Application Report
SPNA143–November 2011

Important ARM® Ltd Application Notes for TI Hercules™ ARM Safety MCUs

Brian Fortman

ABSTRACT

The purpose of this application report is to offer a convenient collection of references to application notes posted by ARM Ltd that apply to TI's Hercules ARM Safety Microcontrollers. The document contains links directly to the ARM documents in the ARM On-Line InfoCenter as well as re-prints the ARM introduction chapter to help you better determine if you want to investigate further.

Contents

1 Fixed Point Arithmetic on the ARM

Application Note 33

This application note describes how to write efficient fixed point arithmetic code using the ARM C compiler and ARM or Thumb assembler. Since the ARM core is an integer processor, all floating point operations must be simulated using integer arithmetic. Using fixed point arithmetic instead of floating point will considerably increase the performance of many algorithms.

This document contains the following sections:

- Principles of Fixed Point Arithmetic on page 3 describes the mathematical concepts needed for fixed point arithmetic.
- Examples on page 5 gives examples of writing fixed point code for signal processing and graphics processing, two of the most common uses.
- Examples on page 5 gives examples of writing fixed point code for signal processing and graphics processing, two of the most common uses.
- Programming in Assembler deals with assembler examples

2 TCP/IP RTOS Integration Case Study

Application Note 76

This application note is intended as a supplement to the Porting TCP/IP Programmer's [ARM DUI 0079B]. It gives more detail on some of the issues that arise when targeting an RTOS, and gives a concrete example of such an implementation on µC/OS, a small RTOS kernel.

3 Migrating from MIPS to ARM

Application Note 235

The purpose of this document is to highlight areas of interest for those involved in migrating software applications from MIPS to ARM platforms. No attempt is made to promote one architecture over the other, merely to explain clearly the issues involved in a decision to migrate an existing software application from one to the other.

Familiarity with the MIPS architecture is assumed and corresponding ARM features are explained.

The ARM architecture is supported by a wide range of technology, tools and infrastructure available from a large number of partners in the ARM Connected Community. Pointers to these resources are given where appropriate, although ARM's own supporting technology is highlighted.

4 Migrating from 8051 to Cortex Microcontrollers

Application Note 237

The ARM Cortex™-M range of microcontroller cores are high performance, low cost and low power 32-bit RISC processors. Currently, the range includes the Cortex-M3, Cortex-M4 and Cortex-M0 cores (the Cortex-M1 is similar in functionality and is targeted at implementation in FPGA devices). Cortex-M processors differ from other processors in ARM's range in that they execute only Thumb-2 instructions and do not support the ARM instruction set. They are based on the ARMv7-M architecture and have an efficient Harvard architecture 3-stage pipeline core. They also feature hardware divide and low-latency interrupt service routine (ISR) entry and exit.

As well as the CPU core, the Cortex-M processors include a number of other components. These include a Nested Vectored Interrupt Controller (NVIC), an optional Memory Protection Unit (MPU), Timer, Debug Access Port (DAP) and optional trace facilities. They have a fixed memory map.

In this document, only the standard 8051 architecture is referenced. There are several extended versions of the architecture (e.g., 8052) that support additional features (e.g., extra status flags, ability to address more memory, etc.). These are not considered.

A comparison of the 8051 is made primarily with the Cortex-M3 devices as these form the bulk of the microcontrollers available using Cortex-M cores. This document also contains an additional comparison with the Cortex-M0, which is in many respects a subset of the Cortex-M3, providing a lower cost solution, albeit at a lower performance point.

5 Coding for the Cortex-R4(F)

ARM Whitepaper AN240

ARM Cortex-R real-time processors offer high-performance computing solutions for deeply embedded systems with demanding real-time response constraints. This document introduces the main features of the Cortex-R4 and Cortex-R4F processors. It also discusses C coding considerations when porting code targeted for an ARM946E-S™ to the Cortex-R4.

6 Migrating From Power Architecture to ARM

ARM Application Note 245

The purpose of this document is to highlight areas of interest for those involved in migrating software applications from Power Architecture to ARM platforms. No attempt is made to promote one architecture over the other, merely to explain clearly the issues involved in a decision to migrate an existing software application from one to the other.

Familiarity with the Power Architecture is assumed and corresponding and additional ARM features are explained.

The ARM architecture is supported by a wide range of technology, tools and infrastructure available from a large number of partners in the ARM Connected Community. Pointers to these resources are given where appropriate, although ARM's own supporting technology is highlighted.

7 Dhrystone Benchmarking for ARM Cortex Processors

ARM Application Note 273

The first version of the Dhrystone benchmark program was written in 1984 by Reinhold Weicker to measure the integer performance of processors and compilers. It became popular because it was very simple, could be quickly compiled and run and computed a single result that was more representative than the MIPS numbers reported by processor vendors. In 1988, version 2 of Dhrystone was released in the Ada, C and Pascal languages.

Today, Dhrystone is no longer seen as a representative benchmark and ARM does not recommend using it. In general, it has been replaced by more complex processor benchmarks such as SPEC and CoreMark.

ARM quotes figures for the C version of Dhrystone 2.1.

Dhrystone performance is calculated using the formula:

Dhrystones per second = number of runs / execution time.

For the result to be valid, the Dhrystone code must be executed for at least two seconds, although longer is generally better, and ARM recommends at least 20 seconds.

This application note describes how to build and run Dhrystone on both bare-metal and OS-hosted systems. If you have any problems or questions about benchmarking ARM Cortex processors, contact ARM support.

8 Managing Memory Protection Performance Concerns in Cached Cortex R4/R5 Processors

Application Note 296

Many embedded systems operate with a multitasking operating system that provides a facility to ensure that the task currently executing does not disrupt the operation of other tasks. System resources and the code and data of other tasks are protected. The protection system typically relies on both hardware and software to do this. In a system with no hardware protection support, each task must work in a cooperative way with other tasks and follow rules.

In contrast, a system with dedicated protection hardware will check and restrict access to system resources, preventing hostile or unintentional access to forbidden resources. Tasks are still required to follow a set of operating system (OS) rules, but these are also enforced by hardware, which gives more robust protection.

ARM provides many processors with this capability, using either a memory protection unit (MPU) or a memory management unit (MMU). This applications note is about MPU-based processors. These provide hardware protection over a number of software-programmed regions, but stop short of providing a full virtual memory system with address translation, which requires an MMU.

An ARM MPU uses regions to manage system protection. A region is a set of attributes associated with an area of memory. The processor core holds these attributes in CP15 registers and identifies each region with a number. A region’s memory boundaries are defined by its base address and its size. Each region possesses additional attributes which define access rights, memory type and the cache policies. Because peripherals are memory-mapped in ARM systems, the same protection mechanism is used for both system peripherals and task memory.

In the Cortex-R4 and Cortex-R5 processors, the presence of the MPU is optional although generally included. If present, there may be either 8, 12 or 16 such regions (defined by the hardware implementer at RTL configuration stage). The smallest length (size) of a region is just 32 bytes. If a region is of 256 bytes or more, it may be divided into eight sub-regions. Although the Cortex-R4 and Cortex-R5 processors have a Harvard view of memory, the regions are common to both instruction and data accesses. However, it is possible to use the *Execute Never (XN)* attribute to disallow instructions execution from a peripheral or data region.