Freescale Semiconductor
Application Note

Document Number: AN2354
Rev. 1, 08/2008

Running the Dhrystone Benchmark for the MPC500 Family

by: Glenn Jackson, TECD Applications

This application note provides an adaptation of the standard Dhrystone version 2.1 to the Motorola MPC500 family of PowerPC architecture-compliant MCUs. Dhrystone version 2.1 is a publicly-available standard for benchmarking various CPUs. All benchmarks are made relative to the original VAX Dhrystone. The original Dhrystone 2.1 function remains intact for this application, with changes made only to the wrapper code that permits timing for the MPC500 PowerPC architecture-compliant family.

Contents

1 Overview

The Dhrystone benchmark is intended to provide a compact benchmark. The user will be able to run the Dhrystone program to detect relative performance differences between internal and external memory settings and other system configurations. Due to the universal nature of the Dhrystone benchmark, results will not confirm optimal performance for the MPC500 PowerPC-architecture family. The user's final

freescale semiconductor

application code results will depend on the system configuration, code function, and whether the code is structured to take full advantage of the MPC500-family features.

2 Detailed Description

Five program files are required for this application to compile. Three of the files are the original Dhrystone 2.1 files: Dhry.h, Dhry_1.c, and Dhry_2.c. The two remaining files, clock.c and Crt0.s, are used to support the specific timing needs of the MPC500 PowerPC-compatible family.

Two additional files, evb.lin (the link file) and the makefile, are included in this application to perform a C compile operation. The example compiler is the Wind River Diab Data C compiler. The Crt0.s file, mentioned above, replaces the default Wind River Diab Data Crt0.s file used in the C compiler compilation.

Changes made to the Dhrystone program must not have any effect on the internal Dhrystone operation. The changes made in this application only effect the wrapper code used to record the begin and end times of the Dhrystone run. Because the original code accounted for many other processors and software packages, portions of the original wrapper code have been commented out to assure no interference with the MPC500 timing adaptations; however, the original code remains in the software so that a user can confirm that the changes do not alter the validity of the original Dhrystone software. Also, numerous *printf* statements have been left in the code but are also commented out.

Following is a review of each program segment that includes a general description and a description of changes needed for proper MPC500 operation.

- The Dhry.h program is composed of a header, global timer, global variable, and structure definitions. The header describes the structure and workings of the Dhrystone program in detail. Because the timer definitions were written for non-MPC500 family processors, they have been commented out; however, they have not been removed so that the user could confirm the changes to the original Dhrystone programs. The variables and structure definitions affect the functioning of the Dhrystone program and have not been changed.
- The Dhry1.c program consists of a wrapper and main program. The wrapper portion contains timing definitions and variables that have been added for MPC500 operation. Timer variables for processors other than the MPC500 have been commented out. There are also *printf* statements in the Dhry1.c program which have been commented out. The *printf* statements may be reactivated as they are not required to generate the resulting Dhrystone data.
- The clock_val variable is assigned a value based on a 4-MHz crystal. If a 20-MHz crystal is used, then the 4-MHz assignment should be commented out and the assignment for the 20-MHz crystal should be commented in the next line of code.
- The main function of the Dhrystone program is located in Dhry1.c. No changes have been made that affect the operation of this part of the code. This assures consistency across processors and configurations in the resulting data from the Dhrystone benchmark runs.
- The Dhry2.c program contains other Dhrystone process functions. These functions are part of the inner working of the Dhrystone program.
- The clock.c program is a support program for the MPC500 configuration. It includes functions that initialize the timer, read the timer value, and calculate the total time for the benchmark execution.

- The Crt0.s program is a support program for the MPC500 configuration. The Crt0.s includes required startup code that establishes compiler structural definitions and initializations for the Wind River Diab Data C compiler. Most of the initializations in the Crt0.s are necessary for all compiles of the MPC500, regardless of the compiler tool involved.

An example listing of the code is available in Section 6, "Code Listings."

3 Compiling and Running the Dhrystone Program for the MPC500 Family

3.1 Compiling Code for the Dhrystone Program

Two files are used to generate the initial compile of the Dhrystone program. The makefile activates the commands that bring the program segments together with specific compiler configuration options. The evb.lin program is the linker program that determines where the Dhrystone program will reside in memory. The evb.lin also establishes the locations for internal modules required for the C compile.

