Journal of the Society of Automotive Engineers Malaysia
Volume 4, Issue 1, pp 73-81, January 2020
e-ISSN 2550-2239 & ISSN 2600-8092

Software Optimization of Vision-Based Around View Monitoring System on Embedded Platform

N. H. Mahamud[*1], Z. Zainal Abidin[**1], H. F. Mohd Zaki[1], Y. Mohd Mustafah[1], W. Sediono[1], H. Abd Rahman[2], S. Hanizam[1], M. A. A. Abdul Matin[1] and N. S. Ahmad Rudin[1]

[1]Centre for Unmanned Technologies (CUTe), Kulliyyah of Engineering, International Islamic University Malaysia, 53100, Kuala Lumpur, Malaysia
[2]Delloyd R&D (M) Sdn. Bhd., Jln. Kebun, Kampung Jawa, 41000 Klang, Selangor, Malaysia

Corresponding authors: [*]hidayah.smksb@gmail.com.my; [**]zzulkifli@iium.edu.my

ORIGINAL ARTICLE **Open Access**

Article History:

Received
14 Sep 2019

Received in revised form
30 Nov 2019

Accepted
1 Dec 2019

Available online
1 Jan 2020

Abstract – *Image processing algorithm requires high computational power. Optimizing the algorithm to be run on an embedded platform is very critical as the platform provides limited computational resources. This research focused on optimizing and implementing a vision-based Around View Monitoring (AVM) system running on two embedded boards of Cortex-A7 quad and Cortex-A15 quad-core, and desktop platform of Intel i7 core. This paper presented a study on several techniques of software optimization that is removing code redundancy and multi-threading. The two methods improve the total processing time of the AVM system by 45% on ARM Cortex-A15 and 47% on ARM Cortex-A7.*

Keywords: Embedded platform, image processing, software optimization, Advanced Driver Safety System (ADAS), vision-based ADAS, AVM system

Journal homepage: www.jsaem.saemalaysia.org.my

1.0 INTRODUCTION

Around View Monitoring (AVM) system can be classified as a part of the Advanced Driver Safety System (ADAS). AVM system provides a 360-degree view of the vehicle's surroundings. For any safety application, a real-time operation is a basic requirement that needs to be fulfilled together with the system accuracy (Ahmad Rudin et al., 2018). On the other hand, the image processing algorithm requires high computational power. Implementing a vision-based system on an embedded Advanced RISC Machine (ARM) platform while maintaining the high-performance system requirement is becoming a great challenge. ARM core is commonly used for the embedded platform as it is developed to give the best performance for an embedded board with lower processing power required.

A lot of researches have been carried out to explore effective optimization methods for vision-based system running on an embedded ARM platform. The optimization will greatly affect the overall performance, power dissipation and overall cost of a system (Park et al.,

2013). This paper presents several software optimization methods to be applied on existing vision-based AVM system project running on an embedded platform.

2.0 RELATED WORKS

Generally, optimization techniques on embedded can be summarized to several categories such as compiler optimization, source code modification, memory optimization and hardware-level optimization (ARM, 2014). Several pieces of research focused on optimization of the developed system that utilizes the embedded multi-core system (Ma & Wang, 2016), memory management (Muck & Frohlich, 2011; Lei & Xiao-ya, 2011), hardware and system-level optimization (Dekkiche et al., 2016; Dedeoğlu et al., 2011; Singhal et al., 2012) and software optimization (Park et al., 2013; Joshi & Gurumurthy, 2014).

Multi-processing and multi-threading are effective to greatly reduce the processing time of an application by running several tasks concurrently. Ma and Wang (2016) discussed on parallel computing framework on a multi-core system using single-threading, multi-threading and parallel computing using OpenMP. It can be concluded that the correct implementation of parallel computing can greatly improve the speedup of a program execution. Equally important, synchronization is very critical in any parallel processing task.

Based on research by Park et al. (2013) and Joshi & Gurumurthy (2014), loop transformation techniques generally reduce the number of iterations and loop overhead that assist in increasing the execution time and code density of applications running on embedded ARM. Examples of loop transformations are loop termination, loop unrolling, loop reversal, loop unswitching, and loop fusion.

