What next for microcontrollers?

Joseph Yiu - January 05, 2010

The embedded world is constantly changing. You might not have noticed, but if you take a minute to recall what a microcontroller system was like 10 years ago and compare it to today's latest microcontroller systems, you will find that PCB design, component packages, level of integration, clock speed, and memory size have all going through several generations of change.

One of the hottest topics in this area is when will the last of remaining 8-bit microcontroller users start to move away from legacy architectures and move to modern 32-bit processor architectures like the ARM Cortex-M based microcontroller family.

Over the last few years there has been a strong momentum of embedded developers starting the migration to 32-bit microcontrollers and, in this multi-part article, we will take a look at some of the factors accelerating this migration.

In the first part of this article we will summarize as to why embedded developers should consider moving to the 32-bit microcontrollers.

The strongest reason for this move is an increase in the complexity of embedded products required by the market and consumer. As embedded products get more and more connected and provide more features, current 8-bit and 16-bit microcontrollers just cannot cope with the processing requirements. Even if an 8-bit or 16-bit microcontroller could cope with the requirements for the current project, it poses a serious risk of limiting future product upgrade opportunities, and the ability to re-use code across developments.

The second common reason is that embedded developers are starting to be aware of the benefits of migrating to 32-bit microcontrollers. Not only do 32-bit microcontrollers provide over ten times the performance but the migration also allows a reduction in power consumption, smaller program size, faster software development time as well as better software reusability. The details of these advantages will be covered in subsequent parts of this article.

Another reason is the choice, range and availability of ARM based devices. Nowadays more and more microcontroller vendors are providing ARM based microcontrollers. These products provide a wider choice of peripherals, performance, memory size, packages, cost, etc.

In addition, the ARM Cortex-M based processors also have a number of features specifically targeted at microcontroller applications. These features allow ARM based microcontrollers to be used in wide and ever growing range of applications. At the same time, the price of ARM based microcontrollers has reduced significantly over the last five years and there are more and more low cost or even free development tools available for developers.

Choosing an ARM based microcontroller is also a much better investment compared to other architectures. Software code developed for ARM based microcontrollers today can be reused across
the wide range of microcontroller vendors for many years to come. As the ARM architecture is becoming more widespread it is also easier to hire software engineers with ARM architecture industry experience than other architectures. This makes the products and assets of embedded developers much more future proof.

Now we will see how the 32-bit microcontrollers win the race on code size, an area traditionally portrayed as a strong area for 8-bit microcontrollers.

**Size of a instruction**

Many people have the impression that 8-bit microcontrollers use 8-bit instructions and 32-bit microcontrollers use 32-bit instructions.

In reality, many instructions in 8-bit microcontrollers are 16-bit, 24-bits or other sizes larger than 8-bit, for example, the PIC18 instruction size is 16-bit -.

Even for the antiquated 8051 architecture, although some instructions are 1 byte long, many others are 2 or 3 bytes long. The same generally applies to 16-bit architectures, for examples, some MSP430 instructions take 6 bytes (or even 8 bytes for the MSP430X).

The ARM Cortex-M3 and Cortex-M0 processors are based on Thumb-2 technology, which provides excellent code density. With Thumb-2 technology, the processors support the Thumb instruction set which includes 16-bit instructions as well as 32-bit instructions, with the 32-bit instruction functionality being a superset of the 16-bit version. In most cases a C compiler will use the 16-bit version of the instruction unless the operation can only be carried out using a 32-bit version.

![Fig 1: Size of a single instruction in various processors](image)

Within a compiled program for Cortex-M processors, the amount of 32-bit instruction is only a small proportion of the total instruction count. For example, the amount of 32-bit instructions in a compiled Dhrystone program image compiled for the Cortex-M3 is only 15.8 percent of the total instruction count (the average instruction size is 18.53 bits).

For the Cortex-M0 the ratio of 32-bit instructions is even lower at 5.4% percent (the average instruction size is 16.9 bits).

**Efficiency of the instruction set**

The Thumb instruction set used by the ARM Cortex-M microcontrollers is also very efficient. For example, the multiple load instructions, multiple store instructions, stack push and stack pop instructions in ARM based microcontrollers allow several data transfers to be carried out by a single instruction.
Powerful memory addressing modes also allow memory access sequences to be simplified on ARM based microcontrollers. For example, memory can be accessed by register offset, immediate offset, PC related or stack pointer related (useful for local variables) addressing modes in a single instruction. Additional features like automatic adjustment of memory pointers are also available.

