Micromega Corporation

Application Note 41

Comparing WinAVR math and uM-FPU V3.1 math

Introduction

The WinAVR C compiler includes a standard C math library (*libm.a*) that provides support for floating point math. This application note compares the WinAVR math library with various alternative methods of using the uM-FPU V3.1 floating point coprocessor to provide Atmel AVR floating point math support. These methods are as follows:

Replacement of the Math Library

The standard C math library, *libm.a*, contains all of the floating point support routines. The *libm.a* library can be replaced with the *libm_fpu.a* library to cause the WinAVR program to use the uM-FPU V3.1 chip for floating point operations. An `fpu_reset` call must be done at the start of the program to initialize the FPU, but otherwise no additional code changes are required.

Using In-line C Code

This method sends instructions and data to the uM-FPU V3.1 chip using the low-level FPU support routines. The math operations are implemented using the FPU instruction set.

Calling User-Defined Functions

This method stores the FPU code in a user-defined function. The function is stored in Flash memory on the uM-FPU V3.1 chip. This allows for minimum execution times.

The execution times and code space requirements for a simple benchmark are compared for the standard WinAVR math routines and each of the methods using the uM-FPU V3.1 chip described above.

Benchmark Code

The benchmark code used for these comparison consisted of a series of simple operations as follows:

```
a = 1000.0;
b = 0.001;
c = (a + b) / (a - b) * 100.0;
c = sin(a)
c = cos(a)
c = tan(a)
c = asin(b)
c = acos(b)
c = atan(b)
c = atan2(a, b)
c = exp(b)
c = log(b)
c = log10(b)
c = pow(a, b)
c = sqrt(b)
```

uM-FPU V3.1 Interface

Execution Times

The Atmel AVR interfaces with the uM-FPU V3.1 chip using the hardware SPI port running at 4 MHz. The FPU support routines are contained in the file *fpu_hw.S*. They provide the support for writing data and instruction to the FPU, and for reading the results. The data transfer times are included in all of the execution times shown below.

Execution time for the WinAVR math library routines are directly related to the clock speed of Atmel AVR since the math operations are implemented in software. If the clock speed is doubled, the execution time is reduced by a factor of two.

Execution times for the FPU routines remains relatively constant for Atmel AVR clock speeds above 4 MHz since the hardware SPI always transfers data at 4 MHz. The transfer times and FPU execution times remain constant, with only the call overheads being affected by the Atmel AVR clock speed.

Figures 1 and 2 show a comparison of the results of running the complete benchmark at both 8 MHz and 16 MHz.

Figure 1 - Benchmark Timing with Atmel AVR running at 8 MHz

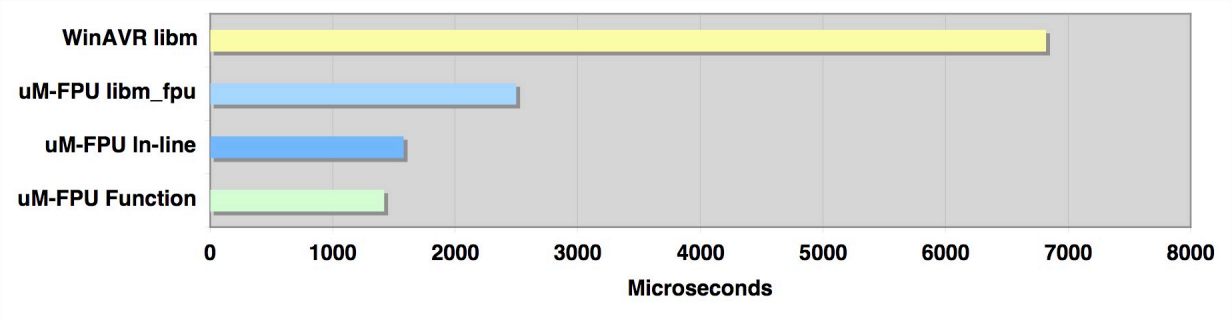
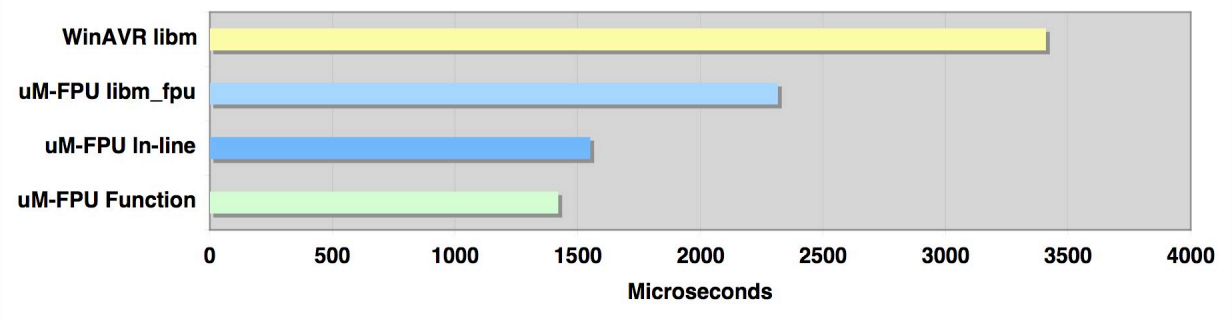