GNU Compiler Collection (GCC) also provides several optimization flags that can be enabled during the compilation process (ARM, 2014). General optimizations include (1) -O0 for no optimization, (2) -O1 for common, basic optimization methods, (3) -O2 for additional optimization such as instruction scheduling, (4) -O3 for powerful optimization that helps to produce faster application but at the same time increases the program size and lastly (5) -Os that focused to decrease the size of program but may decrease the processing speed.

Other optimization methods include kernel-level optimization (Dekkiche et al., 2016) and the use of GPU and DSP processor to support parallel computing of multimedia programming such as audio, image and video processing (Dedeoğlu et al., 2011; Singhal et al., 2012)

3.0 METHODOLOGY

3.1 System Setup

This subchapter will discuss the overview of the AVM system to be implemented and details of the embedded ARM platform used for testing. The application runs on two embedded boards of different specifications and a desktop for a performance comparison.

3.1.1 Vision-based AVM system

This research work focused on optimizing a vision-based 1-input AVM system. The AVM system primarily involves image-stitching and motion estimation. Image stitching will create a continuous image of the surrounding vehicle based on input from one camera allocated at the front or rear of the vehicle. The area of the image to be stitch depends on the speed and angle of the vehicle.

This system integrates the motion estimation technique based on block matching methods to determine the region of the image to be stitch. The algorithm will search every pixel or a block of the pixel from the current frame and compare it to a reference called template taken from the previous frame. From the block matching detection, the motion of the vehicle can be calculated.

The AVM application mainly used the OpenCV library for image processing tasks. OpenCV is an open-source computer vision library that provides a large support for optimized image processing algorithm. OpenCV also provides NEON instruction support on some part of the library function. This greatly helps to optimize the execution of the image processing task on the ARM platform.

The AVM application is developed using QT Creator Integrated Design Environment (IDE), running on Ubuntu 16.04 LTS. To run the AVM system on an embedded ARM platform, the source code will be cross-compiled using an SDK of the targeted platform. The cross-compile step will create a compatible binary file to be run on the targeted board. During the compilation process, compiler optimization -O3 in enabled.

3.1.2 Renesas R-Car H2 (ARM Cortex-A15)

R-Car H2 is a development board introduced by Renesas Electronics Corporation that targeted a high-performance System on Chip (SoC) for the automotive industry (Renesas, n.d.). The board incorporates the ARM Cortex-A15 quad operating at 1.5 GHz based on 32-bit ARMv7 CPU architecture. Each core supports Vector Floating Point v4 Extension (VFPv4) and NEON instruction set technology (Advanced SIMD instruction). NEON helps to improve multimedia operations such as audio and video processing and computer vision. The board also includes Imagination's PowerVR G6400 GPU operating at 550 MHz. The R-Car H2 development board runs of Linux based operating system. Figure 1 shows a demonstration of the AVM system running on the Renesas R-Car H2 development board.

3.1.3 Telechip TCC 8971 (ARM Cortex-A7)

TCC 8971 is an automotive-grade SoC with powerful multimedia support with low power consumption (Telechips, n.d.). TCC 8971 incorporates ARM Cortex A7-Quad (1.0 GHz) based on 32-bit ARMv7 CPU architecture. Each core supports NEON instruction set technology. It also includes 2D/3D Graphic support including MALI400 GPU MP2 and GC300 for 2D. The system runs on a Linux-based operating system. Figure 2 shows the AVM system running on the Telechip TCC8971 development board.

