

# Enhancing Performance Using an ARM Microcontroller with Zero Wait-State Flash

**Author:**

By: *Wolfgang Schwartz,*  
*Applications Manager, Philips*

**Synopsis:**

*The access times of on-chip Flash memories usually limit the maximum speed of microcontrollers. This article explains how Philips solved this problem for their new ARM core-based LPC2100 microcontroller family, targeted at the 16/32-bit market of embedded real-time applications. Featuring a 128-bit wide zero-wait state flash memory, the parts operate at 60 MHz, reaching 54 Dhrystone MIPs. The use of the ARM7TDMI-S core with Real-Time Monitor and Real-Time Trace enables the customer to take advantage of broad industry support for the core with currently available software applications and tools from third party vendors. Also a real customer application is shown, which demonstrates the optimization of a system using these new microcontrollers.*

Today there are very few electronic gadgets that are not controlled by one or more microcontrollers (MCUs). In high-volume applications, MCUs are embedded in highly integrated specialized ICs. Cost effectively produced in high volume, the versatility of the MCUs also makes them suitable for specialized low volume equipment. Due to the inherent programmability, complex microcontrollers can be adapted to many applications.

As standard products, a vast choice of 4- and 8-bit microcontrollers are available off-the-shelf by many suppliers. For 32-bit MCUs, it's a different story. These microcontrollers, which can combine logic and analog functions as well as different types of memories, are often made in more conservative semiconductor technologies. By using state-of-the-art processes, higher levels of integration are possible, bringing 32-bit architectures like ARM into the reach of standard microcontrollers.

**ARM with on-chip flash memory**

To take full advantage of the processing power of the ARM core, fast program and data memories must be used. While static RAM is usually fast enough to support read- and write-accesses at highest speed within one cycle, flash memory can be significantly slower.

An ARM7™ family core requires one instruction every clock cycle. In 0.18µm technology, typical operation can reach approximately 80 Mhz – that is one 32-bit instruction supplied every 12.5ns. A flash-ROM however has a typical access time of 50ns, which enables a maximum speed of 20 Mhz. Semiconductor manufacturers can cope with this shortfall in a variety of

ways by including certain functions in their microcontrollers:

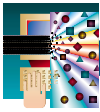
1. Limit maximum clock frequency to about 20 MHz
2. Insert wait-cycles during flash accesses
3. Use an instruction cache
4. Copy the program code from flash to RAM

The first two solutions allow cost-effective systems; however, they waste performance due to the asymmetry between the CPU- and memory speed being insufficiently addressed.

The use of instruction caches increases the over-all performance, yet is met with various disadvantages in microcontroller systems. Next to the substantial chip-area required by the fast RAM, the non-deterministic behaviour of caches might be a problem because, in real-time applications, it is expected that the program flow be predictable if run in similar situations.

In the fourth case, a program can run at highest possible speed after the code is copied into RAM before execution. However, an integration of the RAM is not (economically) feasible, as the RAM would have to mirror the complete flash memory. A static RAM cell is usually formed with six transistors requiring significantly more chip area than the flash cell. Memory technologies optimized for dynamic RAM cells cannot be used reasonably for microcontrollers.

An external RAM requires the use of a 32-bit wide data bus and - depending on the size of the memory - an address bus up to 32-bits wide plus some synchronization



signals. The availability of the data and address busses gives a high flexibility for the size of the system memory, unfortunately cost-effective systems with the lowest possible number of components cannot be realized this way. Furthermore, the number of microcontroller pins not directly necessary for the application is driven high. Limiting the external memory's bus width to 16-bit may reduce costs but works against the goal to increase the performance by faster memory accesses.

The number of microcontroller pins is ideally determined only by the number of I/O-functions required by the application. In the 8-bit world, there are product families that, next to the I/O-pins, have only two additional pins for the power supply, meaning that functions like oscillator and reset, as example, are completely integrated. An example of this configuration is the Philips flash-based LPC900-family. 32-bit controllers are commonly associated with packages far above 100 pins. By applying the 32-bit ARM7TDMI-S core to microcontrollers, on the one hand, Philips intended to boost the calculation power of the processors. On the other hand, the successful low-pin-count concept used in 8-bit was to be extended to 32-bit  $\mu$ Cs (figure 1).