The file structure, shown in Figure 1, is kept simple to serve as a quick and easy example. The command line instructions are invoked in this directory location.

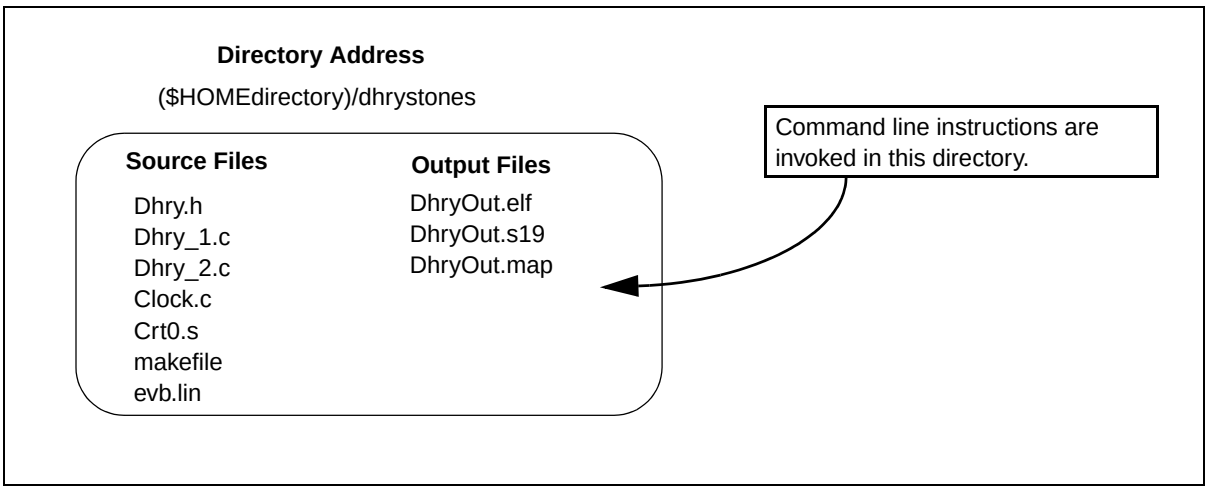

Figure 1. File Directory Structure

The makefile illustrates the steps necessary to compile the Dhrystone example. The comments in the makefile serve as instructions.

The Dhrystone code is compiled by placing all of the source files in one directory and using the *make -f makefile* instruction on a command line. The output file is named in the "EXECUTABLE = DhryOut" line and will be called DhryOut.elf.

The makefile is used with the make command. In the makefile, a list of modules can be used as variables for building an object and source file list. Suffixes are included with the variables of the makefile.

Diab Data includes a dmake command line tool which requires a slightly altered version of the makefile. This version, shown in Section 3.3, "Listing 2: The Dmakefile," is called the dmakefile. The calling

command for this version is *dmake -f dmakefile*. The dmake file requires the object file list to be specifically spelled out. In dmake, the file suffixes are added on a separate command line in the dmakefile.

Either the make or dmake command will work with the appropriate syntax.