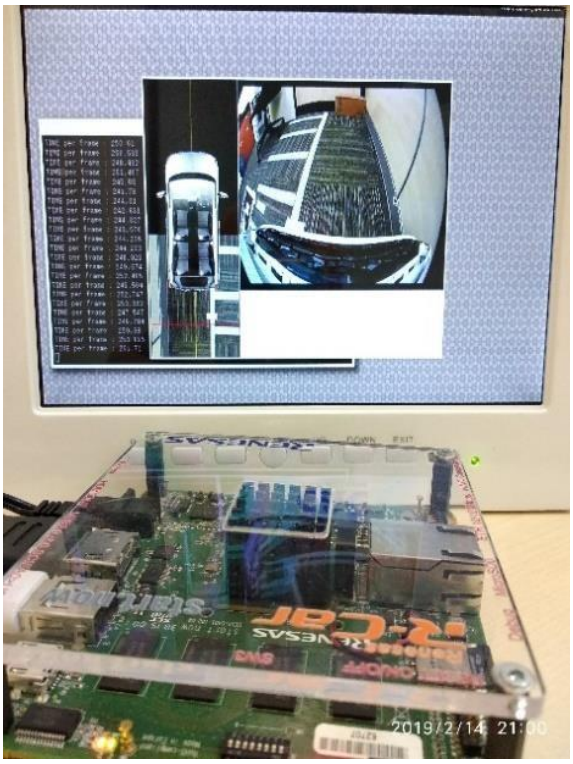


Figure 1: AVM system running on Renesas R-Car H2

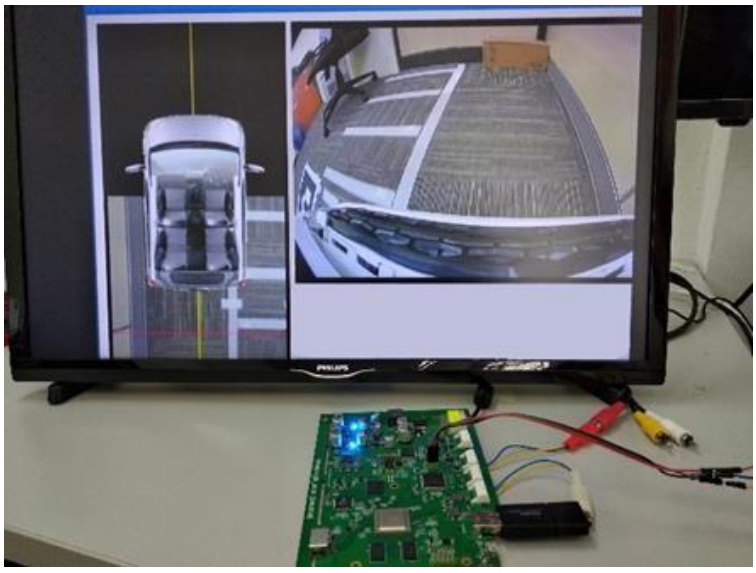


Figure 2: AVM system running on Telechip TCC8971

3.1.4 Desktop Platform (Intel Core i7)

The desktop platform incorporates 64-bit, Quad Intel Core i7 chip operating at 2.40 GHz. The platform runs Linux based operating system of Ubuntu 16.04.6 LTS. It also provides graphics engine support with Intel (R) HD Graphics 4600 and NVIDIA GeForce GT 750M. Figure 3 shows the AVM system running on the desktop platform.

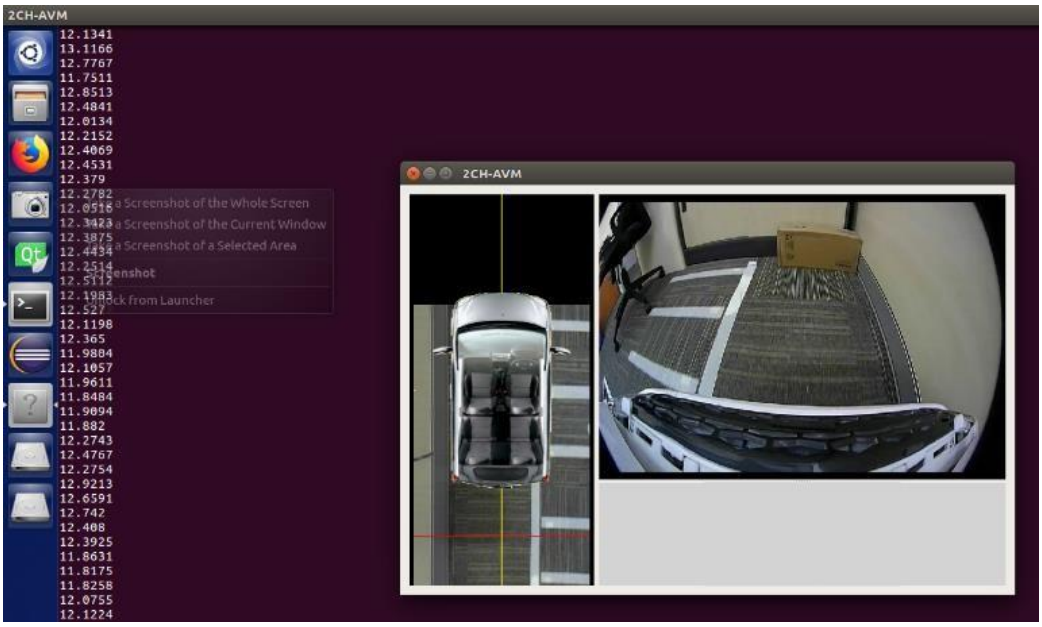


Figure 3: AVM system running on a desktop platform

4.0 RESULTS AND DISCUSSION

This subchapter will discuss the optimization methods applied and discussions on the results.