Figure 1: ARM LPC in a tiny LQFP48-package

A solution had to be found that made instructions available from the internal flash memory sufficiently fast to the ARM core. For this purpose, a block called a Memory Accelerator Module (MAM) was developed.

### Memory Accelerator Module

The task of the MAM is to provide the core with the next instruction without delay, whenever an instruction is needed. Usually program and data memories are designed to be as wide as a processor word, such as 32-bits in the case of ARM7 family cores. For cost reasons, 16-bit wide memories are often used in systems with external memory. Of course, the result is a considerable loss of performance. The obvious way of increasing the speed of the system is to widen the flash memory to 128-bits, for example. Each access to the memory will then deliver four 32-bit ARM-instructions or eight 16-bit Thumb® instructions. Nevertheless the CPU still must wait for the first instruction until the memory access is finished. Only then can the next three (ARM) resp. seven (Thumb) instructions be made available without further delay.

To also avoid this delay, the flash memory is split into two banks, again each being 128-bits wide. While 128-bits are read and buffered from one flash bank, previously buffered instructions of the other bank are being executed. Because each bank holds several instructions at one time, it takes several processor cycles before instructions are required from the other bank. This time is used by the MAM to re-load the buffer (Prefetch Buffer) with 128 bits of data from the flash bank currently not engaged. Reading from the rather slow flash memory, several wait states can be added by this without affecting the execution time of instructions.

This simple pre-fetching principle (see figure 2) does the desired job and allows the CPU to operate at four-times higher speed than usually possible with flash - as long as there are no jumps in the program flow. Branches, subroutine-calls and interrupts sometimes break the row of consecutive instructions. In this case, with the exception of very short skips, the content of the Prefetch Buffer of the associated bank is irrelevant.

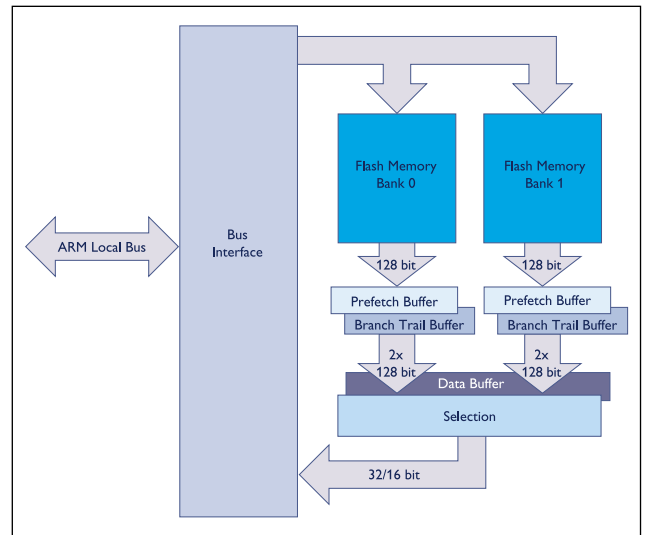


Figure 2: Simplified block diagram of the Memory Acceleration Modules (MAM)

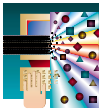
### Program Branches

When the target address falls into the reach of the second bank, the re-loading has been started already speculatively. If the reading of the relevant flash block will be finished within the next cycle, the contents of the second bank's Prefetch Buffer can be accessed without any delay. Otherwise one or a few wait cycles must be added until the reading of the flash is complete.

If the jump goes beyond the reach of the Prefetch Buffer, the CPU has to be stopped to perform wait cycles in any case until data from the "right" target address has been copied into the buffer. Only in this case, the relatively long access time of the flash takes affect. After this, the next instructions can be read at full speed from the buffers until the program flow is interrupted by the next instruction miss again.

A jump can also go backwards. There is a certain probability that the program is executing loops. To prevent the slowing down of such loops by repeatedly reading the instructions from the flash, every memory bank has a second buffer, which holds the last instructions executed. These Branch Trail Buffers are also 128 bits wide. While instructions from one Prefetch Buffer are executed, the Prefetch Buffer of the other bank is already being loaded with new instructions from the flash memory. A loop-back instruction would now require old program code, which is no

*Continued on page 18*



Continued from page 15

longer available in the other Prefetch Buffer. However, before this information was over-written, it was copied into the associated Branch Trail Buffer, which now supplies the next (loop-) instructions without any delay.

