WHITE PAPER

Optimizing a processor design for low power control applications

What criteria should be considered for low power, and how is the ARM® Cortex®-M0+ processor designed for such applications

Joseph Yiu

Abstract
ARM® Cortex®-M based microcontrollers are becoming the defacto standard for the next generation of low power control applications.This paper looks at the various criteria to be considered when selecting a processor for low power control applications, and how the Cortex-M0+ processor, the lowest power member of the ARM Cortex-M processor family, was designed to meet these requirements.

Introduction
Many microcontrollers are used in battery powered devices and control tasks are essential for many of these devices. The peripherals being controlled cover a wide range: ADC, DAC, LCD, SD cards, general purpose I/O pins (e.g. LED, buttons), etc. In some cases, the peripheral control processes take up the majority of the execution time in the microcontroller. In order to allow extended battery life in these devices, the Cortex-M0+ processor has been designed with these applications in mind and various features has been developed to optimize the processor for these applications.

Before going into the details of the processor design, let us start by looking into some of the requirements for low power control applications.

The key requirements, highlighted by many customers are:

1. Low power
    - Low operational power
    - Ultra low idle power (sleep mode)
    - Energy efficiency data processing
2. Fast system response time
    - Low interrupt latency
    - Low wake up latency
    - Fast I/O access speed
3. Deterministic behavior
4. Ease of use

Unlike in mobile computing applications, the maximum clock speed and maximum data processing capability are typically less important for low power control applications.

During the design phase of the Cortex-M0+ processor, ARM engineers worked closely with a number of key silicon partners to define these requirements and then specifically optimized the processor design for those areas deemed to be particularly critical for these types of microcontroller applications.

The Cortex-M0+ processor programmer's model is similar in a number of ways to the Cortex-M0 processor. It has:

1. The same instruction set as the Cortex-M0 processor (ARMv6-M architecture)
2. The same number of interrupts; up to 32 (with optional Wakeup Interrupt Controller (WIC) support), and a Non-Maskable Interrupt (NMI) input.
3. The same system level features in the Cortex-M0 processor, such as multiple sleep modes and the 32-bit AHB interface for the system bus.

However, under the hood of the Cortex-M0+ is a completely new 'clean sheet' design, for example, the pipeline is now compressed to just two stages, compared to three stages on the Cortex-M0. In addition, the Cortex-M0+ processor also includes an optional non-privileged execution level, optional Memory Protection Unit (MPU), single cycle I/O interface and various other enhancements that will be discussed throughout this article.

Lower power

The Cortex-M0+ processor has a two stage pipeline.

First stage – contains the instruction fetch, as well as a pre-decoding of the instruction.
Second stage – contains the full decoding of the instruction as well as the instruction execution.

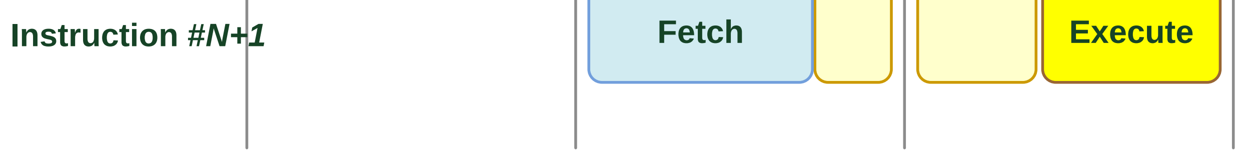

Figure 1: Pipeline of the Cortex-M0+ Processor

Since the instruction set of the ARMv6-M architecture is already very optimized, the merging of part of the decode stage into fetch and execution stage does not add timing delays to the design. The instruction decoder is partitioned very carefully to balance the timing impact at each pipeline stage. As a result, the typical maximum clock frequency very similar to the Cortex-M0 processor design, which has a 3 stage pipeline.

The reduction of the pipeline to two stages brings many advantages in terms of energy efficiency. First, the fewer flip-flops in the processor design reduces the dynamic power. Secondly, by having a shorter pipeline, the branch shadows are reduced, as shown in figure 2, typically resulting in increased overall code execution performance.

