Microprocessor (MPU) or Microcontroller (MCU)? What factors should you consider when selecting the right processing device for your next design

Authors:
By Frédéric Gaillard, product marketing manager and Andreas Eieland, senior product marketing manager, Atmel

Selecting the right device on which to base your new design can be daunting. The need to make the right balance of price, performance and power consumption has many implications. First, there will be the immediate technology considerations for the design you are able to embark on, but if the device, whether microcontroller (MCU) or microprocessor (MPU), becomes the basis of a platform approach for a range of new products then the decision can have long-lasting consequences.

Firstly, let us consider some of the primary differences between an MCU and MPU. Typically an MCU uses on-chip embedded Flash memory in which to store and execute its program. Storing the program in this way means that the MCU has a very short start-up period and can be executing code very quickly. The only practical limitation to using embedded memory is that the total available memory space is finite. Most Flash MCU devices available on the market have a maximum of 2 Mbytes of Program memory and, depending on the application, this may prove to be a limiting factor. MPUs do not have memory constraints in the same way. They use external memory to provide program and data storage. The program is typically stored in non-volatile memory, such as NAND or serial Flash, and at start-up is loaded into an external DRAM and then commences execution. This means the MPU will not be up and running as quickly as an MCU but the amount of DRAM and NVM you can connect to the processor is in the range of hundreds of Mbytes and even Gbytes for NAND. Another difference is power. By embedding its own power supply, an

MCU needs just one single voltage power rail. By comparison, an MPU requires several difference voltage rails for core, DDR etc. The developer needs to cater for this with additional power ICs / converters on-board.

From the application perspective, some aspects of the design specification might drive device selection in particular ways. For example, is the number of peripheral interface channels required more than can be catered for by an MCU? Or, does the marketing specification stipulate a user interface capability that will not be possible with an MCU because it does not contain enough memory on-chip or has the required performance? When embarking on the first design and knowing that, it is highly likely there will be many product variations. In that case, it is very possible a platform-based design approach will be preferred. This would stipulate more “headroom” in terms of processing power and interface capabilities in order to accommodate future feature upgrades.

An attribute that is difficult to determine is the required processing performance any given design might require. Processing power, measured in terms of Dhrystone MIPS (DMIPS), helps quantify these criteria. For example, an ARM Cortex-M4-based microcontroller such as Atmel’s SAM4 MCU is rated at 150 DMIPS while an ARM Cortex-A5 application processor (MPU) such as Atmel’s SAMA5D3 can deliver up to 850 DMIPS. One way of estimating the DMIPS required is by looking at the parts of the application that may be performance hungry. Running a full operating system (OS), such as Linux, Android or Windows CE, for your application would demand at least 300 – 400 DMIPS. For many applications, a straightforward RTOS might suffice and an allowance of 50 DMIPS would be more than adequate. Using an RTOS also has the benefit that it requires little memory space; a kernel of just a few kB being typical. Unfortunately, a full OS demands a memory management unit (MMU) in order to run; this in turn specifies the type of processor core to be used and require more processor capability.

For running applications that are more number-crunching intensive enough, DMIPS allowance needs to be reserved on top of any OS and other communication and control tasks. The more numeric-based the application, the more likely a MPU is required.

Whether the intended application is aimed at consumer electronics or industrial automation, the user interface (UI) can be a serious consideration. As consumers, we have become familiar and comfortable with using colourful and intuitive graphical UIs. Industrial applications are increasingly using this method of operator interaction although the operating environment can limit how much this is warranted. For the UI there are a number of factors. Firstly, is the processing overhead required. For a UI library such as Qt, which is widely used on top of Linux, an overhead of 80 – 100 DMIPS might suffice. The second factor is to do with the complexity of the UI. The more you have animations, effects, multimedia content, the more changes are applied to the image to be displayed, the more processing power and memory you need. And this requirements scale up with the resolution, that is why for applications designed to be UI centric a MPU is more likely to suit. On the other hand, a simpler UI with pseudo-static images on a lower resolution screen can be addressed by an MCU. Another argument in favour of the MPU is that the generally come equipped with an embedded TFT LCD controller. Very few MCUs have this capability. The TFT LCD controller and some other external driver components have to be added externally. So, while possible to achieve with an MCU, the developer needs to look at the overall BOM. Some Flash MCUs are now coming onto the market with TFT LCD controllers embedded but still, there must be enough embedded SRAM memory available to drive the display. For example, the QVGA 320 x 240 16-colour format requires 150 kB of SRAM to feed and refresh the display. This is a fairly high amount of SRAM to dedicate; so extra memory might be required further adding to the BOM and bridging the gap with the MPU solution. More complex and advanced graphical UIs, especially using screens larger than 4.3” inches, would stipulate an MPU. If MPUs are seen to dominate when it comes to run a UI on a colour TFT screen then MCUs are the kings for segment or dot matrix LCD control and other screens with serial interfaces.

From the connectivity standpoint, most MCU and MPU devices are available, with all the common popular peripheral interfaces. But high speed communication peripherals such as HS USB 2.0, multiple 10/100 Ethernet ports or Gigabit Ethernet port are generally only found on MPU because they are better capable to handle and process large amounts of data. Whether there are enough suitable channels and bandwidth to handle the data traffic is a key question. Depending on the communication protocols used, the impact on code space using third-party stacks should be checked. Applications demanding high-speed connectivity especially in combination with using OS-based stacks will require a MPU-based design.

Another key aspect that will drive the selection between an MCU and an MPU is the need for a real-time/deterministic behaviour of the application. Because of the processor core used in an MCU, as well as the embedded flash and considering the software used that is either an RTOS or bare metal C, the MCU will definitely take the lead on this aspect and will address perfectly the most time critical and deterministic applications.

A final point to consider is power consumption. While MPUs do have low power modes there are not as many or as low as the ones you would find on a typical MCU. With the external hardware supporting an MPU has an added factor, putting an MPU into a low power mode might also be slightly more complex. Also, the actual consumption of an MCU is magnitudes lower than an MPU, in low power mode for example with SRAM and register retention, you can consider a factor 10 to 100. Obviously this is directly related to the amount of RAM an operating system requires and therefore to be powered to resume operation instantaneously. The decisions involved in selecting either an MCU or MPU-based approach are many and involve performance, capability and the BOM budget. Broadly speaking, MCUs tend to be used in cost optimised solutions where a tight control of BOM and power saving is essential. MPUs tend to be used for functionally rich and high performance applications. MCUs tend to be used in ultra low power applications such as remote controls, consumer electrics and smart meters where the design emphasis puts longevity of battery life and none or little UI interaction. They are also used where a highly deterministic behaviour is needed. MPUs are ideal for OS-based industrial and consumer applications that might be compute intensive and require multiple high-speed connectivity or a rich UI.

Selecting a vendor offering highly compatible MCU and MPU products where you can easily migrate up and down and maximize software reuse provides the best return on investment over time.