3.2 Listing 1: The Makefile

```
/*************************************************************************/
# Makefile for Motorola MPC500 Power PC compatible
# compiled code for Dhrystone ver. 2.1
# Used with the make command and
# DiabData compiler version 4.3g

# Put the supporting files here.
MODULES = Dhry_1 Dhry_2 clock

# Object and source file lists are generated here
# from the modules listed above.
OBJECTS = $(MODULES:%=%.o)
SOURCES = $(MODULES:%=%.c)

# Variable commands set up for the compile.
CC      = dcc
AS      = das
LD      = dcc
DUMP    = ddump

# Non-compressed regular run command options:
# NOTE: Using -g3 is only slightly faster than no -g3
#       in COPTS (+0.5 Dhrystones).
# NOTE: Use the much slower -g option for a full debug.
# NOTE: Using -XO and -g3 for speed options (fastest combination)
#
COPTS   = -tPPC555EH:cross -@E+err.log -g3 -S -XO
AOPTS   = -tPPC555EH:cross -@E+err.log -g
LOPTS   = -tPPC555EH:cross -@E+err.log -Wscrt0.o -m2 -lm

# NOTE: When using code compression; remove the comments here for
#        COPTS, AOPTS, and LOPTS. Also, comment out COPTS, AOPTS, and
#        LOPTS above.
# Code compression command options:
#COPTS   = -tPPC555CH:cross -Xprepare-compress -@E+err.log -g -S -c -XO
#AOPTS   = -tPPC555CH:cross -Xprepare-compress -@E+err.log -g
#LOPTS   = -tPPC555CH:cross -Xassociate-headers -@E+err.log -Wscrt0.o -m2 -lm

# Set up the name of the output executable file(s):
EXECUTABLE = DhryOut

default: crt0.o $(EXECUTABLE).elf $(EXECUTABLE).s19

# Compile crt0.o from src/crt0.s
# Use the local crt0.o with "-Wscrt0.o" in LOPTS
crt0.o: crt0.s
        $(AS) $(AOPTS) crt0.s

$(EXECUTABLE).elf: $(OBJECTS)
```

```
        $(LD) $(LOPTS) $(OBJECTS) -o $(EXECUTABLE).elf -Wm evb.lin > $(EXECUTABLE).map
        $(DUMP) -tv $(EXECUTABLE).elf >>$(EXECUTABLE).map

$(OBJECTS): $(SOURCES)
        $(CC) $(COPTS) $*.c
        $(AS) $(AOPTS) $*.s

# Generate s record file for flashing

$(EXECUTABLE).s19: $(EXECUTABLE).elf
        $(DUMP) -Rv -o $(EXECUTABLE).s19 $(EXECUTABLE).elf
```

3.3 Listing 2: The Dmakefile

```
/*************************************************************************/
# dmakefile for Motorola MPC500 Power PC compatible
# compiled code for Dhrystone ver. 2.1
# Used with the dmake command and
# DiabData compiler version 4.3g

# Object and source file lists are generated here
# from the modules listed above.
OBJECTS = Dhry_1.o Dhry_2.o clock.o

# Variable commands set up for the compile.
CC      = dcc
AS      = das
LD      = dcc
DUMP     = ddump

# Non-compressed regular run command options:
# NOTE: Using -g3 is only slightly faster than no -g3
#       in COPTS (+0.5 Dhrystones).
# NOTE: Use the much slower -g option for a full debug.
# NOTE: Use -XO and -g3 for speed options (fastest combination)
#
COPTS   = -tPPC555EH:cross -@E+err.log -g3 -c -XO
AOPTS   = -tPPC555EH:cross -@E+err.log -g
LOPTS   = -tPPC555EH:cross -@E+err.log -Wscrt0.o -m2 -lm

# Code compression command options:
#COPTS   = -tPPC555CH:cross -Xprepare-compress -@E+err.log -g3 -c -XO
#AOPTS   = -tPPC555CH:cross -Xprepare-compress -@E+err.log -g
#LOPTS   = -tPPC555CH:cross -Xassociate-headers -@E+err.log -Wscrt0.o -m2 -lm

# Set up the name of the output executable file(s):
EXECUTABLE = DhryOutd

.SUFFIXES: .c .s

default: crt0.o $(EXECUTABLE).elf $(EXECUTABLE).s19

# Compile crt0.o from src/crt0.s
# Use the local crt0.o with "-Wscrt0.o" in LOPTS
crt0.o: crt0.s
        $(AS) $(AOPTS) crt0.s
```

```
dcc $(COPTS) -o clock.o clock.c
das $(AOPTS) clock.c

.c.o :
        $(CC) $(COPTS) -o $*.o $<
.s.o :
        $(AS) $(AOPTS) $<

$(EXECUTABLE).elf: $(OBJECTS)
        $(LD) $(LOPTS) $(OBJECTS) -o $(EXECUTABLE).elf -Wm evbdec.lin > $(EXECUTABLE).map
        $(DUMP) -tv $(EXECUTABLE).elf >>$(EXECUTABLE).map


# Generate s record file for flashing

$(EXECUTABLE).s19: $(EXECUTABLE).elf
        $(DUMP) -Rv -o $(EXECUTABLE).s19 $(EXECUTABLE).elf
```

3.4 Listing 3: The Linkfile evb.lin

The program in listing 3 shows the link file evb.lin. The memory area sets the starting location and length of the code. The starting point of RAM values and length are also set in this area. The main code is organized at 0x2000 because the address space between 0x0 and 0x1FFF is reserved for an exception table. Exceptions are not used in this example so that a minimal Dhrystone version can be illustrated.