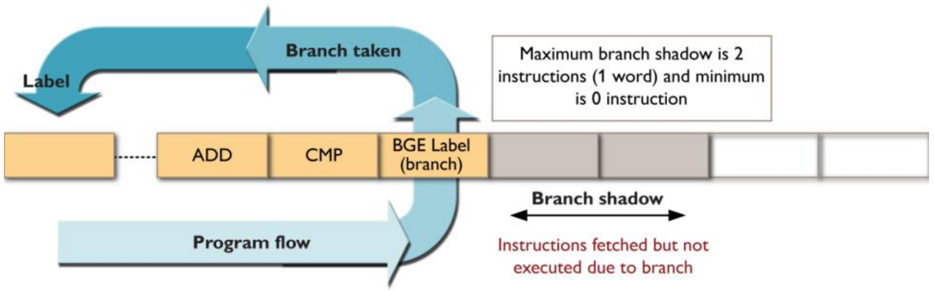
Branch taken
Label
Maximum branch shadow is 2 instructions (1 word) and minimum is 0 instruction
ADD
CMP
BGE Label (branch)
Branch shadow
Program flow
Instructions fetched but not executed due to branch

Figure 2: Shorter pipeline stages in the Cortex-M0+ processor results in smaller branch shadow

The shorter pipeline also has the additional benefit of a lower branch penalty with the result that the CoreMark/MHz figure for the Cortex-M0+ processor is about 9% higher than the Cortex-M0 processor. While higher performance was not necessarily a key target of the processor design, it does help in achieving higher energy efficiency.

The smaller branch shadow helps reduce the processor power consumption, by reducing the amount of unnecessary flash memory accesses, and helps power reduction at the system level. In microcontroller designs, a significant area of the silicon can be used by the flash memory. The reduction in flash memory accesses can make a big difference in reducing overall system level power consumption.

Another optimization implemented in the Cortex-M0+ processor is half word accesses to unaligned branch target. When the processor is branching to an address that is not aligned to a word boundary, the instruction fetch is carried out as 16-bit transfer (figure 3). As a result, the flash memory only needs to enable the read operation for half of the data lanes. Again, this reduces the power consumption at the chip system level.

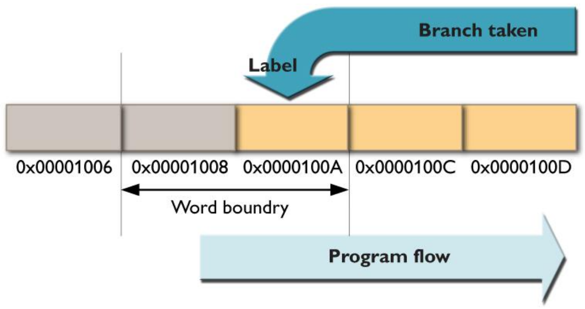

Figure 3: Unaligned branch target addresses are fetched using 16-bit transfers to reduce power

Various power optimization methods have also been used in the processor design to reduce operational power. These included:

- A separate power domain for the debug logic, allowing debug logic to be powered down when a debugger is not attached to the microcontroller.
- Architectural clock gating which enables clock gating cells to be inserted into the design at various critical locations, to maximize power savings.
- Logic is optimized for lowest power rather than smallest area.
- Most instruction fetches are carried out in 32-bit transfers. Since most of the instructions are 16-bit, the processor only needs to fetch instructions every other clock cycle. This effectively reduces the activity of the flash memory by almost 50% thereby saving power.

Overall, the various power reduction measures in the Cortex-M0+ processor result in a best case 30% reduction of operational power when compared to the Cortex-M0 processor. The measurement is based on the running of the Dhrystone loop on a processor design, with a 90nm LP process. This figure does not include the other potential power savings at the system level, as previously described.

Due to the small area of the design, the leakage current of the Cortex-M0+ processor is very small. The area of the Cortex-M0+ is about 12,000 gates in minimum configuration, once again very similar to the Cortex-M0 processor.

The Cortex-M0+ processor supports the WIC (Wakeup Interrupt Controller) feature, which allows the processor to be powered down during deep sleep, and resume almost instantaneously using SRPG (State Retention Power Gating) technology. The WIC is a very small block, closely coupled to the processor, which mirrors the interrupt masking and detection logic when the processor is in deep sleep. Even if SRPG technology is not implemented, the WIC feature can still be used to allow all clocks to be stopped during deep sleep, and still be able to wake up the system when a interrupt request arrive.