Figure 2 - Benchmark Timing with Atmel AVR running at 16 MHz



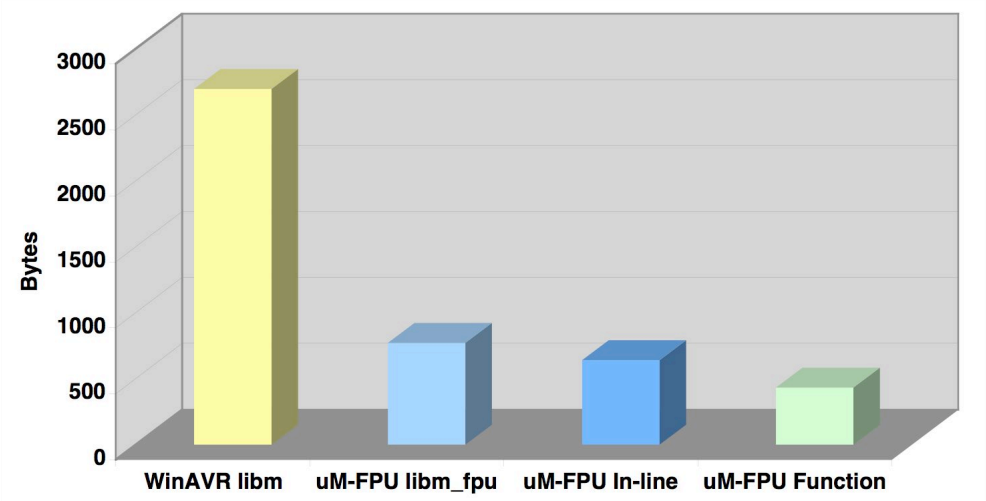
Note: Execution times for the individual functions in the benchmark are shown in Appendix A.

Code Space

Using the uM-FPU for floating point math significantly reduces the code space required on the Atmel AVR (see figure 3). This is important, since code space is often a critical resource in Atmel AVR projects.

When using the WinAVR math library, a significant amount of code is required for basic floating point support, and each new math function requires additional code. Using the uM-FPU, a fixed amount of code is required for the FPU interface routines, but no additional code is required for each math functions. The code space overhead for calling a math function is the same when using the *libm_fpu.a* replacement, and much less when using FPU in-line code or function calls.

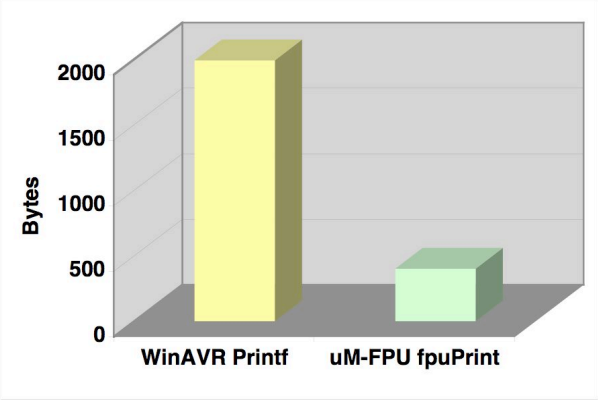
Figure 3 - Code Space used by Benchmark



Printing Floating Point Values

Using the WinAVR `printf` function to display floating point numbers requires a large amount of code space. The default `printf` library is integer only, and must be replaced with a much larger version to print floating point values. The uM-FPU has an `FTOA` instruction that converts floating point values to formatted strings, so no additional code is required. The `fpuPrint.c` file contains various routines for sending formatted strings to the UART (stdout).

Figure 4 - Code Space used to print Floating Point values



Summary

Significant reduction in code space can be achieved by using the uM-FPU V3.1 chip for floating point math on the Atmel AVR. Execution time is improved by at least a factor of two for an Atmel AVR running at 16 MHz and by at least a factor of four when running at 8 MHz. Further gains can be achieved by using the FPU as a true coprocessor. The FPU can perform math calculations while the Atmel AVR is performing other tasks.

Additional benefits can be realized by using some of the specialized features of the uM-FPU V3.1 chip.
e.g.

- using the FPU to read data from a GPS receiver and perform navigational calculations
- using the matrix operations to perform 2D and 3D transformations
- using the FFT instruction for Fast Fourier Transforms
- using the 12-bit A/D channels to collect and pre-scale analog values

Further Information

See the Micromega website (http://www.micromegacorp.com) for additional information regarding the uM-FPU V3.1 floating point coprocessor, including:

uM-FPU V3.1 Datasheet
uM-FPU V3.1 Instruction Set
Using the uM-FPU V3 Integrated Development Environment (IDE)
Using uM-FPU V3.1 with the WinAVR™ Compiler

Appendix A - Execution Times for Individual Functions