All ARM based processors are very efficient at handling 8-bit and 16-bit data. Compact memory access instructions for signed and unsigned 8-bit, 16-bit and 32-bit data are all available. There are also a number of instructions specially included for data type conversions. Overall the handling of 8-bit and 16-bit data in ARM processors is just as easy and efficient as handling 32-bit data.

The ARM Cortex-M based microcontrollers also provide powerful conditional execution features. Apart from the comprehensive choices of branch conditions for signed and unsigned data types, which are available on all ARM based microcontrollers, the ARM Cortex-M3 based microcontrollers also provide conditional execution, and combined compare and branch instructions.

Both Cortex-M0 and Cortex-M3 support 32-bit single cycle multiply operation. In addition, Cortex-M3 based microcontrollers also supports signed and unsigned integer divide, saturation, 32 and 64-bit multiply-accumulate (MAC) operations as well as a number of bit field operation instructions.

**8-bit application myths**

Many embedded developers mistakenly believe that their application only does 8-bit data processing so there is no need to migrate to 32-bit processors. But looking a bit deeper into the C compiler manuals the humble integer is actually 16-bits on 8-bit microcontrollers - each time an integer operation is carried out or, if you access to C library function that requires integer operations, you are processing 16-bit data. The 8-bit processor core has to use a sequence of instructions and more clock cycles to handle the equivalent required processing.

The same situation applies to pointers. In most 8-bit or 16-bit microcontrollers, you need at least 16-bits for an address pointer. This can increase if you are using generic memory pointers in 8051 (due to the extra information required to indicate which memory is referenced to), or using memory bank switching or similar techniques to overcome the 64kbytes memory barrier. As a result, processing of memory pointers can be very inefficient in 8-bit systems.

Since each integer variable in the register bank takes multiple registers, the use of integers in 8-bit microcontrollers also results in more memory accesses, along with more instructions for memory read/write, and more instructions for stack operations. All of these issues greatly increase the size of the program code on 8-bit microcontrollers.

**Example**

So how does all of this compare in a specific benchmarking example? For example, the Dhrystone program compiled for various architectures with size optimization yields the following results:

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>Silicon Labs C8051F320</th>
<th>Cortex-M0</th>
<th>Cortex-M3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loops</td>
<td>Keil uVision 3.8 PK51 5.18</td>
<td>RVDS 4.0-SP2</td>
<td>RVDS 4.0-SP2</td>
</tr>
<tr>
<td>Binary output size (bytes)</td>
<td>3185</td>
<td>912</td>
<td>960</td>
</tr>
</tbody>
</table>

The majority of embedded applications benefit by migrating to ARM Cortex-M based microcontrollers due to the smaller code size, which means that a cheaper device with less memory is required. This reduction in code size is due to much better instruction set efficiency, the smaller
size of instructions as well as the need for handling of 16-bit or larger data in most embedded applications.

The smaller code size advantage in ARM microcontrollers affects performance as well as power and cost. In the next part of this multipart article we will look at how ARM Cortex-M based microcontrollers compare with 8-bit microcontrollers in these areas. **Performance and power consumption**

One reason that makes many embedded developers switch from 8-bit and 16-bit microcontrollers to 32-bit microcontrollers is the need for better performance in their embedded products. Whilst often less obvious and less understood, switching to ARM microcontrollers will also reduce the power consumption and extend the battery life of embedded products. In this part, we will investigate how ARM based microcontrollers compare to other microcontrollers in terms of performance, and how they help in reducing power consumption.

A common way to compare performance of microcontrollers is to use the Dhrystone benchmark. It is free, easy to use and small enough to put into microcontrollers with very small memory size (although it is not the 'ideal' benchmarking suite\(^1\)). The original 8051 had a performance of just 0.0094 DMIPS/MHz.

Newer 8051s have slightly better performance, for example, the Maxim 80C310 device has a DMIPS of 0.027, and the fastest 8051 microprocessor has a claimed Dhrystone performance of 0.1 DMIPS/MHz. This is still a lot slower than ARM Cortex-M based microcontrollers, where the Cortex-M3 processor has a maximum performance of 1.25 DMIPS/MHz and the Cortex-M0 processor achieves 0.9 DMIPS/MHz.