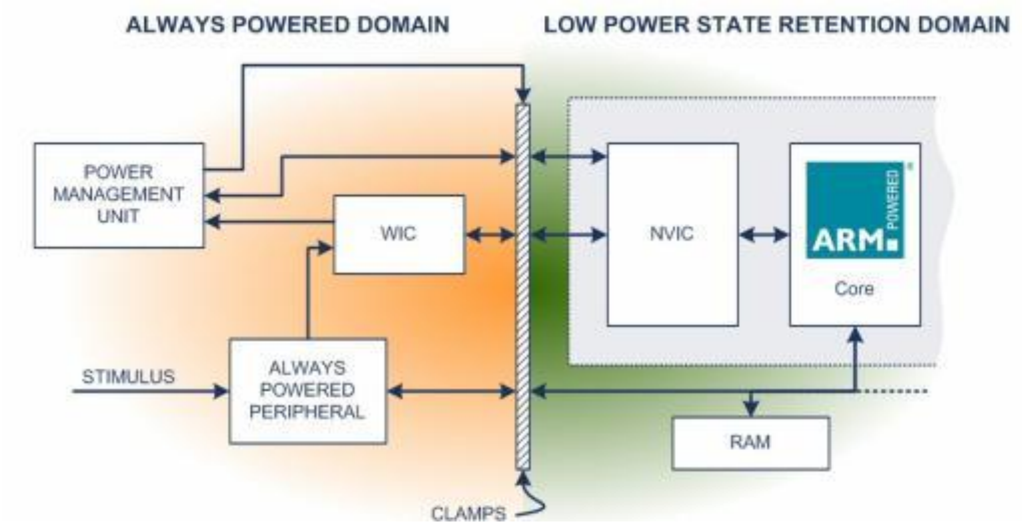

Figure 4: Use of the WIC to mirror interrupt detection function during deep sleep.

When using the WIC feature with SRPG technology, the leakage current of the microcontroller can be reduced to an absolute minimum.

Better responsiveness
The responsiveness of an embedded system can depend on many different factors, including how the software design "waits" for something to happen. For example, there are three common methods to handle this:

- Reading or polling a hardware register to detect an expected value
- Interrupt when an event occurs
- Wake up a system from sleep when an event occurs

In the reading/polling methods, we can generalize the operations as:

- Read a hardware register
- Compare/test the value
- Conditional branch

The quicker these operations are carried out, the better the system responsiveness.

To allow for a faster response when the polling method is used, the Cortex-M0+ processor has an optional single cycle I/O interface. This is a 32-bit generic interface supporting 8-bit, 16-bit and 32-bit data transfers using the standard memory load/store instructions. Microcontroller vendors define which memory range is mapped to this interface during the design stage of the microcontroller, and software developers can access this memory space very easily, using normal pointers in C.

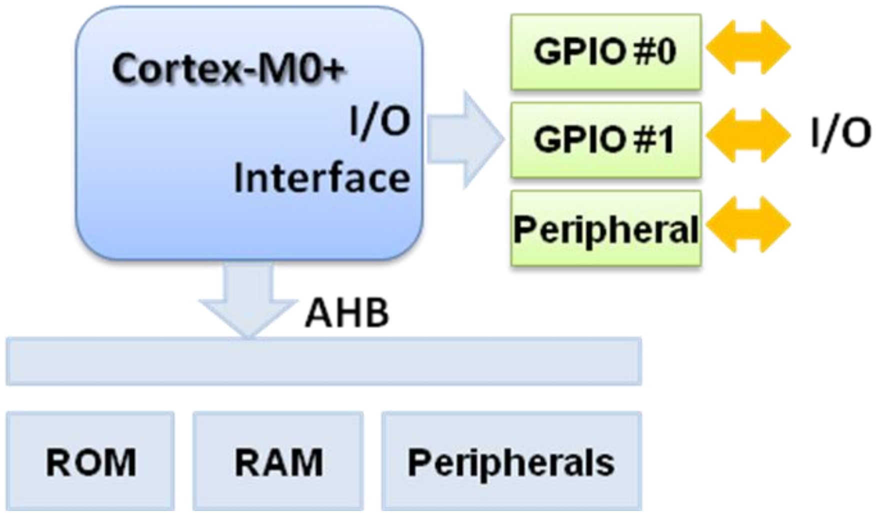

Figure 5: Single cycle I/O port enables faster I/O and peripheral accesses

When combined with the short pipeline stage design of the Cortex-M0+ processor, a polling loop only need 4 cycles to complete one iteration:

```
;   Instruction                          Cycle
    LDR    R0,=PERIPHERAL_BASE_ADDRESS ;  2
    MOVS   R1,#0                       ;  1
loop       ; address label
    LDR    R2,[R0, #REG_X_ADDR_OFFSET] ;  1
    CMP    R2, R1                      ;  1
    BEQ    loop                        ;1 or 2
```

