Choosing a MCU for your next design; 8 bit or 32 bit?

Author:
Ingar Fredriksen, MCU marketing director, Atmel and Pal Kastnes, MCU Applications Staff Engineer, Atmel

The rise in popularity of 32-bit MCU devices across the embedded community is well known. The ability of these function-rich devices to suit an array of different applications explains why many embedded developers are selecting them for their next designs. Designers recognize that complex devices such as these offer everything they need in terms of raw compute power, a rich peripheral set and easy access to a wide range of development tools and libraries. Many of these 32-bit devices are based on the highly successful ARM® cores, and developers feel confident in having access to second source devices and a comprehensive set of development, test and validation tools being available in the market.

	2012	2013	2014	2015	2016	2017	CAGR
Total Semiconductor	325,367	339,666	361,612	385,052	395,974	413,602	4.9%
Microcontroller (MCU)	16,008	16,202	17,211	18,799	19,307	20,480	5.1%
4-bit MCU	154	159	161	157	145	133	-2.8%
8-bit MCU	6,057	6,565	6,936	7,532	7,768	8,259	6.4%
16-bit MCU	4,021	3,611	3,765	4,060	4,053	4,019	0.0%
32-bit MCU	5,776	5,868	6,349	7,050	7,341	8,069	6.9%

Table 1 – Semiconductor MCU revenue market forecast - millions of dollars; courtesy of iSuppli.

But a closer look at recent MCU market trends, see Table 1, (*iSuppli MCU market 2012 – 2017*), reveals that it is not only the 32-bit devices that are experiencing strong growth. 8-bit MCUs are also showing a compound growth rate close to that of 32-bit; 6.4% compared to 32-bit 6.9%. Other industry analysts forecast that the growth rates of 8- and 32-bit microcontrollers are identical. The growth trend of 8-bit devices clearly highlights that there must be some compelling reasons to use an 8-bit device in place of a 32-bit MCU. This article aims to give some insight to the reasons why 8-bit devices are retaining market share.

Essential differences

The principle differences between 8- and 32-bit MCUs are those of cost and price structure, CPU performance, ease of use, efficiency in hardware near functions and static power consumption. When embarking on a new design, developers need to carefully scope out the requirements for an MCU based on the amount of processing capability required, the degree of interfacing needed and, for battery-powered designs, the all important power consumption profiles. There is no doubt that a 32-bit MCU has a higher performance capability than an 8-bit device, but the engineer is faced with the traditional decision of choosing between what is the best device available in the market against what the application actually needs. This is about selecting a device that is right for his application rather than making it a choice of which CPU core (8- or 32-bit) should be used.

Of course, these decisions will greatly influence the likely BOM cost. Clearly, with a lower gate count, a less complex 8-bit device will be cheaper than a 32-bit device. Making a comparison against 8- and 32-bit MCUs available from leading vendors, each with a similar amount of Flash memory, pin-out etc, typically the 8-bit device will be about 20% cheaper. But this is only the first of many considerations. Another aspect relates to the ease with which a designer can start setting up for a new development.

Ease of development

MCU suppliers tend to add more features and functionality to their 32-bit devices as opposed to their 8-bit products. Consequently, there are far more setup considerations to deal with on a more complex device. While some 32-bit MCUs can run with a limited setup similar to that of an 8-bit device, it doesn't apply if you want to use the more powerful device's additional features. For example, a typical 32-bit ARM device will have independent clock settings for the core itself, the AHB bus, the APBA bus, in addition to as the APBB bus. They can all be set to different frequencies. You will typically also have to switch to the clock you want to use as this is set in software not hardware as in the case of most of the 8-bit parts. Changing the clock means you will also have to set up the wait states for the Flash and possibly decide this based on measured V_{cc} voltage. However, this setup can be much simpler with an 8-bit MCU. For example, the Atmel® tinyAVR® and megaAVR® products require you to only initialize the stack pointer, which typically takes four lines of code, prior to commencing coding your application. The choice of clock, brown-out detector, reset pin function, etc. is all pre-programmed into the device. The architecture is also much more straightforward than a 32-bit device with internal registers, peripherals and SRAM all mapped on the same data bus. The peripherals and CPU would normally run at the same frequency, so no peripheral bus configuration is necessary and designers can avoid being concerned about latency in synchronizing between different clock domains.

Performance

When it comes to the desired CPU performance the engineer should consider all use cases. The reality is that many embedded designs do not have high compute requirements. Often very little manipulation of data is required, so balancing the needs of this against the other requirements of power consumption and peripheral interfacing is crucial. For example, a simple thermostat application will spend most of its life in a sleep mode. Every so often it will wake up and measure the temperature and then make a decision to turn a relay on/off or to send an instruction to a host controller. It will then resume sleep. The compute and interface requirements of this application are small, but equally as many other applications such as fire detectors, power tools, flow meters and appliance controls have a similar use profile too.

Efficiency of hardware near functions

Many modern microcontrollers have incorporated some hardware functions that serve to help the CPU operate as efficiently as possible. In Atmel's case, both the 8-bit AVR and 32-bit ARM-based MCU families feature the Peripheral Event System. An event system is a set of hardware-based features that allows peripherals to interact without intervention from the CPU. It allows peripherals to send signals directly to other peripherals, ensuring a short and 100% predictable response time. When fully using the capabilities of the event system, it is possible to configure the chip to do complex operations – with very little intervention from the CPU – saving both valuable program memory and execution time. In the case of detecting a

hardware event it is important to firstly detect the event and then switch control to the desired interrupt service routine (ISR). In these situations CPU speed is not the single determining factor. It is a question of how long, in terms of cycles, does it take to respond to the interrupt, run the ISR and return. As the following example will show, 8-bit devices can be more efficient in handling hardware near actions. Consider a simple example of receiving one byte on the SPI, using an interrupt to detect it and then running a simple ISR routine to read the byte from the SPI peripheral and store it in SRAM.

