization techniques.

Here the resource optimization is broadly divided into two categories; Memory optimization and Runtime optimization. Memory optimization is performed in RAM and Flash memories. The memory optimization is performed by reducing the size of variables, measuring points (measuring points are local variables which displays intermediate values of the logic at any point in the c code in real-time) and calibrations in the code and also by moving rarely used functions or object files from RAM to other memory like SPRAM (Scratch Pad RAM). The runtime optimization is more focussed towards reducing the total load on the CPU which is achieved by moving some functions within the different memories of the microcontroller.

The 32 bit TC1793 microcontroller used here is tricore memory which has broadly 3 sections of RAM namely DMI (Direct or Data Memory Interface), LMU (Local Memory Unit), PMI (Program Memory Interface) or SPRAM (Scratch Pad RAM). [1]

DMI (Data memory Interface): This is the section of RAM relatively closer to the CPU hence providing runtime benefits for frequently used functions. For our convenience it is considered as RAM0 where usually the frequently accessed central services/functions are placed.

LMU (Local Memory Unit): This is the section of RAM which is connected to the CPU via a data bus and hence slower than RAM0. The less frequently used variables are stored here for runtime optimization methods. For our convenience it is considered as RAM1.

SPRAM (Scratch Pad RAM): The SPRAM can store both the variables and the functions/code and it is closest to the CPU. Very frequently used functions/code/variables are stored in this section for most runtime benefits.

II. LITREATURE REVIEW

[1]Describes in detail, TC1793 microcontroller architecture and different memories [2], [3] Detailed explanation of development and working of PCM ECU. [7] Explains about the generation of Hex file and its testing onto microcontroller. [6], [8], [9] explains about the newly added features for this software.

III. METHODOLOGY

A. Hex File generation

The hex file is generated for testing and to verify whether the software is working properly when it is dumped in the ECU.

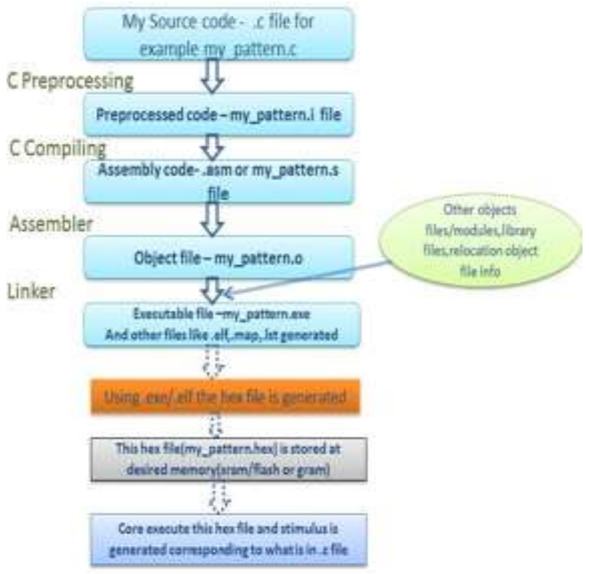

My Source code - .c file for example my_pattern.c
C Preprocessing
Preprocessed code – my_pattern.i file
C Compiling
Assembly code- .asm or my_pattern.s file
Assembler
Object file – my_pattern.o
Other objects files/modules,library files,relocation object file info
Linker
Executable file –my_pattern.exe And other files like .elf,.map,.lst generated
Using .exe/.elf the hex file is generated
This hex file(my_pattern.hex) is stored at desired memory(sram/flash or gram)
Core execute this hex file and stimulus is generated corresponding to what is in .c file

Fig 1: Build stages

Before generating the final hex file software undergoes different build stages;

1) Core processes stage: In this stage constants are checked and all the processes are checked for its execution. If any of the processes are missing then will get build error.
2) Damos stage: In this stage all the .xml files are checked whether it contains all the variables and calibrations and it will check the export and import of variables and it also checks for variable definition properties and declaration of variables.
3) Compiler stage: In this stage all the .c files are converted to object files ie .o files. .c files are checked for the coding guide lines, syntax errors, semantic errors, lexical errors.
4) Linker stage: In this stage one or more object files generated from the compiler are taken and generate the executable file. In this stage invocator file starts its execution and changes related to memory are made in these files. The linking of the object files, memory allocation is done at this stage.
5) A2L stage: An A2L description file contains all the information from the files generated from the .xml files related to variables, calibrations and constants, their size and what data types they are using such as parameters, distinctive maps and curves, not imaginary and implicit measurement variables alternate dependencies. The organized computer data is necessary for the following objects, such as storage structure, memory address, data type and conversion rules for converting them into physical units.
6) Hexmod stage: In this stage the final hex file will be generated. A hex file is formatted in a way to store the hexadecimal information of machine language code. These hex files are dumped into the controllers using programmers to test the working of software.