### Data areas

The flash memory not only holds program code, there are also areas with data that are never or rarely modified. This can include constants, tables or text strings. Also for this kind of data, a special Data Latch, common to both memory banks, has been implemented. Data accesses to addresses, of which the data is not yet available in the Data Latch, trigger the parallel reading of the next four 32-bit words from the flash. This speeds up sequential accesses, as only after every fourth word a new read sequence has to be initiated. However if addressing is more or less random, the Data Latch usually does not hold the requested data and no acceleration is achieved.

### Duration of real flash accesses

When the MAM is used, the ARM core never reads instructions or data directly from the flash blocks, but rather gets the information from the Prefetch and Branch-Trail Buffers or Data Latch respectively. The real loading of the flash contents into the buffer registers is synchronized with the system clock. Up to 20 MHz data can be read from the flash without delay. At higher system clocks, wait cycles must be inserted. The total number of cycles to be applied to flash accesses can be set between one and seven by the user. A single cycle is sufficient for clocks up to 20 Mhz. Two cycles must be used at speeds up to 40 Mhz (i.e. one wait cycle has to be added). For speeds up to 60 MHz, the maximum clock frequency currently specified, three cycles are required.

### MAM Options

The efficiency of the MAM has been investigated using various benchmark tests. The results depend on the system clock. Since the flash can be read within one cycle up to 20 MHz, no difference can be seen when executing code from RAM or flash - with or without MAM (figure 3). At higher frequencies, the shorter access times of the RAM come into play, and the ARM core continues to run at highest performance. When code is executed directly

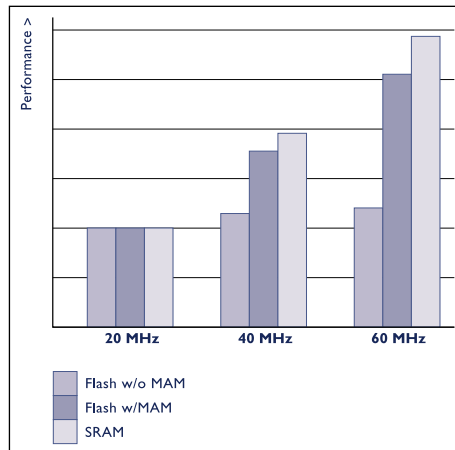


Figure 3: Benchmark results for different program memory implementations

from flash, increasing the frequency above 20 MHz does not have a noticeable effect. The performance is determined primarily by the slow memory accesses. Enabling the MAM, however, significantly boosts the performance, and the core can continue to run virtually at the highest possible speed. Compared to the ideal case, only instruction or data misses cause a hiccup from time to time.

On one hand, the use of the MAM obviously increases the performance at higher clock frequencies. On the other hand, the predictability suffers from the different access times of the flash and the buffers. To suit different requirements, the degree of MAM usage can be selected in three levels:

1. MAM off  
All flash accesses are performed directly without involving the buffers; the preselected number of wait cycles is asserted. The processor's behaviour can be easily predicted - but performance is sacrificed at higher clock frequencies.
2. MAM partially enabled  
Sequential instructions will be fetched from the Prefetch Buffers however even short branches will initiate direct flash accesses. All data requests will, for the sake of predictability, be answered from the flash instead of the Data Latches.
3. MAM fully enabled  
All accesses are preferably serviced from the Prefetch or Branch Trail Buffers or the Data Latches. The flash will only be read directly, if no relevant data is available in the buffers. At highest performance the exact behavior is harder to predict.

### Flash programming

Programming of the flash contents is not affected by the MAM. When single sectors of the flash are erased or programmed, the allocation of the data to either of the two memory banks is fully transparent to the user. A boot block holds the routines with the necessary programming algorithms. These routines can also be called by the user program, enabling in-application programming. A boot loader supports serial programming of the devices.