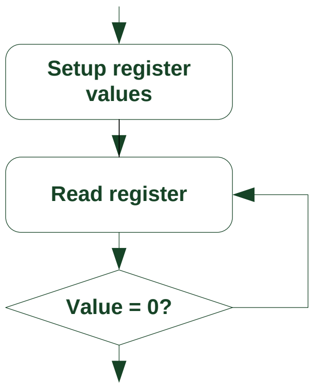

Figure 6: A simple loop only takes 4 clock cycles

The first LDR loads a 32-bit address constant from the program memory (2 cycles), and the second LDR instruction is reading from the single cycle I/O interface (one cycle). The conditional branch instruction only takes 1 cycle if the branch is not taken, and 2 cycles if branch is taken. With the processor running at a relatively low frequency of, for example 10MHz, the loop only takes 0.4us for each iteration.

For less frequent events, it is more common to use interrupts as the handling method. The interrupt latency of the Cortex-M0+ processor is 15 clock cycles, with zero wait state memory systems. This latency value includes the time required to save a number of the registers to the stack and fetch of the exception vector. The interrupt handler can then start interrupt processing immediately as there is no additional hidden software overhead of state saving, or checking of the interrupt controller status.

A less commonly known feature is the Wait-For-Event (WFE). The WFE instruction is usually used in an idle loop and can put the processor into sleep mode if no event is detected. Different from Wait-For-Interrupt (WFI), a WFE can be woken up by an event and the event does not need to be an interrupt request. Such an event could be:

1. Using specific on-chip hardware to generate an event signal feeding into the processor.
2. A new pending of an interrupt: This requires the SEVONPEND (Send Even on Pending) feature to be enabled first. The interrupt does not have to be enabled.

In some cases, the microcontroller vendor may not implement all of the hardware features detailed in point 1, however most peripherals will have interrupt generation capability. If the interrupt was not enabled at the NVIC and SEVONPEND feature is enabled, the processor will wake up from WFE and resume program execution from where it was, and there is no need to handle stacking and enter the interrupt handler. In this arrangement, the wake up latency from the interrupt event to resumption of program execution is only 5 cycles.

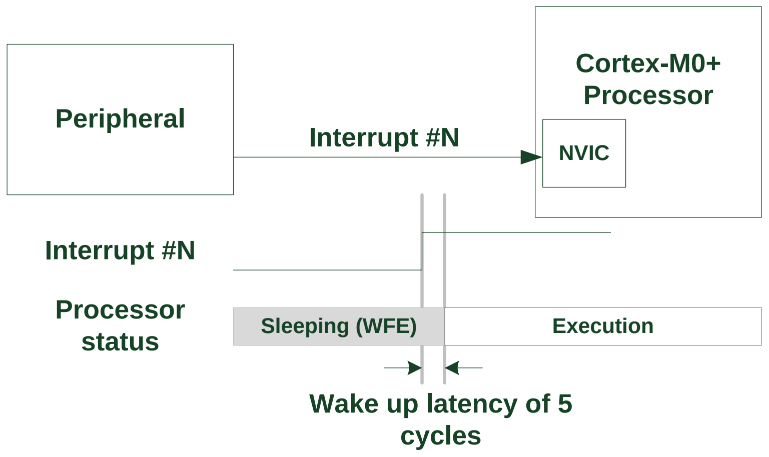

Figure 7: Use SEVONPEND feature to resume program execution from WFE sleep

Deterministic behavior
Cortex-M microcontrollers need to have deterministic behavior in order to make them suitable for real time control tasks. In order to satisfy such requirements, the interrupt controller of the Cortex-M0+ processor, and the execution pipeline, is designed in the following way:

- If an interrupt takes place during a multi-cycle instruction, the instruction is abandoned and restarted after the interrupt service routine (ISR) completed. Therefore we can achieve a fixed interrupt latency of 15 clock cycles, regardless of what the processor was executing at the time the interrupt was flagged.
- Nested interrupts are automatically handled by the NVIC hardware. There is no software overhead in enabling nested interrupts and therefore a high priority interrupt will always get serviced in the fixed latency window.
- Note: If the memory system contains wait states in some parts of the memory system, the interrupt latency could be longer. However, the Cortex-M0+ processor has an optional zero jitter feature to force the interrupt latency to be same for any situation.

In addition, the simplistic software requirement for interrupt handling in the Cortex-M processor family also enables more deterministic behavior as most of the interrupt processing is handled by NVIC.