Action required	8-bit AVR (cycles)	32-bit CM0+ (cycles)	Comment
Detect interrupt and jump to interrupt vector	3	12	
Identify interrupt triggered	0	6	Not needed on AVR since every interrupt has its own interrupt vector. CM0+ core is limited to 32 vectors
Push register to stack	1	0	Not needed on CM0+ since interrupt detection will push 8 registers to stack
Read received byte from SPI to register	1	2	Assuming peripheral and CPU running at same speed. (Best case for CM0+)
Write received byte to SRAM	2	2	
Pop register from stack	1	0	Not needed for CM0+ since interrupt return will pop the 8 registers back
Return from interrupt	3	10	
Cycles in main ( ) until next interrupt can be serviced	1	1	
Minimum number of cycles to receive one byte from SPI using interrupt	12	33	

Table 2 – 8 and 32-bit comparison of cycles required for responding to ISR request and moving one byte of data from SPI peripheral to SRAM

Table 2 compares this example between an Atmel 8-bit AVR device and an Atmel ARM Cortex® M0+based 32-bit MCU. This has been calculated with information available and is based on minimum implementations but engineers should check with their own application since the interrupt detection and return from interrupt could take more cycles than shown in the above table. Requiring 12 cycles versus 33 cycles equates to having a theoretical maximum SPI bandwidth of 1.67 MB per second (13.3 Mbps) for the 8-bit CPU and a 606 KB (4.8 Mbps) bandwidth for a 32-bit CPU when running at 20 MHz.

The degree of numeric processing can also have an impact on the stack and memory required. Using the Fibonacci algorithm is a particularly good way of testing memory requirements. Since it only uses a local variable, everything needs to be pushed to the stack. When making a comparison between an 8-bit AVR

and an ARM 32-bit CM0+ based device, and using a recursive 15-stage Fibonacci algorithm, the AVR uses a total of 70 bytes of stack, including 30 for return stack (15 calls deep). The ARM-based device uses 192 bytes (60 should be return stack). This means the CSTACK is more than 3 times the size of the 8-bit solution. In more normal C-code you will typically get more of the variables on the stack in a packed format so this is an extreme corner. Saying that you may need 1.5 to 3 times more SRAM for the same 8-bit centric application on a 32-bit versus a native 8-bit device however is a fair estimation.

Power consumption

No MCU article would be complete without investigating static power consumption. This alone may be a key factor in choosing between an 8-bit or 32-bit device, especially for battery-powered applications. Table 3 below illustrates these differences in power consumption between 8-bit and 32-bit devices in both active and static modes. Aggressive manufacturing technologies result in an increased transistor leakage current which roughly doubles with each process generation, and is proportional to the number of gates. Leakage current increases exponentially at higher temperatures that can be easily overlooked when designing a consumer design. Mobile phones and personal media players get taken everywhere, and as we have all found out, temperatures experienced during the summer inside a car can easily climb above 40 degrees C.

The amount of time the microcontroller will spend in active mode versus static mode has a significant contribution to the overall application power budget.

	CPU	Flash	Pin Count	Active[1]	Static[2] Typ @ 25°C	Static[2] Max @ 85°C
Atmel SAM D20	32-bit ARM Cortex-M0+	16 – 256kB	32 – 64	140 µA/MHz	2 µA	85 µA
Freescale Kinetis K20	32-bit ARM Cortex-M4	32 – 160kB	32 – 64	280 µA/MHz	1.9 µA	30 µA
Atmel ATmegaX8PA	8-bit AVR	4 - 32kB	28 - 32	300 µA/MHz	100 nA	2 µA
ST STM8S00X	8-bit STM8	8 - 64kB	20 – 48	230 µA/MHz	4.5 µA	17 µA

[1] CPU active, running code from internal NVM memory at 3V under typical conditions.
[2] CPU off, microcontroller in sleep mode with full SRAM retention at 3V under typical

Table 3 – comparison of static v. active power consumption, measured at 25 degrees C for 8-bit and 32-bit devices.

Naturally the ratio between active and static modes will vary depending on the application requirements. Taking the previous SPI interrupt example (Table 2) and assuming a SPI data bandwidth of 80 kbps, the 8-bit CPU will spend 1.2% of its time in active mode compared to that of the 32-bit which will spend 3.3% in active mode. See Table 4.

Power budget	8-bit AVR	32-bit CM0+
Active mode at 10MHz	1.2% x 3000uA	3.3% x 1400uA
Sleep mode	98.8% x 0.1uA	96.7% x 2.0uA
Average consumption	36.1 uA	48.1 uA

Table 4 – Average power budget for SPI interrupt example from Table 1

Conclusion

When contemplating whether to use an 8- or 32-bit microcontroller for a future design, it may be that it is for an Internet of things (IoT) application. How IoT actually takes shape is provoking a lot of debate but it will certainly challenge engineers to make a detailed appraisal of the MCU required. Wireless connectivity, especially ZigBee, will also be an essential component, but that does not automatically mean that a higher power device is required. There are a number of 8-bit microcontroller products already available that satisfy the need for low levels of processing and wireless connectivity. For example, the Atmel ATmegaRFR2 series provides an IEEE 802.15.4 compliant single-chip 2.4 GHz wireless microcontroller solution that is ideal for battery-powered low cost IoT designs.

Atmel | Enabling Unlimited Possibilities®

Atmel Corporation 1600 Technology Drive, San Jose, CA 95110 USA **T:** (+1)(408) 441.0311 **F:** (+1)(408) 436.4200 | **www.atmel.com**