

8位MCU和32位MCU的使用案例

本文比较了8位元MCU和32位元MCU的使用案例，可作为如何选择这两种MCU架构的指南使用。

本文大部分32位元范例将关注于ARM Cortex-M装置，Cortex-M在不同MCU供应商产品组合中表现非常相似。由于8位元MCU有很多种架构，所以很难对8位元供应商之间进行类似的产品比较。为了进行比较，本文将使用广泛应用、易于理解的8051 8位元架构。

事实上，「ARM Cortex和8051哪个比较好」不是个逻辑问题，反而像是在问「吉他和钢琴哪个好」？真正要解决的问题是「哪种MCU最能帮助解决目前面临的问题？」。

不同的任务须使用不同的工具，使用者目的是要了解「如何才能善用所拥有的工具」，包括8位元和32位元装置。

对不同的装置进行比较，须要对其进行测量。有很多建构工具可供选择，本文尽量选择一些认为能够进行最公平的比较，且最能代表开发人员真实体验的情境。

以下ARM资料是透过GCC nanoCLibrary和-O3最佳化选项所生成。

此一比较试验并不为任何一种装置的代码最佳化，只是简单实现90%开发人员都会使用的常见代码，并呈现普通开发人员所见到的结果，而不是理想状态下的结果。当然，花费诸多时间、精力和财力去调整8051代码使其表现胜过ARM是可能的，反之亦然，但一开始就选择适合该项工作的最佳工具比费尽心力做最佳化简单多了。

8位元MCU功效持续精进

在开始对架构进行比较前，要注意到并非所有的MCU都是一样，这一点非常重要。

如果将基于ARM Cortex-M0 处理器的现代MCU与30年前的8051 MCU做对比，8051 MCU在性能上当然不会胜出。幸运的是，许多供应商一直对8位元处理器持续投资。

例如芯科实验室 (Silicon Labs) 正持续更新基于8051核心的EFM8 MCU产品线，其效能比原始的8051架构更高，而且开发过程也已实现现代化。所以在许多应用中，8位元核心能够容易弥补比M0 或M3核心不利的地方，甚至在一些方面性能更佳。

开发工具也很重要。现代嵌入式韧体开发需要全功能IDE、现成的韧体库、丰富的范例、完整的评估和入门套件，以及助手应用，以简化硬件设定、资料库管理和量产编程之类的工作。当MCU有了现代化的8位元核心和开发环境时，在很多情况下，这样的MCU将超越基于ARM-Cortex的类似MCU。

以系统规模选择MCU

第一个一般性原则是：ARM Cortex-M核心更适用于较大的系统规模（ $> 64\text{KB}$ 代码），而8051装置适用于较小的系统规模（ $< 8\text{KB}$ 代码）。中等规模的系统可以选择两种方式，这取决于系统要执行的任务。须要注意的是，在大多数情况下，周边组合将会发挥重要作用。如果需要三个UART、一个LCD控制器、四个时脉和两个ADC，使用者可能不会在8位元MCU上找到所有的周边。

易用性与成本/尺寸之比较

对于中等规模的系统来说，使用任何一种架构都可以完成工作。但主要须考量是选择ARM核心带来的易用性，还是8051装置带来的成本和物理尺寸优势。

ARM Cortex-M架构具备统一的储存模式，并在所有常见编译器中支援完整的C99，这使得该架构非常易于写韧体。此外，还可得到一系列资料库和协力厂商代码。

当然，这种易用性的代价就是成本。对于高复杂性、上市时间较短的应用或缺乏经验的韧体开发人员来说，易用性是个重要因素。

比起32位元MCU，8位元MCU的成本颇具优势。使用者经常会发现内建2KB/512B (Flash/RAM) 的小容量8位元MCU，但却很难找到低于8KB/2KB的32位元MCU。在不需很多资源的系统中，储存容量小的MCU能够让系统开发人员获得显著的成本降低。因此，对成本极为敏感或仅需较小储存容量的应用，会更倾向于选择8051解决方案。

8位元晶片通常也具备物理尺寸上的优势。例如Silicon Labs提供的最小32位元QFN封装为4mm×4mm，而基于8051的8位元晶片的QFN封装可小至2mm×2mm。