B. RTI File generation

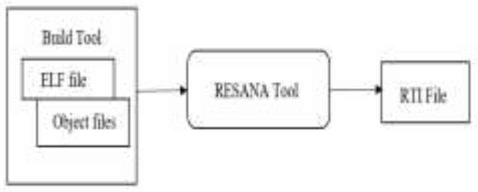


Fig 2: Generation of RTI file

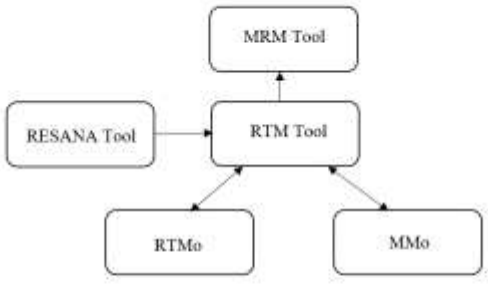


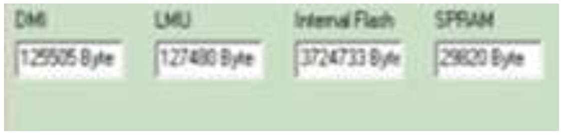
Fig 3: Loading RTI file to tools to check memory consumption

The RTI file is generated using the RESANA tool by giving the entire software as input. (RESANA stands for Resource Analysis. It consolidates all the variables, constants, calibrations, measuring points and functions in a single report and also gives the information regarding the memory addresses and size of each of the entities, thus letting the user analyze the static resource consumption of the microcontroller.) RTI file contains the information about all the functions, variables and their memory addresses. It also contains the information of the size of each variable, functions and calibrations so as to facilitate the tracking location of each variable in different sections of the RAM and hence the overall memory statistics. The RTI file is loaded to the MRM tool (MRM stands for Memory and Runtime measurement tool. This tool supports project responsible by tracking the ECU resource consumption.)to get the overall memory consumption of different memory sections in the microcontroller. This information regarding the memory usage is used to identify the functions/variables and measuring points which occupy large chunks of memory. These are then moved to the slower sections of memory or removed depending on their use.

C. Optimization techniques to reduce the load on CPU and memory

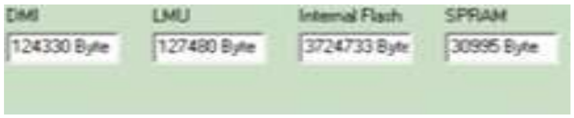
1) *Shifting the measuring points to SPRAM which benefits to ~1.17kB of RAM.*

Measuring points are local variables which displays intermediate values of the logic at any point in the c code in real-time. Since these measuring points are only used for tapping the value of variable at any particular point in the logic and are not used for the calculation of any variable, they are assigned a separate memory section (.mpram). These measuring points present in the software consumes approximately 2kb of total RAM (DMI). Few measuring points present in DMI part of the memory, moved to scratchpad RAM resulting in ~1.17Kb reduction of DMI RAM and since it is closer to the CPU, DMI memory can be utilized for other frequently used functions/variables. The RTI file before and after modifying the software is loaded to RTM tool (RTM Tool stands for Runtime Measurement Tool which is used to measure and analyse the runtime of various functions/processes present in the software) to verify the reduction in memory consumption.



