The Art of Automotive Application Design: Performance Optimizations for a 32bit Automotive Microcontroller

By Giovanni D'Aquino, Senior Application Engineer, STMicroelectronics

The design of automotive Electronic Control Unit (ECU) systems is a strict mixture of requirements concerning various aspects of performance, including core execution speed, power consumption, and electromagnetic emissions (EMC). Silicon manufacturers also must consider the die area itself.

Due to the increased use of operating systems, standard software infrastructure, and complex real time software in the ECU, the microcontroller’s calculation power is a vital element in the quest for optimal microcontroller design.

While silicon manufacturers are always on the lookout for ways to improve system performance for future developments, users of over the counter (OTC) MCUs face the major challenge of meeting their performance requirements at the best price.

We’ll illustrate the major considerations designers will face when meeting application requirements with a 32bit microcontroller architecture design, and explain how to get maximum performance during software development by knowing the architecture and using common debugging techniques.

Introduction

In the world of automotive microcontrollers, whatever the design aspect, the goal is to get the maximum performance respecting application requirements at a minimum price.

The improvement process is driven by cost aspects, but equally from what customers expect in term of features and performances requested of the SoC.

Silicon suppliers take benefit of improved manufacturing and process technology to continuously design new microcontroller generations with backward compatible hardware and software allowing better performance with lower cost. Of course, each device, even if primarily designed to address a specific market segment (chassis, safety, or body, for example), will be used differently by several customers and for different applications. Thus, finding a design tradeoff that takes into account both design requirements and customer feedback on current and future applications is a significant challenge.

In this scenario, from a customer perspective, it is no trivial task to select an MCU to design an application. Comparisons between different MCUs are often affected by the users’ previous experiences, by a lack of consistent parameters to analytically show MCU features versus the ones needed for the application. This includes the availability of consolidated development and debug toolchains that can help speed up SW development.

The challenge for SoC integrators is to produce a suitable tradeoff between parameters often in contrast each other to meet application requirements.
The design of the application, often an improvement or a redesign, requires a strong analysis of workloads, timelines and application criticism in order to fine-tune all configurations. Though it may seem as complex and magical as an obscure art practiced by witch doctors, technicians follow rigorous development methodologies like ‘PDCA,’ which takes into account all the MCU’s intrinsic features and development tools to reach the planned target performances.

**Performance optimizations**

In order to improve the MCU’s overall system performance, much has to be handled correctly - particularly the large amount of instructions and data fetch to the memory hierarchy. This problem, also known as “Memory Wall,” strongly impacts all accesses, specifically the ones to the embedded flash memory (slower than SRAM) and introduces to the access flow some latencies, which are dependent on the different operating frequency of core and memory hierarchy architecture.

Another key topic that strongly influences application development is the intrinsic real-time nature of automotive systems where the activities are triggered by interrupts or are dependent on real-time data like ADC inputs from the external world.

A popular development methodology, focused on managing SW process improvement, is based on continuous iteration bringing the SW developer ever closer to the goal, the PDCA (Plan-Do-Check-Act management method).

![PDCA Diagram](image)

*Figure 1: PDCA*

In the PLAN stage, the objectives and actions necessary to obtain results in accordance with the expected target are established.

In the DO stage, the plan for changing HW/SW configurations changes in the application is launched.

The CHECK stage allows the developer to review the previous step effectiveness and to measure and compare the results versus the expected performance improvements. Finally, in the ACT stage, developers either move on to the next project priority, or enhance and rework the current one to match the expected target.
This methodology, supported by a good development environment and strong device knowledge, gives the user a step-by-step guide to improve the automotive system’s overall performance.

Whatever language, design flow and applied programming best practices developers may use, the debugging phase is probably the most crucial in the application development and optimization stage.

And because the debugging phase impacts overall system development, the tools used are key to reach application targets. Tools allow the user to see what happens within the device, how many resources it is using and how it interacts with the external world. This, of course, brings to focus any problems, bottlenecks, and inefficiencies in general. However, the drawbacks of using any tools are the time and resources consumed, especially when real time systems are involved.

There are several debugging techniques, but probably the best one is based on the use of profilers. Profilers provide assistance to inspect the code execution, to understand how the code behaves and to know the frequency of functions reference as well as the time spent by the MCU to execute them.

These tools often also offer code coverage information, giving the user evidence of the statement and decision coverage required for a product’s quality certification requirement. System profiling offers developers a picture of the distribution of all workloads, which helps them find out how and where the computational power is consumed, the nature of data access distribution versus the memory hierarchy, and finally, helping identify the optimization areas in terms of algorithm and data mapping.