After reset, control is passed automatically to the boot loader. The boot loader first checks the flash for a valid user program. The decision, whether the flash contains a valid program or not, is done based on the contents of the exception vector table (figure 4). The ARM exception vector table contains a reserved address at 0x14. If this address contains the 2's complement of the checksum of the remaining vectors, the sum of all vectors equals 0. Only if this criterion is fulfilled, the boot loader transfers control to the reset vector contained in the flash vector table.

To prevent old data from being read from the MAM buffer registers after programming a flash sector, the contents of all Prefetch and Branch Trail buffers as well as the Data Latch are marked as invalid at the beginning of the programming cycle. After programming, all data is read directly from the flash until the buffers hold valid information again.

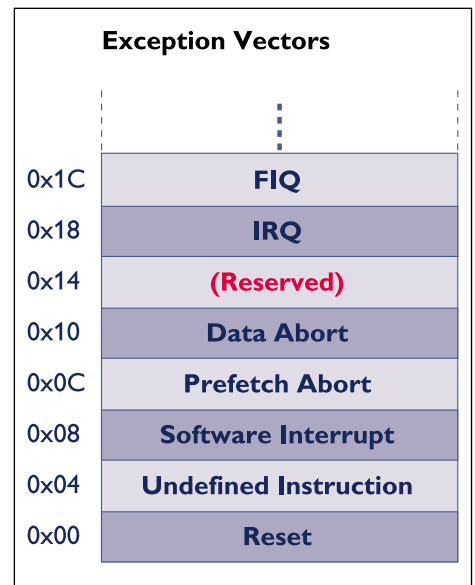
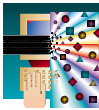


Figure 4: Exception vector table



- Single-chip microcontroller based on the ARM7TDMI-S core
- On-chip memory subsystem with 128 and 256 kilobytes (LPC2124/29) of 0 wait-states Flash and up to 64 kilobytes of SRAM (LPC2106)
- Vectored interrupt controller (VIC)
- EmbeddedICE-RT and ETM (Embedded Trace Macrocell) enable extensive, real-time debug facilities
- Two UARTs, one with full modem interface
- Additional serial interfaces including SPI and I<sup>2</sup>C
- Two 32-bit timers, each with 4 capture/compare channels
- PWM unit with 6 channels
- Real-time clock and Watchdog timer
- Four 10-bit A/D channels (LPC2114/19/24/29)
- Two Full-CAN channels (LPC2119/29)
- High-performance 60 MHz (54 Drystone MIPS)
- Small footprint packages:
  - 7mm x 7mm LQFP-48 (LPC210x)
  - 10mm x 10mm LQFP-64 (LPC21x4/x9)

Table 1: Features of the LPC2100 family

As no flash accesses are possible during programming, the ARM core is kept in a waiting state. If the watchdog function is enabled, the user should ensure that the preset watchdog period is not exceeded.

**Integrated peripheral functions**

Thanks to the MAM, the ARM core can be operated at highest speed even with on-chip flash memory. Because no external memories with their wide address and data busses are required, powerful Low Pin-Count ARM microcontrollers can be developed. The Philips LPC2100-family is the first implementation of an ARM7 family core in 0.18µm flash technology (table 1). This enables the combination of flash with elements of an extensive logic library. The flash blocks are built using a robust 2-transistor NOR flash cell (figure 5). The

reliability of the flash cell is even further improved by involving an error correction. First derivatives offer 128kbyte or 256kbyte of in-system and in-application programmable flash and 16, 32, or 64kbyte of RAM respectively. The flash technology used also permits the easy integration of EEPROM blocks on future derivatives.

The peripheral functions of the first products particularly support serial communication protocols. Next to the Rx/D and Tx/D connections, one of two UARTs additionally make available six modem control signals. In addition, interfaces for the I<sup>2</sup>C and SPI protocols are provided. Two general purpose 32-bit counter/timers each feature four capture/compare channels. Clock and calendar functions can easily be realized with the real-time clock (RTC), while a watchdog timer can supervise the program flow. Other types feature a fast (2.44µs) 10-bit A/D-converter with four inputs and two Full-CAN channels.