晶片级封装 (CSP) 的8位元和32位元架构之间的差异较小，但却使成本增加，且组装较难。对于空间严格受限的应用来说，通常须要选择8051装置来满足限制要求。

通用代码/RAM效率易影响MCU成本

8051 MCU成本较低的主要原因之一是是使用Flash和RAM的效率通常比ARM Cortex-M核心更高，这允许系统采用更少资源实现。系统越大，这种影响就越小。

然而，这种8位元储存资源的优势并不总是如此，这一点很重要。在某些情况下，ARM核心会像8051核心一样高效或比其更高效。例如32位元运算在ARM MCU上仅需要一条指令，而在8051 MCU上则需要多条8位元指令。显然，这种代码在ARM架构上有更高的执行效率。

ARM架构在Flash/RAM尺寸较小时的两个主要缺点是代码空间效率和RAM使用的可预测性。首要也是最明显的问题是通用代码空间效率。8051核心使用1位元组、2位元组或3位元组指令，而ARM核心使用2位元组或4位元组指令。

通常情况下，8051指令更小，但这一优势因实际上花费许多时间而受到削弱，ARM核心比8051在一条指令下能做更多工作。32位元运算就是这样一个范例。以实践来说，指令宽度是能在8051上产生适度的更密集代码。

代码空间效率

在含有分散式存取变数的系统中，ARM架构的载入/储存架构通常比指令宽度更为重要。试想讯号量的实现，一个变数需要在代码周围的多个不同位置进行减量（分配）或者增量（释放）。ARM核心必须将变数载入到暂存器，对其进行操作并重新储存，这需要三条指令。另一方面，8051核心可以直接在记忆体位置上进行操作，且仅需一条指令。随着每次对变数完成工作量的增大，由载入/储存而产生的消耗就变得微不足道。但对于每次仅完成一点工作的情况来说，载入/储存能产生重要影响，让8051获得明显的效率优势。

尽管讯号量在嵌入式软体中并不非常见结构，但简单的计数器和标志却广泛应用于控制导向的应用中并发挥相同的作用。许多常见的MCU代码都属于这一类型。

另一个原因是ARM处理器比8051核心具有更多的自由使用堆叠。通常情况下，8051装置针对每次函式呼叫仅在堆叠上储存返回位址（2位元组），透过通常分配给堆叠的静态变数处理大量的任务。在某些情况下，这会产生问题，因为这会造成函数预设不可重入。然而，这也意味着必须保留的堆叠空间很小，且完全可预测，这在RAM容量有限的MCU中至关重要。

举个简单的例子，试验者设计了以下程式，然后测量funcB内部的堆叠深度（图1），发现M0核心的堆叠用了四十八个位元组，而8051核心的堆叠仅用了十六个位元组。当然，8051核心还静态配置了八个位元组的RAM，总共用了二十四位元组。在较大的系统中，这个差异显得微不足道，但是在仅有256位元组的ARM的系统中，这就变得很重要。

```
int main(void){
    funcA(0xA CED);
    while (1);
}

void funcA(uint32_t a){
    uint8_t i, j=0;
    for (i=0; i<3; i++){j = funcB(i, j); }
}

uint16_t funcB(uint16_t testA, uint16_t testB){
    return (testA * testB)/(testA - testB)
}
```

微信号: mcugeek

图1 测量funcB内部堆叠程式示意图

架构细节之考量

假设有基于ARM和基于8051的MCU各一个，配有所需的周边，那么对于较大的系统或需要重点考虑易用性的应用来说，ARM装置是更好的选择。如果首要考量的是低成本/小尺寸，那么8051装置将是更好的选择。本文以下对于每种架构更擅长的应用进行更详细的分析，同时也划分出一般原则。

影响延时因素

两种架构的中断和函式呼叫延时存在很大差异，8051比ARM Cortex-M核心更快。

此外，高阶周边汇流排（APB）配备的周边也会影响延时，这是因为资料必须透过APB和AMBA高性能汇流排（AHB）传输。最后，当使用高频核心时脉时，许多基于Cortex-M的MCU需要分配APB时脉，这也增加了周边延时。