What about other 8-bit and 16-bit architectures? PIC18 has a performance of 0.02 DMIPS/MHz (internal clock), slower than some of the 8051s and the 16-bit products from Microchip are also less than half of the performance of ARM Cortex-M3 based microcontrollers.

**Fig 2: Basic performance comparison using Dhrystone**

**Limitations of 8-bit and 16-bit microcontrollers**

In general, 8-bit microcontrollers are very inefficient at handling 16-bit and 32-bit data. As mentioned above, this includes integers, pointers in C code and C library functions. Each time an integer variable or pointer is processed, a sequence of instructions is required which results in slower performance and larger code size.

Another issue that causes inefficiency in many 8-bit and 16-bit microcontrollers is the limitations of the instruction set and the programmer's model. For example, the 8051 heavily relies on an accumulator (ACC) and data pointer (DPTR) for data transfer and processing. As a result
instructions are required to move data in and out of the ACC and the DPTR and this become a large overhead for code size and execution cycles.

The memory interface itself also limits the performance of 8-bit and 16-bit processors. For example, many 8051 instructions take multiple bytes. Since the program memory interface is 8-bit, it takes a number of read accesses and hence multiple clock cycles to fetch the instruction even though the instruction is a very simple operation.

The performance of 8-bit and 16-bit microcontrollers is further reduced if access is required to more than 64Kbytes of memory. These architectures are designed to work with 16-bit addresses (they use 16-bit program counter, 16-bit data pointers, and instruction set are designed to support 64Kbytes address range). If more than 64Kbytes of memory are required, additional hardware and instruction overhead is required to generate the extra address bits.

For a typical 8051 that needs to access more than 64kBytes of memory, the memory is divided into banks and all bank switching code has to be carried out via bank #0 (a fixed bank). The resulting increase in code size and clock cycle overhead can reduce the efficiency of memory usage. Some 16-bit microcontrollers get around this by having larger program counter or using memory segmentation, but the handling of large address values still requires additional processing and hence reduces the performance and results in larger program code.

What about power consumption? One of the most common concerns about moving to the ARM architecture is whether it will increase the power consumption. If you look at the latest ARM based microcontroller products it becomes clear that the ARM Cortex-M based microcontrollers actually have lower power consumption than many 16-bit and 8-bit microcontrollers.

ARM based processors are designed for low power and implement many low power techniques. For example, the Cortex-M0 and Cortex-M3 processors support architectural defined sleep modes and a sleep-on-exit feature (which allows the processor to return to sleep mode as soon as interrupt handling is completed). **Power consumption in microcontroller**

To understand why Cortex-M based microcontrollers can reduce power consumption of embedded systems a good starting point is to look at what is inside a typical microcontroller product. In modern microcontrollers, the processor core is not the largest contributor to area.

---

![Fig 3: Use of ARM Cortex-M based processors reduces silicon area](here)

To see a bigger version of this graphic click [here](here).
As mentioned earlier, the code density of 8-bit microcontroller is very poor. As a result of this a larger block of flash memory is needed and this increases the overall power consumption. The excellent code density of ARM based microcontrollers enables microcontrollers to make use of smaller blocks of flash memory reducing both power consumption and cost.

**Memory access efficiency**

Using a 32-bit bus reduces the power consumption by reducing the number of memory accesses required. For copying same amount of data in memory, an 8-bit microcontroller requires four times the number of memory accesses and with a higher number of instruction fetches required for the copying operations. Therefore, even with the same memory size, 8-bit microcontrollers consume a lot more power to achieve the same end result.

Instruction fetches in Cortex-M based microcontrollers are also much more efficient than 8-bit and 16-bit microcontrollers because each instruction fetch is 32-bit, allowing up to two 16-bit Thumb instructions to be fetched per cycle and allowing more bus bandwidth for data accesses. For the same length of instruction sequence an 8-bit microcontroller needs to use four times the number of memory accesses, and a 16-bit microcontroller needs to use twice the number of instruction fetches. As a result, 8-bit and 16-bit microcontrollers consume a lot more energy than ARM based microcontrollers.

**Reducing power consumption by lowering the operating frequency**

The high performance of 32-bit microcontrollers allows the power consumption to be reduced by running the application at a much lower clock frequency. For example, an application running at 30MHz on an 8051 could be run on an ARM Cortex-M3 based microcontroller at just a 3MHz clock frequency and still achieve the same level of performance.