A Vectored Interrupt Controller (VIC) complements the ARM core in order to handle any interrupt sources. The high frequency processor clock is derived from a standard crystal using an on-chip phased lock loop (PLL).

Despite the extra chip area needed, Philips has chosen to implement the ARM7 family blocks — EmbeddedICE Logic and Embedded Trace Macrocell – along with the software module RealMonitor, thereby equipping the chips with the optimal prerequisites to make system development as comfortable as possible, even though the Low Pin-Count concept does not enable direct access to the address and data buses, which are not brought outside. Emulators can communicate directly with the core using the JTAG-interface. Trace information is collected internally and highly compressed before it is transferred to the tools outside.

**Application example**

The combination of the 32-bit ARM7 family core's calculation power with a small package, and the available serial interfaces, enabled Innovada to realize

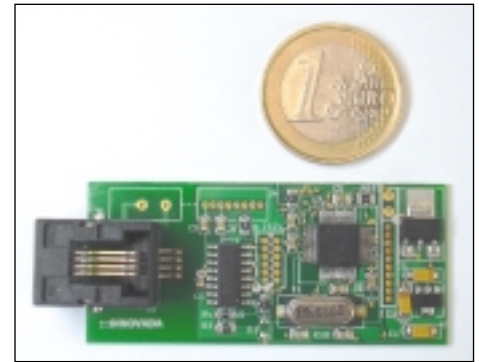


Figure 6: Sample application: Innovada Modem-Netchip Modul

a communication module much smaller and more cost-effective than in a previous solution. The module is designed for integration into equipment that transmits measurement values or is remotely controlled via the Internet, for example. This Modem-NetChip module is comprised of a software modem and a new, innovative phone-interface (DAA). The computing power of the ARM core allows the software modem and TCP/IP-support to be run on the LPC2100, without a separate DSP required. Possible applications include point of sale (POS)-terminals, security systems and a variety of other embedded systems. The RJ11-connector and the coin in figure 6 demonstrate how tiny the complete module is. Innovada also licenses the modem software separately for integration into systems.

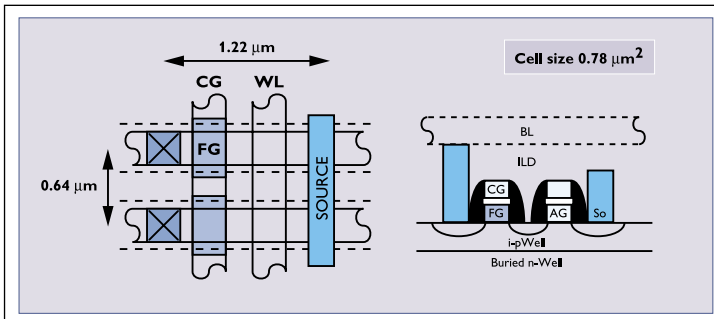


Figure 5: Two transistor NOR Flash cell

- Philips Semiconductors (general): <<http://www.semiconductors.philips.com>>
- Philips Semiconductors (Microcontrollers): <<http://www.semiconductors.philips.com/markets/mms/products/microcontrollers/index.html>>
- Overview LPC2100-Familie: <[http://www.semiconductors.philips.com/markets/mms/products/microcontrollers/key\\_solutions/32bit/index.html](http://www.semiconductors.philips.com/markets/mms/products/microcontrollers/key_solutions/32bit/index.html)>
- Data sheet LPC2100: <[http://www.semiconductors.philips.com/acrobat/datasheets/LPC2104\\_2105\\_2106-02.pdf](http://www.semiconductors.philips.com/acrobat/datasheets/LPC2104_2105_2106-02.pdf)> or <[http://www.semiconductors.philips.com/acrobat/datasheets/zip/LPC2104\\_2105\\_2106-02.zip](http://www.semiconductors.philips.com/acrobat/datasheets/zip/LPC2104_2105_2106-02.zip)>
- Development Tools LPC2100: <[http://www.semiconductors.philips.com/markets/mms/products/microcontrollers/support/development\\_tools/lpc2100/](http://www.semiconductors.philips.com/markets/mms/products/microcontrollers/support/development_tools/lpc2100/)>
- Innovada <<http://www.innovada.com/>>