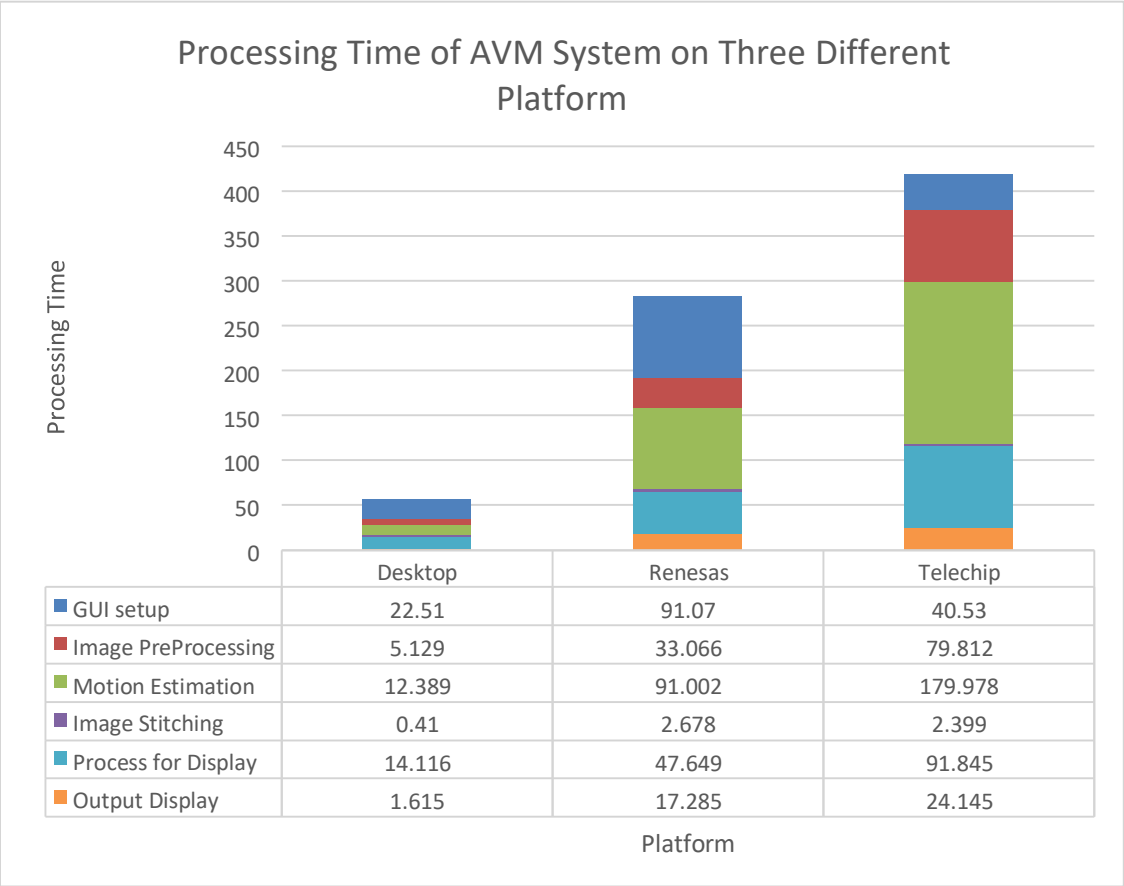
4.1 Identifying Hotspot

The first step in system optimization is to determine a program hotspot or section of code that takes a lot of processing time per frame. Important functions are timed using std::chrono library. For benchmarking purposes, the AVM application is set to run for 3 seconds on the desktop platform and 10 seconds on an embedded platform. The average processing time per frame is calculated and recorded.

The AVM application is divided into several tasks to compute the timing. Figure 4 shows the average processing time of the AVM original source code. The processing time is timed per frame except for the GUI setup task that runs once at the program starts.