**Reducing power consumption by reducing active cycles**

Alternatively, by making use of ARM based microcontroller sleep modes, it is possible to further reduce power when the processing of a task has completed. Cortex-M based microcontrollers have much higher performance when compared to 8-bit and 16-bit microcontrollers so it’s possible to complete a task and enter the sleep mode much faster thereby reducing the overall number of active cycles of the system.

![Cortex microcontrollers can lower system power consumption by reducing active cycles](here)
ARM Cortex-M based microcontrollers provide the best energy efficiency and much higher performance when compared to 8-bit and 16-bit microcontrollers. ARM based processors are designed for high energy efficiency and applications can make use of the high performance advantages to reduce power consumption in a number of ways. **Software development**

Any microcontroller is simply a piece of hardware until it is programmed with application software. In this part of the article we will look into the various aspects of application development for ARM Cortex-M based microcontrollers. Software development for ARM Cortex-M based microcontrollers can be much easier than for 8-bit microcontroller products. Not only is a Cortex-M based processor fully C programmable, it also comes with various enhanced debug features to help locating problems in software. There are also plenty of examples and tutorials on the internet, including many from ARM based microcontroller vendor's websites, alongside any additional resources included in microcontroller development kits.

**Porting software from an 8-bit or 16-bit microcontrollers to ARM**

ARM Cortex-M based microcontrollers often have more registers in the peripherals when compared to simple 8-bit microcontrollers. The peripherals in ARM based microcontrollers usually come with more features and therefore more programmable registers are available. But don’t worry, ARM based microcontroller vendors provide device driver libraries to help in the setup of the peripherals with just a few function calls.

Compared to most 8-bit architectures and 16-bit architectures, ARM based microcontroller programming is much more flexible. For example, there is no hardware stack limitation, functions can be accessed recursively (local variables are stored in the stack rather than in static memory locations), and there is no need to worry about saving of special registers value in interrupt handlers as this is handled by the processor during interrupt entry. For example, for the MSP430 you might need to disable interrupts during multiplication processes, whilst on the PIC you might need to save table pointers and multiplier registers in interrupt handlers.

It is useful to be aware that the correct use of data types for an architecture can make a big difference on code size and performance - the sizes of some data types are different between ARM based microcontrollers and 8-bit / 16-bit microcontrollers.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>8-bit / 16-bit microcontrollers</th>
<th>ARM based microcontrollers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>16-bit</td>
<td>32-bit (or use short int for 16-bit)</td>
</tr>
<tr>
<td>enum</td>
<td>8-bit/16-bit</td>
<td>8-bit/16-bit/32-bit</td>
</tr>
<tr>
<td>Pointer</td>
<td>16-bit or more</td>
<td>32-bit</td>
</tr>
<tr>
<td>Double precision floating point</td>
<td>32-bit (use single precision)</td>
<td>64-bit (or use float for 32-bit)</td>
</tr>
</tbody>
</table>

If an application relies on the size of the data type, for example, by expecting an integer to overflow at a 16-bit boundary, then the code needs to be modified for optimized running on ARM based microcontrollers.

Another impact of data size differences is the size of a data array. For example, an integer array in ROM in an 8-bit microcontroller application may be defined as:

```c
const int mydata = { 1234, 5678, };```
This should be changed to the following for running on ARM based microcontrollers to avoid an unnecessary increase in ROM size:

```
const short int mydata = { 1234, 5678, };
```

The difference in floating point instructions can also lead to slight differences in calculation results. Due to the limitations of 8-bit and 16-bit microcontroller performance, when double is used, it is handled as single precision (32-bit). In ARM based microcontrollers, the double data type is 64-bit, hence 32-bit floating point (single precision) should use float data type instead. This difference also affects math functions. For example, the following code from Whetstone will generate double precision math functions on ARM based microcontrollers:

```c
X = T * atan(T2 * sin(X) * cos(X) / (cos(X+Y) + cos(X-Y) - 1.0));
```

For single precision only, the program code should be changed to