Ease of Use
Although ease of use if not strictly a technical requirement for selecting a microcontroller in low power control applications, it is important for many users. The architecture of the Cortex-M processor family is designed to be easy to use. For example, almost everything can be programmed in C, including ISRs. Since all the peripherals are memory mapped, all peripherals can be programmed using pointers in standard C language. You will rarely need to use any compiler specific C language extension features, such as special data types for I/O registers, and there is no need to spend time hand optimized program code to get better performance as modern C compilers are very capable of generating the most efficient Thumb instructions.

With 32-bit addressing, it is easy to access the whole of the available memory space without implementing any memory paging. Memory paging makes software development and debugging much more difficult, as well as reduceing the efficiency of the system.

One stepping stone towards easier software development with the Cortex-M processor family is the availability of the Cortex-M Software Interface Standard (CMSIS). In CMSIS, standard APIs are provided in device drivers from the MCU vendors for accessing processor features (e.g. interrupt control). The CMSIS APIs for the Cortex-M processors are compiler independent and can be used on all Cortex-M processors, making software more reusable and portable.

Ease of use also extends to the debug features of Cortex-M processors. In order to make the analysis of a software problem easier, besides using the normal debug features you would expect in any modern microcontrollers such as breakpoints and watchpoints, the Cortex-M0+ processor also optionally supports program trace through a hardware unit called the Micro Trace Buffer (MTB). The MTB connects the processor to the system SRAM, and allocates a small part of the SRAM to store branch information.

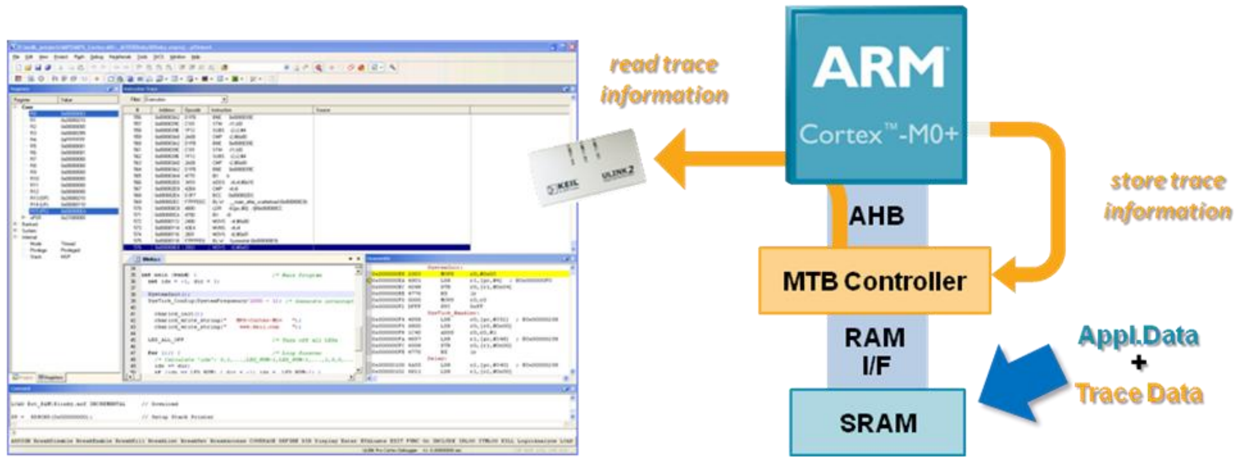

Figure 8: MTB feature provide program trace

The SRAM is shared between application data and trace data, and the MTB can use the allocated SRAM space in circular buffer manner which allows the debugger to extract recent execution history when the processor is stopped. The memory size allocated for trace is configurable by the debugger. Usually only a small area is needed as only branch or interrupt information is captured. The MTB feature does not require any additional pins (as reading of trace information is handled by the Serial-Wire debug or JTAG debug interface) and only needs a very small silicon area. The MTB also supports trigger and one-shot mode features.

Summary

The Cortex-M0+ processor design is optimized for low power control applications. It satisfies various requirements in these applications:

- Low power
- Responsiveness
- Deterministic
- Ease of use

At the same time, the Cortex-M0+ processor delivers outstanding performance in the footprint of an 8-bit / 16-bit processor. With a vast ecosystem already supporting the ARM architecture, including microcontroller vendors, compiler suites, debug tools, and middleware vendors, developing the next low power embedded system for control applications is easy.