From the result, we can see that the Telechip platform takes more processing time compared to other platforms. The initial frame per second (FPS) of the AVM application running on Desktop, Renesas, and Telechip is 15, 5 and 2 fps respectively.

Referring to Figure 4, we can see that Motion Estimation and Process for Display takes a great processing time per frame compared to other functions. Thus, we will be focusing more on the optimization of the two functions.



	Desktop	Renesas	Telechip
GUI setup	22.51	91.07	40.53
Image PreProcessing	5.129	33.066	79.812
Motion Estimation	12.389	91.002	179.978
Image Stitching	0.41	2.678	2.399
Process for Display	14.116	47.649	91.845
Output Display	1.615	17.285	24.145

Figure 4: Processing time of sub-functions in main process task

4.2 Removing Redundancy

Removing redundant code is an example of source code modification. We determine and modified any unnecessary steps or instructions in the code. It reduces lines of instructions in the source code, cycle count and even the number of memory access times.

From the AVM original source code, we identified that a variable declaration of OpenCV cv::Mat type for the same variable has been initialized for every frame. This causes the same initialization instruction for the same variable to be executed at every frame. Accordingly, the part of the code is restructured to a global variable and called once at the function's constructor code. The percentage of improvement for the Process for Display task running on several platforms is shown in Table 1. The percentage is calculated based on the initial processing time of the task.

Table 1: Percentage of improvement by removing redundancy (Process for Display task)

Platform	Improvement Percentage
Desktop	83.76 %
Renesas R-Car H2	67.18 %
Telechip TCC 8971	68.90 %

Next, variable declarations in the Motion Estimation function is reorganized. All possible variable declarations are moved into the class' header file. The resulted processing time obtained for Motion Estimation function after the modification is tabulated in Table 2.

Table 2: Percentage of improvement by removing redundancy (Motion Estimation function)

Platform	Improvement Percentage
Desktop	24.74 %
Renesas R-Car H2	27.25 %
Telechip TCC 8971	32.69 %

Based on the result obtained, we can conclude that the repeated initialization of variables causes a great increase in the total execution time of the program running on an embedded platform. Any redundant code should be restructured to be outside of a function call or any flow control statements especially for or while loop. Generally, having a good structure of code helps to increase the processing speed of the program.

Even though declaring a variable as global helps to prevent declaration and initialization at every frame, it also has some disadvantages. The global variable stays at a part of the stack for a global variable and stays in the memory for the whole execution of the program. On the other hand, local variables allocated and deallocated a memory space whenever the function is being used. Using a global variable may increase the program's complexity as the variable stays alive as long as the program is still running. The global variable is also visible to other functions in the program. Any accidental changes by other functions may cause an error to the whole program.

4.3 Multi-Threading

Multi-threading is a promising technique to improve the processing speed of any algorithm. A single thread executes a single operation; thus multi-threading allows multiple operations or tasks to be executed concurrently. C++11 enables the programmer to easily code for multithreading using std::thread library from thread.h header file.

The Motion Estimation function used a block matching based algorithm. High processing time is expected in the algorithm as it needs to search through every pixel block in the region of interest. To optimize the process, the searching region is divided into two smaller regions of interest. This will enable a searching task of one region to run on the main thread while another searching task of the second region to run on a new thread. As a result, the processing time of the Motion Estimation function reduces. The percentage of improvement concerning the initial execution time is calculated and tabulated as shown in Table 3.

Table 3: Percentage of improvement by multi-threading method (Motion Estimation function)

Platform	Improvement Percentage
Desktop	40.78 %
Renesas R-Car H2	40.56 %
Telechip TCC 8971	44.83 %

Based on the result obtained, by executing the block matching algorithm in multithread, the percentage of processing time for Motion Estimation has improved on all the three platforms by almost half.

4.4 Implementing Optimization on AVM System

All the optimization techniques tested and discussed above focus on source code modification to improve the processing speed. We combined all the optimization methods discussed and applied the techniques in the AVM application. The total processing time per frame is computed and compared with the initial time. The percentage of improvement is calculated and tabulated in Table 4.