```c
X = T * atanf(T2 * sinf(X) * cosf(X) / (cosf(X+Y) + cosf(X-Y) - 1.0F));
```

Debug For some users, one of the key requirements for selecting a microcontroller is the debug support. ARM Cortex-M based microcontrollers support comprehensive debug features including hardware breakpoints, watch points, register accesses, and on the fly memory access. The debug connection can be based on either JTAG or a Serial Wire protocol (two signals), and standardized Cortex debug connector arrangements make it easy for target boards to be connected to debug hosts.

![Fig 5: Debug features on the Cortex-M microcontrollers](image)

For users of Cortex-M3, additional debug features are available via trace support. The basic Cortex-M3 based processor supports selective data trace, event trace, exception trace and a text based output channel (instrumentation trace). The trace data can be collected by a single pin interface called Serial Wire Output, which shares the same JTAG/Serial Wire connector with the debug host connection. This allows useful information about the program execution to be captured by low cost debug hardware without the need for additional trace hardware.

Many Cortex-M3 based microcontrollers also support the Embedded Trace Macrocell (ETM) which extends the trace support to include full instruction trace. This feature allows application code
execution to be analyzed in detail, and allows code profiling to be carried out. Due to the similarity between the Cortex-M0 and Cortex-M3, it is possible to develop and debug an application on a Cortex-M3 with instruction trace, and then port the application to Cortex-M0 with just a small amount of modification.

![Fig 6: Trace features provide high visibility](here)

To see a bigger version of this graphic click [here](#).

One of the most important advantages of using ARM based microcontrollers is the availability of choices. Cortex-M based microcontrollers are available from a growing number of microcontroller vendors, with different peripherals, interfaces, memory sizes, packages, and frequency ranges. Compiler suites range from free, or low cost suites, to professional compiler suites with many advanced features. There is also an increasing range of support from embedded OS, codec, and middleware vendors. **Software portability**

ARM Cortex-M based microcontrollers also provide high levels of software portability. Although there are multiple microcontroller vendors, each providing their own device driver libraries, and there are multiple C compiler vendors providing compiler suites, software can be ported easily via the Cortex Microcontroller Software Interface Standard (CMSIS).

CMSIS is included in microcontroller device driver libraries from many microcontroller vendors. It provide software interfaces to core functions, core registers and provides standardized system exception handler names. Software developed with CMSIS can be ported between different Cortex-M based microcontrollers easily and allows embedded OS or middleware products to support multiple vendors and multiple compiler suites at the same time. It also helps safeguard the investment on software development by providing better software reusability.

**Cost of migration**

The technology enhancements that have taken place in recent years have enabled the cost of ARM Cortex microcontrollers to be on par with that of 8-bit and 16-bit microcontrollers. This is also caused by the increasing user demands for high performance, feature rich, microcontrollers, which greatly increases production volume and brings down unit costs.
A number of low cost development suites and free development packages are available for ARM Cortex-M based microcontrollers. At the same time product development time can be reduced because Cortex-M based microcontrollers are easy to use and have higher performance, hence it can lower overall product development cost.

Migrating from 8-bit microcontrollers to ARM Cortex-M based microcontrollers can provide much better performance and allow sophisticated software to be development at low cost. It can also reduce the power consumption and code size. The same level of benefits are not seen when migrating to 16-bit architecture or alternative 32-bit architectures.

Migrating from 8-bit to 16-bit architectures can only solve part of the limitations seen with 8-bit microcontrollers. 16-bit architectures have the same inefficiency issue with handling large memory size (> 64k bytes) and are usually based on proprietary architectures, which limit the choice of devices and software portability. Other 32-bit microcontroller architectures are lagging behind in terms of interrupt features, energy efficiency, system features and software support. In general, migration to ARM gives the most benefits including lower cost and future proofing.
Whatever your application requirements are, you can find a suitable ARM Cortex-M microcontroller product easily. And in case you need to enhance your product for more features and higher performance or lower power, the architecture compatibility advantage of ARM Cortex-M processors also allows you to move between different ARM microcontroller products easily.

Fig 8: The Cortex-M microcontrollers cover all aspects of deeply embedded system requirements.

Looking into the future, with more Cortex-M based microcontroller products becoming available, more and more embedded projects will migrate to ARM. In the long term, it is clear that the ARM Cortex-M based microcontrollers are the de facto standard for microcontrollers.

A processor design engineer at ARM Ltd., Joseph Yiu works on a variety of Arm-based SoC projects, as well as the design of IP blocks including AMBA Development Kits, PrimeCells, CoreSight. He most recently has been working on Cortex-M3 development.

References

"