The “Sections” part of the code provides structure definitions used in the Diab Data C compile. The variables at the end of the link file assign actual address values to each of the groups and subsections.

```
/*****************************************************************************/
/* evb.lin link file for MPC500 */
/* Memory locations 0x0 - 0x2000 are reserved for exception table. */

MEMORY
{
        internal_flash: org = 0x2000,   len = 0x10000
        internal_ram:   org = 0x3f9800, len = 0x57F0
        stack:          org = 0x3ff000, len = 0xFF0
}

SECTIONS
{
/* The following instructions will only work with an exceptions.s file*/
/*      reset_exception   0x100 : {}            */
/*      ext_irq_exception 0x500 : {}            */

        GROUP : {
            .text (TEXT)    : {
                 *(.text)
                 *(.rodata)
                 *(.rdata)
                 *(.init)
                 *(.fini)
                 *(.eini)
            }
            .sdata2 (TEXT) : {}
```

```
        } > internal_flash

/*  End of added sections  */

      GROUP : {
              .data  (DATA) LOAD(ADDR(.sdata2)+SIZEOF(.sdata2)) : {}
             .sdata (DATA) LOAD(ADDR(.sdata2)+SIZEOF(.sdata2)+SIZEOF(.data)) : {}
             .sbss  (BSS) : {}
             .bss   (BSS) : {}
        } > internal_ram
}

__SP_INIT       = ADDR(stack)+SIZEOF(stack);
__SP_END        = ADDR(stack);
__DATA_ROM      = ADDR(.sdata2)+SIZEOF(.sdata2);
__DATA_RAM      = ADDR(.data);
__DATA_END      = ADDR(.sdata)+SIZEOF(.sdata);
__BSS_START     = ADDR(.sbss);
__BSS_END       = ADDR(.bss)+SIZEOF(.bss);
__HEAP_START    = ADDR(.bss)+SIZEOF(.bss);
__HEAP_END      = ADDR(internal_ram)+SIZEOF(internal_ram);
```

3.5 Running Code for the Dhrystone Program

In order to run the compiled .elf file, the file should be loaded into Flash. The instruction pointer (IP) in the debug tool should be set to 0x2004, which is the start of the program. Start the program running with any *go* command. The program will run to completion. The instructions for viewing the results are found in Section 5.2, “Running and Viewing Results.”

4 Code-Compressed Version of the Dhrystone Program

4.1 Compiling Code for Compression

The following description is intended to show the minimum additional steps required for code compression operation. This section is not intended as a discussion of all aspects of the code compression process.

Code-compressed operation is achieved with a few additional steps. First, the code is compiled with the hooks that are needed later for compression. Next, a vocabulary is generated for code decompression. Third, the code is compressed and stored as a loadable *.elf file. Finally, the compressed code can be run after loading the DECRAM.

The file structure is kept simple to serve as a quick and easy example. The file structure is illustrated in Figure 2. The command line instructions, including the compression instructions, are invoked from this directory.