The base for this iterative process of performance improvement is a strong knowledge of the architecture and features of the MCU used for the application. Starting from the scratch, the user can disable or put in some lower power consumption mode all unnecessary modules and after that can focus all the analysis on resource load balancing trying to speed up the application code as well as to reduce power consumption by limiting MCU operating frequency.

**Automotive application requirements**

The ideal MCU automotive system should have a low power consumption, respect EMC emissions standards as well as all requirements in terms of timeline (e.g. real time applications), and this implies that it should improve execution speed.

These metrics are bound by physical laws, making it more or less impossible to optimize them all together.

For example, consider the well-known law defining dynamic power consumption \( (P \propto V^2 f) \); there is a linear relation versus the execution speed through the core operating frequency, while voltage is strongly related to the operating frequency since it’s not possible to step down the voltage without decreasing the frequency as well.
In addition, high frequencies around the MCU due to switching (clocks, pads, communication busses) strongly impact EMC emissions.

Using a well-established MCU as a particular example, the SPC564Axx from STMicroelectronics, we’ll highlight the areas that should be focused on when optimizing an application.

Quick overview of the SPC564Axxx
The device is part of an automotive powertrain 32bit microcontroller family that serves mid-range engine management and transmission control application areas.

Figure 2: SPC564Axx Block Diagram
It embeds the e200z4 that complies with the Power Architecture® embedded category architecture and implements the VLE (variable-length encoding) APU, providing improved code density.

In order to further optimize time-critical functions, this core also has additional instruction support for digital signal processing (DSP), called Signal Processing Extension Auxiliary Processing Unit (SPE-APU).

The correct use of the peripherals can dramatically improve overall system performance. In particular, the use of the interrupt controller, the enhanced Direct Memory Access (eDMA), and intelligent peripherals such as the Enhanced Timer Processing unit (eTPU2), can off-load significant work from the CPU.

In general, the main hardware features that can be configured to impact the overall performance of the device are the Instruction Cache, Frequency-modulated PLL, Flash Bus Interface Unit, XBAR, and Branch Prediction.

**Instruction Cache**
The exact usage of cache is application dependent but in general it could improve overall performance enabling it for all internal and external memories that code is being executed from and consider locking critical performance routines in cache.

**Frequency-Modulated PLL**
The frequency-modulated phase-locked loop (FMPLL) allows the user to generate high speed system clocks from a crystal oscillator or external clock generator. Due to the insertion of additional Flash wait states as system frequency increases that system performance does not scale linearly.

**Flash Bus Interface Unit**
The Flash Bus Interface Unit (FBIU) interfaces the system bus to the Flash memory array controller. It contains prefetch buffers and a prefetch controller, which, if enabled, speculatively prefetches sequential lines of data from the Flash array into the buffer. Prefetch buffer hits allow zero-wait state responses.
The device flash has two sets of four line read buffers, one set for the 128-bit wide low- and medium-address space and one set for the 256-bit wide high address space. This implies that to improve performance and reduce flash access latency, critical code should be located in the high address space.

**Crossbar Switch**
The multi-port crossbar switch (XBAR) allows for concurrent transactions to occur from any master port to any slave port. If a slave port is simultaneously requested by more than one master port, arbitration logic selects the higher priority master and grants it ownership of the slave port. It is possible to choose, either with fixed or round robin priority mode, which requesting master is granted access. In order to maximize data throughput, it is essential to keep arbitration delays to a minimum. Further, by correctly parking slaves on relevant masters, the initial access times to the slaves can be minimized by negating any initial arbitration penalties.

**Branch Target Buffer (BTB)**
To resolve branch instructions and improve the accuracy of branch predictions, the core implements a dynamic branch prediction mechanism using a branch target buffer (BTB), a fully associative address cache of branch target addresses. Its purpose is to accelerate the execution of software loops with some potential change of flow within the loop body.

**Memory Management Unit (MMU)**
Finally, although the MMU does not directly impact performance, it is essential that it is correctly configured. It is within the MMU that memory regions are configured to permit the use of system cache to improve performance and Variable Length Encoding (VLE) to enhance code density.
**Conclusion**
Performance optimization is not an obscure art but a strong challenge where developers, basing their work on knowledge and experience, act as scientists to determine the best tradeoff between different parameters to meet application requirements.

Giovanni D'Aquino is senior application engineer for STMicroelectronics in Catania, Italy.

He joined the company in 2003 and has been working in R&D as a SW developer and OS expert. He has nine years experience in Power Architecture™ and currently works in the automotive microcontroller division as Chassis and Safety microcontroller's application leader.

Giovanni graduated with a Master’s degree in Computer Engineering in 2002 and received his second Master’s specializing in Management of Development’s Projects about ICT Security in 2004.