Table 4: Percentage of total processing time improved (AVM application system)

Platform	Improvement Percentage
Desktop	48.91 %
Renesas R-Car H2	45.21 %
Telechip TCC 8971	47.30 %

The final estimated FPS of the application after optimization are 39, 10 and 5 running on Desktop, Renesas and Telechip platform respectively. The processing time improved by half after applying the optimization method. Based on the result, the Renesas board seems to be doing better than the Telechip board. The difference might come from the different ARM Cortex-A core version which also has a different operating frequency. Renesas also have VFP architecture support available on the board.

5.0 CONCLUSION AND FUTURE WORKS

In conclusion, the execution time on the desktop platform is proved to be the best because of its powerful processing power. In comparison, even though the processing speed on ARM development board (Renesas and Telechip) has improved, we still far from having the system to work in real-time on an embedded platform. It is more challenging as the computer vision algorithm itself is computationally expensive. Thus, it is critical to have a good optimization technique that fully utilize the limited resources of the embedded platform.

More research will be done to optimize the AVM application to be able to run at real-time speed on an embedded platform. One of the recommendations is to code part of them in a low-level language (such as NEON) or directly accessing the kernel or memory. The implementation of the OpenCV library also can be restructured. The use of GPU using the CUDA and OpenCL framework can also be explored.

ACKNOWLEDGMENTS

This research was sponsored by CREST and Delloyd R&D Sdn. Bhd., under the grant CREST ID: P11C217 (Smart Driver Assistance System).

REFERENCES

Ahmad Rudin, N.S., Mohd Mustafah, Y., Zainal Abidin, Z., Cho, J., Mohd Zaki, H.F., Nik Hashim, N.N.W., & Abdul Rahman, H. (2018). Vision-based Lane Departure Warning System. *Journal of the Society of Automotive Engineers Malaysia*, *2*(2), 166-176.

ARM (2014). *ARM Cortex-A series: Programmer's guide*. England: ARM Limited.

Dedeoğlu, G., Kisačanin, B., Moore, D., Sharma, V., & Miller, A. (2011). *An optimized vision library approach for embedded systems*. Proceedings of the IEEE Workshop on Embedded Computer Vision 2011, 8-13. doi: 10.1109/CVPRW.2011.5981731

Dekkiche, D., Vinck, B., & Merigot, A. (2016). *Targeting system-level and kernel-level optimizations of computer vision applications on embedded systems*. Proceedings of the Sixth International Symposium on Embedded Computing and System Design (ISED). doi: 10.1166/jolpe.2017.1510

Joshi, P.V., & Gurumurthy, K.S. (2014). *Analysing and improving the performance of software code for real time embedded systems*. Proceedings of the 2nd International Conference on Devices, Circuits and Systems (ICDCS), 1-5. doi: 10.1109/ICDCSyst.2014.6926134

Lei, W., & Xiao-ya, F. (2011). *Study on L2 cache of multi-core processor and optimization for embedded*. Proceedings of the 2011 IEEE International Conference on Signal Processing, Communications and Computing (ICSPCC), 1-5. doi: 10.1109/ICSPCC.2011.6061647

Ma, W., & Wang, Z. (2016). *Performance analysis of parallel computing framework on embedded multi-core trustworthy systems*. Proceedings of the International Symposium on System and Software Reliability (ISSSR), 25-59. doi: 10.1109/isssr.2016.01

Muck, T.R., & Frohlich, A.A. (2011). *Run-time scratch-pad memory management for embedded systems*. Proceedings of the IECON – 37th Annual Conference of the IEEE Industrial Electronics Society, 2833-2838). doi: 10.1109/IECON.2011.6119761

Park, I., Lee, H., & Lee, H. (2013). *Software optimization for embedded communication systems*. Proceedings of the International Conference on Information Networking 2013 (ICOIN), 676-679. doi: 10.1109/ICOIN.2013.6496708

Renesas (n.d.). *R-Car H2*. Tokyo, Japan: Renesas Electronics Corporation. Retrieved from https://www.renesas.com/sg/en/solutions/automotive/soc/r-car-h2.html

Singhal, N., Yoo, J., Choi, H., & Park, I. (2012). Implementation and optimization of image processing algorithms on embedded GPU. *IEICE Transactions on Information and Systems*, *E95-D*(5), 1475-1484.

Telechips (n.d.). Intelligent automotive solution for driving & entertainment. Seoul, Korea: Telechips Inc. Retrieved from https://www.telechips.com/eng/product/automotive.php