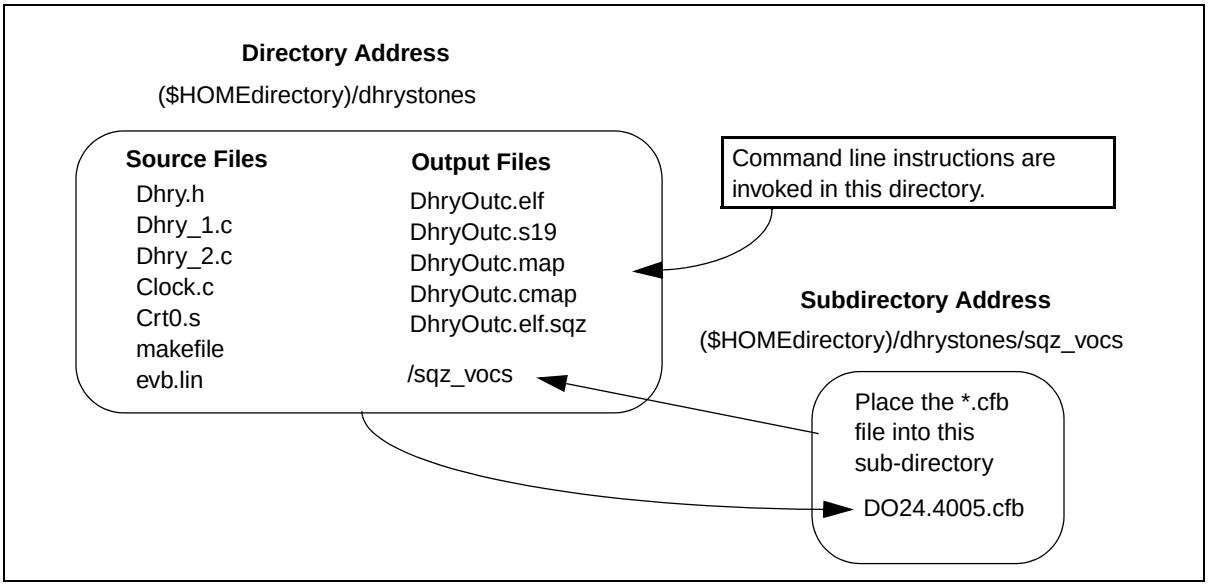

Figure 2. Code Compression File Directory Structure

To compile with compression hooks, the makefiles illustrated above need only their COPTS, AOPTS, and LOPTS lines altered. The alternative lines are present in the makefiles and may be commented in after the original lines are commented out. For COPTS and AOPTS, the *-Xprepare-compress* option is added. For LOPTS (linker options) the option of *-Xassociate-headers* is added. The resulting output .elf file will run without compression but will be ready for compression. A unique output name could be generated by changing the output to "EXECUTABLE = DhryOutC," where C denotes a compiled non-compressed output .elf file (DhryOutC.elf) that can be compressed.

This application note uses the MPC566 to provide an overview of the compression process. A vocabulary file is used by the hardware to decompress the instructions. The vocabulary is generated by the VocGEN tool in a command line (DOS) window using the following command:

```
vocgen -bs 4 -tp MPC565_00 DhryOutc.elf     --> This will output a *.cfb file.
```

The newly generated vocabulary will be called DO24.4005.cfb (or something similar) and must be placed into a subdirectory (called sqz_vocs) where the Squeezard tool will be used (see Figure 2). The vocabulary file and the DhryOutc.elf source file will both be used to generate the compressed file. The command line window used to invoke the Squeezard compression tool should be located at the source file directory location with the sqz_vocs as a subdirectory to this location. Squeezard is invoked with the following command:

```
squeezard -tp MPC565_00 -sd 20000 -o_m -evf DO24.4005.cfb DhryOutc.elf
```

This will generate the output compressed .elf file which is renamed DhryOutc.sqz.elf. Load this file into Flash in the same manner as any other .elf-format file.

4.2 Running Compressed Dhrystone Code

The MPC566 hardware must be initialized in compression mode. This can be performed with the reset configuration word bits 21 (comp_en bit) and 22 (exc_comp bit) set at power on reset. The Flash

programmer tools will work in this configuration. With the DhryOutc.sqz.elf file loaded into Flash, the IP must be set to a program location that will load the DECRAM with the vocabulary. After the vocabulary is loaded, the program will proceed to the beginning of the program and continue to run until completion. If the DECRAM has been loaded, further compression mode runs will work with the IP set to the beginning of the program.

In code compression mode, the IP address value must be shifted left by two bits. If the initial IP address is 0x10000, then the value of 0x40000 must be entered into the IP value. (This is the major difference between compressed and non-compressed operation.) When observing code in the debug window, the non-compressed address will be visible and the IP will point to this non-compressed address.