试验者做了个简单的实验，实验中的中断是透过I/O引脚触发的。该中断对引脚发出一些讯号，并根据引发中断的引脚更新标志，之后再量测其部分参数的变化。图2为此次32位元Cortex-M与8051对照实验的程式码与参数比较。

```

---
//Status var
Volatile uint8_t hello;

//ISR
void GPIO_ODD_IRQHandler(void)
{
    GPIO->P[gpioPortA].DOUTSET = 0x03; // T1
    GPIO->P[gpioPortA].DOUTCLR = 0x01; // T2

    if(GPIO->IF & 0x0100){
        hello = 4;
    }
    else
    {
        hello = 5;
    }
    GPIO->IFC = 0xFFFF; // clear interrupt
    GPIO->P[gpioPortA].DOUTCLR = 0x02; //T3
}

//Main loop
while (1)
{
    hello = 0;
    GPIO->P[gpioPortA].DOUTSET = 0x04; //T0
    while(!hello);
    GPIO->P[gpioPortA].DOUTCLR = 0x04; //T4

    for(i=0; i< 0x1000; i++);
}
---

```

Parameter	ARM	8051	
ISR Entry latency (T1-T0)	1.09	0.94	µs
Min pulse width (T2-T1)	0.09	0.08	µs
ISR Execution Time (T3-T1)	1.09	0.74	µs
ISR Exit Time (T4-T3)	0.83	0.57	µs
TOTAL	3.10	2.53	µs

微信号: mcugeek

图2 测试程式码与所得结果参数

8051核心在中断服务程式 (ISR) 进入和退出时显示出优势。但是，随着中断服务程式 (ISR) 越来越大和执行时间的增加，这些延迟将变得微不足道。和既有原则一致，系统越大，8051的优势越小。此外，如果中断服务程式 (ISR) 涉及到大量资料移转或大于8位元的整数资料运算，中断服务程式 (ISR) 执行时间的优势将转向ARM核心。例如，一个采用新样本更新16位元或32位元转动平均 (Rolling Average) 的ADC ISR可能在ARM装置上执行的更快。

控制vs处理

8051核心的基本功能是控制代码，其中对于变数的存取是分散的，并且使用了许多控制逻辑 (If、Case等)。8051核心在处理8位元资料时也是非常有效的，而ARM Cortex-M核心擅长资料处理和32位元运算。此外，32位元资料通道使得ARM MCU复制大的资料更加有效，因为它每次可以移动四个位元组，而8051每次仅能够移动一个位元组。因此，那些主要把资料从一个地方移到另一个地方 (例如UART到CRC或者到USB) 的资料流处理应用更适合选择基于ARM处理器的系统。

来做个简单的实验。试验者编译以下两种架构的函数 (公式1)，变数大小为uint8_t、uint16_t和uint32_t。

```

uint32_tfuncB(uint32_t testA, uint32_t testB){
return (testA * testB)/(testA—testB)
}
|data type | 32bit(-o3) | 8bit |
| uint8_t | 20 | 13 | bytes
| uint16_t | 20 | 20 | bytes
| uint32_t | 16 | 52 | bytes
.....公式1

```

随着资料量的增加，8051核心需要越来越多的代码来完成这项工作，最终超过了ARM函数的大小。在16位元的情况下，代码大小几乎类似，在执行速度上稍优于32位元核心，因为相同代码通常需要更少周期。还有一点很重要：只有采用最佳化的ARM编译代码时，这种比较才有效。未最佳化的代码需要花费几倍长的时间。

这并不意味着有大量资料移动或32位元运算的应用不应该选择8051核心完成。

在许多情况下，其它方面的考量将超过ARM核心的效率优势，或者说这种优势是无关紧要的。举例来说，考虑使用UART到SPI桥接器时，该应用花费大部分时间在周边之间复制资料，而ARM核心会更高效地完成该任务。然而，这也是一个非常小的应用，可能放入到一个仅有2KB储存容量的晶片就足够了。

尽管8051核心效率较低，但它仍然有足够的处理能力去处理该应用中的高资料速率。对于ARM装置来说，可用的额外周期可能处于空闲回圈或「等待中断」(WFI)，等待下一个可用的资料到来。在这种情况下，8051核心仍然最有意义，因为额外的CPU周期是微不足道的，而较小的Flash封装会节约成本。如果使用者要利用额外的周期去做些有意义的工作，那么额外的效率将是至关重要的，且效率越高越可能越有利于ARM核心。这个例子说明，清楚被开发系统所关注的环境中的各种架构优势是何等重要，而作出这个最佳的决定是简单但却重要的一步。