DMI	LMU	Internal Flash	SPRAM
125505 Byte	127480 Byte	3724733 Byte	29820 Byte

Fig 4: Before Modifying

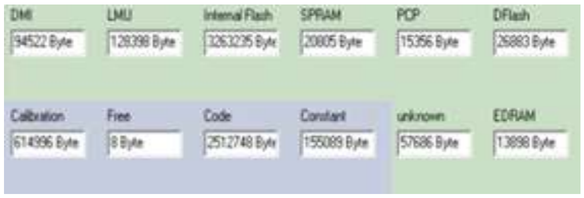


DMI	LMU	Internal Flash	SPRAM
124330 Byte	127480 Byte	3724733 Byte	30995 Byte

Fig 5: After Modifying

2) *Removal of unused functions from the code*

There are certain functions that exist in the software, but are not called/referenced often by the CPU as they might be bounded to a different system conditions. For example a diesel system or a Gasoline system. These functions are identified and are removed from the FLASH, freeing up more memory. By loading the RTI file into RTM tool, all the nonreferenced functions are identified, with the information about their location and amount of memory consumption in the software. The RTI file before and after modifying the software is loaded to RTM tool to verify the reduction in memory consumption.



DMI	LMU	Internal Flash	SPRAM	PCP	DFlash
94522 Byte	128398 Byte	3263235 Byte	20805 Byte	15356 Byte	26883 Byte

Calibration	Free	Code	Constant	unknown	EDRAM
614996 Byte	8 Byte	2512748 Byte	155089 Byte	57686 Byte	13898 Byte

Fig 6: Before Modifying

Fig 7: After Modifying

3) *DDDI array size reduction benefits to ~8kB of RAM.*

Universal Diagnostic Service protocol provides a number of necessary functionalities for repairers, developers and testers so that they can, read or write data in ECU memory, program the flash memory and create specific behavior for an ECU such as provide a response on a timer interrupt event. [5]
It provides different services:

Read Data By Identifier: This service provides to fetch data of one or more values of a control unit. The current state of the sensor which is a dynamic value can be known. Every value is associated to a Data Identifier (DID) between the range of 0 and 65535. The information of DID is sent only on request, and is for information that no ECU uses, but it is helpful for service tool or a software tester. [5]

Read Data By Identifier Periodic: With this service values are sent periodically by a control unit. The values to be sent must be defined to only using the "Dynamically Define Data Identifier". [5]

DDDI: This service allows the client to dynamically define the data in the server, a data identifier can be read via Read Data by Identifier service at a later time. The intention of this service is to provide the client with the ability to group one or more data elements into a data superset that can be requested in mass via the Read Data by Identifier or Read data by periodic identifier service. The data elements to be grouped together can either be referenced by;

- The position, source DID, size (in bytes)
- Address of memory and its length
- Combinations of the two above methods listed above through multiple requests.

In this software to provide DDDI service, RAM uses 16K Bytes and it is reduced to 8K Bytes to save up more memory. Therefore by reducing the memory used by DDDI service to 8K Bytes, the RAM reduction is achieved. The overall usage of DDDI is reduced from 16K Bytes to 8K Bytes in the configuration file, resulting in the reduction of RAM usage to 8K Bytes. After loading RTI to MRM tool the changes are reflected as below.

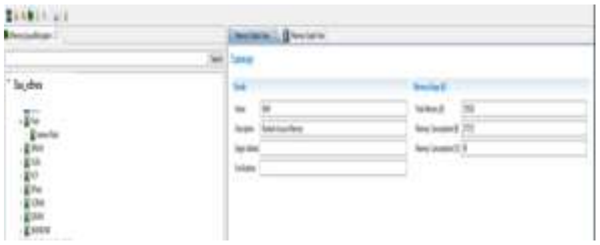


Fig 8: Before Modifying

Fig 9: After Modifying

4) Runtime Optimization

The runtime is the total load on the CPU and in this microcontroller, used in our project; the runtime limit is capped to 92% to prevent the ECU resets during vehicle usage, by the end user. [4] However with the addition of more and more functionalities, the runtime in our project reached 98% and thus following measures are taken to reduce the runtime to below 92%.

1. Most frequently used code and data that is the variables are identified and are

placed in the closest memory to the CPU. SPRAM for code and DMI for data.

2. TC1793 microcontroller uses starting 16K Bytes of RAM, as it uses absolute addressing method in which the program explicitly names the address that is going to be used and hence the most frequently used variables/functions are identified and are placed in the initial 16K Bytes of the DMI RAM. [1]
3. During the normal working of the system, many variables are being written and exported to other modules/functions in the software, for them to read the variable for their calculation. All such data exchanges are identified, by moving the processes to the slower process raster and by allocating the code for the export and import function in the same code area, so as to facilitate optimum runtime consumption with increasing the efficiency of microcontroller. In the below graph the runtime is reduced below 92% from 98%.