The initial IP address is found in the DhryOutc.elf.cmap file. The .sqz_dib_text address is the first address location at which the IP setting will download the DECRAM and proceed on to the regular program. For additional runs of the Dhrystone program, the IP can be set to the .text address in the *.cmap file (left shift this address by two bits). This will set the IP to the start of the program.

The Dhry_tics and Seconds results may be viewed in the same variable window as in a non-compressed run. See Section 5.2, “Running and Viewing Results.”

5 Conclusions and Results

5.1 Calculating the Results

The results of the Dhrystone program are based on the speed of the original VAX computer. The VAX equivalent for Dhrystone version 2.1 is 1757 Dhrystone cycles/VAX MIPS. The number of seconds that the Dhrystone program takes to run is calculated from the number of clock ticks. The number of clock ticks are stored in the Dhry_Ticks variable found in the program. The Dhry_Ticks number of clock ticks are divided by one million to derive the number of seconds. The equation involved is as follows:

$$\frac{1,000,000[\text{Dhrystones cycles}]}{\text{number of seconds} \times 1757[\text{Dhrystones cycles/VAX mips}]}$$

This yields:

$$X\ [\text{VAX mips/second}]$$

The resulting number X is the number of times faster the MPC500 will be than the original VAX speed. This will yield a standard output value in Dhrystone units that can be compared to other processors. However, the user must keep in mind that such a benchmark number may not reveal all of the processing power of the MPC500 family. When the Dhrystone benchmark is used across various configurations of the MPC500 family (i.e., internal or external, various speeds, different internal settings, varying wait states, etc.), the user can compare how changing the system configurations will affect a final application system solution.

The Dhry_Ticks count is established by:

1. Starting the timebase (in crt0.s)

2. Initializing the PPC decrementer timer to the value of 0xFFFFFFFF (called in Dhry1.c; function is in Clock.c);
3. Setting the Begin_Time variable to the PPC decrementer value just before entering the Dhrystone program (dhryl.c)
4. Setting the End_Time variable to the PPC decrementer value just after leaving the Dhrystone program (dhryl.c)
5. Determining the value of Dhry_Ticks; calculating the difference between Begin_Time and End_Time (dhryl.c)

From the Decrementer Time-Out Periods table (Table 6-6 in the MPC565 User's Manual), one can observe that with a 4-MHz crystal, 999,999 PPC decrementer ticks equals approximately 1.0 second. Therefore, dividing the value of Dhry_Ticks by one million (1,000,000), will yield the results of the Dhrystone run in seconds.

5.2 Running and Viewing Results

Running the program involves loading DhryOut.elf into the internal Flash of the MPC500 device with any available Flash loader. Before starting the program, the program counter (instruction pointer) should be set to address 0x2004.

To read the results of a run, open a memory window to view the two output variables, Dhry_Ticks and Seconds. Dhry_Ticks is located at 0x3F9B38 and Seconds is located at 0x3F9B3C in the internal RAM.

In the Lauterbach debug tool, the variables can be observed with the *Var.view Dhry_Ticks* and *Var.view Seconds* commands. In the SDS debug tool, the variables can be observed with the *watch Dhry_Ticks* and *watch Seconds* commands. A watch window will pop up and reveal the variable values.

This value is the number of seconds entered into the equation above. Note that a change in compile or make options may change the hex address location for Dhry_Ticks and Seconds.

Results from Dhrystone running times are shown in Table 1 below. The internal Flash times were run on an MPC555 at 40MHz and an MPC563 at 56MHz. The external Flash times were run on an MPC561 with a sample burst mode timing of 4-1-1-1 and a sample extended burst mode timing of 4-1-1-1-1-1-1-1. The external times were run on the MPC561 at 32MHz and the results were interpolated up to the 56MHz value. This is legitimate since the results of these benchmarks are linear with respect to clock speed. The 32MHz clock speed is the fastest time that the evaluation board would run and still achieve the 1 clock secondary burst timings. This timing simulates results that will be possible for future external Flashes.