指标为8051特殊优势

8051装置不像ARM装置般统一的储存映射，而是对存取码（Flash）、IDATA（内部RAM）和XDATA（外部RAM）有不 \diamond \diamond 的指令。为了产生高效的代码，8051代码的指标会说明它指向什么空间。然而，在某些情况下，使用通用指标，可以指向任何空间，但是这种类型的指标是低效的存取。例如，将指标指向缓冲区并将该缓冲区资料输出到UART的函数。如果指标是XDATA指标，那么XDATA阵列能被发送到UART，但在代码空间中的阵列，首先需要被复制到XDATA。通用指标能同时指向代码和XDATA空间，但速度较慢，并且需要更多的代码来存取。

专用区域指标在大多情况下能发挥作用，但是通用指标在编写使用情况未知的可重用代码时非常灵活。如果这种情况在应用中很常见，那么8051就失去了其效率优势。

仔细评估了解MCU使用优势

本文已经多次注意到，运算倾向于选择ARM，而控制倾向于选择8051，但没有应用仅仅着眼于计算或控制，该怎样才能定义各种应用，并计算出它的合适范围呢？

本文考量一个由10%的32位元计算、25%的控制代码和65%的一般代码构成的假定应用时，其不能明确的归成8或32位元类别。这个应用也更注重代码空间而不是执行速度，因为其并不需要所有可用的运算效能，并且必须为成本进行最佳化。

成本比应用速度更为重要的事实在一般代码情形下将为8051核心带来些微优势。此外，8051核心在控制代码中有中间等级的优势。ARM核心在32位元计算方面占上风，但是这并非是很多应用所重视的。考量到所有这些因素，这个特殊的应用选择8051核心更加合适。

如果做一个细微的改变，假设该应用更关心执行速度而非成本，那么通用代码不会倾向于哪种架构，并且ARM核心在计算代码中全面占有优势。在这种情况下，虽然有比计算更多的控制代码，但是最后结果将相当均衡。显然，在这个过程中有很多的评估，但是分解应用，然后评估每一元件的技术将能确保使用者了解，在哪种情况下哪种架构有更显著的优势。

功耗影响须多方考量

当查阅资料手册时，很容易根据功耗资料得到哪个MCU更佳的结论。虽然睡眠模式和工作模式电流性能在某些类型MCU上确实更佳，但是这一评估可能会非常容易产生误导。

工作周期（在每个电源模式上分别占用多少时间）始终占据能耗的主导地位。除非两个晶片的工作周期相同，否则资料手册中的电流规格几乎是没有意义的。最适合应用需求的核心架构，通常具备更低的能耗。

假设有一个系统，在装置被唤醒后添加一个16位元ADC样本到转动平均，然后返回到休眠状态，直到获取下一个样本时才又被唤醒。该任务涉及到大量16位元和32位元计算。ARM装置将能够进行计算，并比8051装置更快返回到休眠状态，这会让系统功耗更低，即使8051具备更好的睡眠和工作模式电流。当然，如果进行的任务更适合8051装置，那么MCU能耗由于相同的原因而对系统有利。

周边特性也能够以类似的方式影响功耗。例如，大多数Silicon Labs的EFM32 32位元MCU具备低功耗的UART（LEUART），能够在低功耗模式下接收资料，而却只有两个EFM8 MCU具备此功能。此周边影响电源的工作周期，且在任何须要等待UART通讯的应用中，具备LEUART的EFM32 MCU都比缺乏LEUART的EFM8有利。遗憾的是，除了让MCU供应商的现场应用工程师利用EFM8来解决问题，没有简单的指南来评估这些周边因素。同时，系统设计人员还应了解各种MCU能耗模式下可完成的处理任务。

ARM/8051各有优缺点选择合适工具为开发重点

如果考量到所有这些变数后，仍然不清楚哪些MCU架构是最好的选择，那会如何？这表示，不管是8位元或是32位元都是很好的选择，使用者使用哪种体系架构都不要紧。如果没有明确的技术优势，那么过去的经验和个人喜好在MCU架构决定中也发挥很大的作用。此外，使用者也可以利用这个机会去评估可能的未来专案。如果大多数未来专案更适合ARM装置，那么选择ARM，如果未来项目更侧重于降低成本和尺寸，那么就选择8051。

8位元MCU仍然可以为嵌入式开发人员提供许多功能，并且越来越重视物联网。当开发人员开始设计时，重要的是确保从工具箱中获得合适的工具。实际上的难题是，不能仅仅依赖于PowerPoint资料中的一两个要点就归纳出选择MCU架构的结论。然而，一旦使用者有正确的资讯，并愿意花一点时间来实际试用产品，就不难作出最佳选择。

(mbbeetchina)