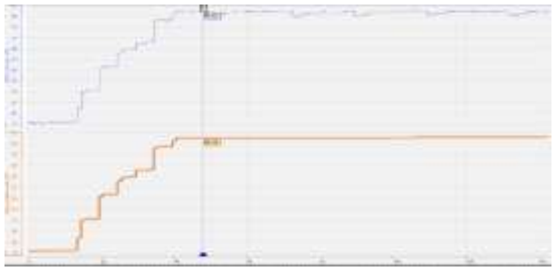


Fig 10: Before Modifying

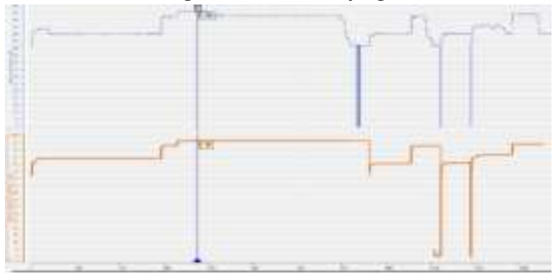


Fig 11: After Modifying

IV. RESULTS AND DISCUSSION

In this project the load on memory and CPU are reduced to maximum extent so as to incorporate the new features in the software and also to increase the efficiency of the microcontroller. As shown in the graphical representation in fig 12 the total memory reduction is ~13k Bytes by using different optimization techniques and the runtime of the CPU reduced below 92%.

DMI RAM: 125505-124330 = 1175 bytes. DMI RAM freed upto 1.17K Bytes by shifting measuring points to SPRAM.

FLASH: 2512748-2509512 = 3236 bytes. Flash memory freed upto 3.2K Bytes by removing the unused functions in the code.

DDDI Service reduced to 8k Bytes from 16k Bytes. Hence the total memory reduced to 12411 bytes.

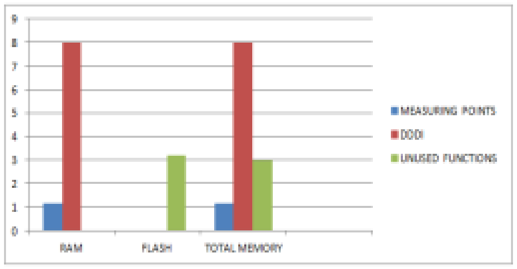


Fig 12: Graphical representation of memory reduction

IV. . CONCLUSION

The improvement in usage of memory tends to reduce the load on memory and CPU of TC1793 microcontroller by identifying all the variables, functions, calibration etc. which consumes more memory by shifting them to other memory locations depending on their frequency of use. The optimization of memory and CPU runtime helps to incorporate new features such as ADAS, FOTA and Cyber security along with the increase in efficiency of microcontroller.

REFERENCES

[1] 32-Bit TC1793 – Infineon Technologies.

[2] Yixin Chen and John Phillips, Delphi Powertrain Systems – "ECU Software Abnormal Behavior Detection Based On Mahalanobis - Taguchi Technique", 2008 World Congress, Detroit, Michigan, April 14-17, 2008

[3] Saiful A. Zulkifli and Muhd. AnisrudienMohd. Dali "Development of Vehicle Powertrain Simulation Module by Interfacing Matlab - ADVISOR to LabVIEW", 2017 IEEE Conference on Systems, Process and Control (ICSPC 2017), 15–17 December 2017, Melaka, Malaysia

[4] 32-Bit TC1793 – Infineon Technologies.

[5] Yixin Chen and John Phillips, Delphi Powertrain Systems – "ECU Software Abnormal Behavior Detection Based On Mahalanobis - Taguchi Technique", 2008 World Congress, Detroit, Michigan, April 14-17, 2008

[6] Saiful A. Zulkifli and Muhd. AnisrudienMohd. Dali "Development of Vehicle Powertrain Simulation Module by Interfacing Matlab - ADVISOR to LabVIEW", 2017 IEEE Conference on Systems, Process and Control (ICSPC 2017), 15–17 December 2017, Melaka, Malaysia.