Table 1. MPC500 Dhrystone Results

Device	Speed	System Config.	BTB[1]	Dhry_ticks	Seconds	Dhrystone VAX MIPS	.text Size (bytes)
MPC555	40 MHz	Internal Flash	Off	10,375,000	10.375	62.59	0x2BD0
MPC56x[2]	56 MHz	Internal Flash	On	7,321,428	7.32	88.71	0x2BD0
MPC56x[2]	56 MHz	Internal Flash	Off	7,464,286	7.46	87.04	0x2BD0
MPC56x[3]	56 MHz	Internal Flash Compressed	On	7,642,858	7.64	84.99	0x1264

Table 1. MPC500 Dhrystone Results (continued)

Device	Speed	System Config.	BTB[1]	Dhry_ticks	Seconds	Dhrystone VAX MIPS	.text Size (bytes)
MPC561	56 MHz	External Burst[4]	On	16,857,143	16.86	38.51	0x2BD0
MPC561	56 MHz	External Burst[5]	On	15,392,857	15.39	42.18	0x2BD0
MPC561	56 MHz	Ext. Extended Burst[6]	On	13,107,144	13.11	49.54	0x2BD0
MPC561	56 MHz	Compressed Ext. Extended Burst[7]	On	11,839,287	11.84	54.84	0x1264

1 The BTB is the Branch Target Buffer

2 The “x” may represent the MPC563 or the MPC565 without compression mode.

3 The “x” may represent the MPC564, or MPC566 in compression mode.

4 External BBC burst with 4-1-1-1 timing.

5 External USIU burst with 4-1-1-1 timing

6 External extended USIU burst with 4-1-1-1-1-1-1-1 timing.

7 Compressed external extended USIU burst with 4-1-1-1-1-1-1-1 timing.

Conclusion

The system configuration generates a variety of results from the Dhrystone benchmark. Running Dhrystone from internal Flash with the branch target buffer activated yields the fastest completion time. Other result times scale linearly according to the processor speed. (For example, it is possible to achieve a 41% improvement by moving from 40MHz to 56MHz.) Running the program with code compression creates a 4% time penalty with internal Flash, but it reduces code size by more than 50%. Using a BBC burst with a 4-1-1-1 beat data access runs at 43% of the fastest internal speed, and using the USIU with a burst of 4-1-1-1 beats for data accesses runs at 48% of the fastest internal speed. Using the USIU extended eight beat burst with cycle timings of 4-1-1-1-1-1-1-1 for data accesses improves the external access time to 56% of the fastest internal speed. Code compression improves external access times because more than one instruction is retrieved with each external access. Using external code compression with the USIU extended eight beat burst will yield a running time that is 62% of the fastest internal speed.

These are example results for this version of the Dhrystone benchmark. User applications should yield similar results, though the form of the application code and the final application system structure may cause some variation. The Dhrystone benchmark can be used as a concise code example that users can incorporate into their system to test for functionality and compare to the results of this application note.