[7] C. Bradatsch and T. Ungerer, R. Zalman and A. Lajtkep – "Towards Runtime Testing in Automotive Embedded Systems", Date of Conference:15-17 June 2011, 2011 6th IEEE International Symposium on Industrial and Embedded Systems

[8] Parag Kharche, Meera Murali, GeetanjaliKhot - "UDS implementation for ECU I/O testing" Date of Conference:3-5 Sept. 2018 2018 3rd IEEE International Conference on Intelligent Transportation Engineering (ICITE)

[9] RyosukeOkuda, Yuki Kajiwara, Kazuaki Terashima – "A survey of technical trend of ADAS and autonomous driving" Date of Conference: 28 -30 April 2014, published in- Proceedings of Technical Program - 2014 International Symposium on VLSI Technology, Systems and Application (VLSI-TSA)

[10] Golam Mostafa ; Yeasin Abedin-"Development of a graphical user interface using Windows 7-C# to download Intel – Hex formatted file into the flash of 89S52 microcontroller" - 2013 2nd International Conference on Advances in Electrical Engineering (ICAEE)

[11] Daniel S. Fowler, Jeremy Bryans, Siraj Ahmed Shaikh, Paul Wooderson – **"**Fuzz Testing for Automotive Cyber - Security" 2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)**.**

[12] Skruch, P, Dlugosz, R., Kogut, K., Markiewicz, P. et al., "The Simulation Strategy and Its Realization in the Development Process of Active Safety and Advanced Driver Assistance Systems," SAE Technical Paper 2015-01-1401, 2015, doi: 10.4271/2015-01-1401.

[13] Priyanka Sharma; Dietmar P. F. Möller–"Protecting ECUs and Vehicles Internal Networks" Date of Conference:3-5 May 2018 Published in: 2018 IEEE International Conference on Electro/Information Technology (EIT)

[14] Parag Kharche, Meera Murali, GeetanjaliKhot - "UDS implementation for ECU I/O testing" Date of Conference:3-5 Sept. 2018 2018 3rd IEEE International Conference on Intelligent Transportation Engineering (ICITE)

[15] RyosukeOkuda, Yuki Kajiwara, Kazuaki Terashima – "A survey of technical trend of ADAS and autonomous driving" Date of Conference: 28 -30 April 2014, published in-Proceedings of Technical Program - 2014 International Symposium on VLSI Technology, Systems and Application (VLSI-TSA)

[16] Golam Mostafa ; Yeasin Abedin-"Development of a graphical user interface using Windows 7-C# to download Intel – Hex formatted file into the flash of 89S52 microcontroller" - 2013 2nd International Conference on Advances in Electrical Engineering (ICAEE)

[17] Daniel S. Fowler, Jeremy Bryans, Siraj Ahmed Shaikh, Paul Wooderson – **"**Fuzz Testing for Automotive Cyber - Security" 2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)**.**

[18] Skruch, P, Dlugosz, R., Kogut, K., Markiewicz, P. et al., "The Simulation Strategy and Its Realization in the Development Process of Active Safety and Advanced Driver Assistance Systems," SAE Technical Paper 2015-01-1401, 2015, doi: 10.4271/2015-01-1401.

[19] Priyanka Sharma; Dietmar P. F. Möller–"Protecting ECUs and Vehicles Internal Networks" Date of Conference:3-5 May 2018 Published in: 2018 IEEE International Conference on Electro/Information Technology (EIT)

AUTHOR'S BIOGRAPHIES

J. C. Narayanaswamy

He received his Bachelor of Engineering (B. E) in Electronics and Communication Engineering from Dr. Ambedkar Institute of Technology in the year 1996. He received his ME degree from BMSCE, Bangalore University, Bangalore in the year 1998. He is currently pursuing Ph. D. degree from VTU University, in the field of Multiband microstrip patch antennas for IoT applications. He has attended various conferences and presented papers in the above domain. He is an Assistant Professor in Electronics and Communication Engineering Department in Bangalore Institute of Technology, Bangalore.

K. M. Shweta

She received Bachelor of Engineering (B. E) in Electronics and Communication Engineering from GM Institute of Technology, Davangere in the year 2017. She is pursuing Master of Technology (M.Tech) in VLSI Design and Embedded Systems from Bangalore Institute of Technology, Bangalore.