6 Code Listings

6.1 Listing 4: Dhry.h

```
/*
****************************************************************************
 *
 *                   "DHRYSTONE" Benchmark Program
 *                   -----------------------------
 *  Version:    C, Version 2.1
 *  File:       dhry.h (part 1 of 3)
```

```
* *  Date:       May 25, 1988
* *  Author:     Reinhold P. Weicker
*                      Siemens AG, E STE 35
*                      Postfach 3240
*                      8520 Erlangen
*                      Germany (West)
*                              Phone:  [xxx-49]-9131-7-20330
*                                      (8-17 Central European Time)
*                              Usenet: ..!mcvax!unido!estevax!weicker
*
*              Original Version (in Ada) published in
*              "Communications of the ACM" vol. 27., no. 10 (Oct. 1984),
*              pp. 1013 - 1030, together with the statistics
*              on which the distribution of statements etc. is based.
*
*              In this C version, the following C library functions are used:
*              - strcpy, strcmp (inside the measurement loop)
*              - printf, scanf (outside the measurement loop)
*              In addition, Berkeley UNIX system calls "times ()" or "time ()"
*              are used for execution time measurement. For measurements
*              on other systems, these calls have to be changed.
*
* Changes: 11/29/01 G. J. commented out variables for non-Power PC compatible
*                   MPC500 timings.
*
*
*  Collection of Results:
*              Reinhold Weicker (address see above) and
*
*              Rick Richardson
*              PC Research. Inc.
*              94 Apple Orchard Drive
*              Tinton Falls, NJ 07724
*                      Phone:  (201) 389-8963 (9-17 EST)
*                      Usenet: ...!uunet!pcrat!rick
*
*      Please send results to Rick Richardson and/or Reinhold Weicker.
*      Complete information should be given on hardware and software used.
*      Hardware information includes: Machine type, CPU, type and size
*      of caches; for microprocessors: clock frequency, memory speed
*      (number of wait states).
*      Software information includes: Compiler (and runtime library)
*      manufacturer and version, compilation switches, OS version.
*      The Operating System version may give an indication about the
*      compiler; Dhrystone itself performs no OS calls in the measurement loop.
*
*      The complete output generated by the program should be mailed
*      such that at least some checks for correctness can be made.
*
******************************************************************************
*
*  History:    This version C/2.1 has been made for two reasons:
*
*              1) There is a need for a common C version of Dhrystone,
       *                 since C is at present the most popular system
*              programming language for the class of processors
*              (microcomputers, minicomputers) where Dhrystone is used most.
```

```
*              There should be, as far as possible, only one C version of
*              Dhrystone such that results can be compared without
*              restrictions. In the past, the C versions distributed
*              by Rick Richardson (Version 1.1) and by Reinhold Weicker
*              had small (though not significant) differences.
*
*              2) As far as it is possible without changes to the Dhrystone
*              statistics, optimizing compilers should be prevented from
*              removing significant statements.
*
*              This C version has been developed in cooperation with
*              Rick Richardson (Tinton Falls, NJ), it incorporates many
*              ideas from the "Version 1.1" distributed previously by
*              him over the UNIX network Usenet.
*              I also thank Chaim Benedelac (National Semiconductor),
*              David Ditzel (SUN), Earl Killian and John Mashey (MIPS),
*              Alan Smith and Rafael Saavedra-Barrera (UC at Berkeley)
*              for their help with comments on earlier versions of the
*              benchmark.
*
*  Changes:    In the initialization part, this version follows mostly
*              Rick Richardson's version distributed via Usenet, not the
*                version distributed earlier via floppy disk by Reinhold Weicker.
*              As a concession to older compilers, names have been made
*              unique within the first 8 characters.
*              Inside the measurement loop, this version follows the
*              version previously distributed by Reinhold Weicker.
*
*              At several places in the benchmark, code has been added,
*              but within the measurement loop only in branches that
*              are not executed. The intention is that optimizing compilers
*              should be prevented from moving code out of the measurement
*              loop, or from removing code altogether. Since the statements
*              that are executed within the measurement loop have NOT been
*              changed, the numbers defining the "Dhrystone distribution"
*              (distribution of statements, operand types and locality)
*              still hold. Except for sophisticated optimizing compilers,
*              execution times for this version should be the same as
*              for previous versions.
*
*              Since it has proven difficult to subtract the time for the
*              measurement loop overhead in a correct way, the loop check
*              has been made a part of the benchmark. This does have
*              an impact - though a very minor one - on the distribution
*              statistics which have been updated for this version.
*
*              All changes within the measurement loop are described
*              and discussed in the companion paper "Rationale for
*              Dhrystone version 2".
*
*              Because of the self-imposed limitation that the order and
*              distribution of the executed statements should not be
*              changed, there are still cases where optimizing compilers
*              may not generate code for some statements. To a certain
*              degree, this is unavoidable for small synthetic benchmarks.
*              Users of the benchmark are advised to check code listings
*              whether code is generated for all statements of Dhrystone.
```

```
*
*               Version 2.1 is identical to version 2.0 distributed via
*               the UNIX network Usenet in March 1988 except that it corrects
*               some minor deficiencies that were found by users of version 2.0.
*               The only change within the measurement loop is that a
*               non-executed "else" part was added to the "if" statement in
*               Func_3, and a non-executed "else" part removed from Proc_3.
*
*****************************************************************************
*
* Defines:      The following "Defines" are possible:
*               -DREG=register          (default: Not defined)
*                       As an approximation to what an average C programmer
*                       might do, the "register" storage class is applied
*                       (if enabled by -DREG=register)
*                       - for local variables, if they are used (dynamically)
*                         five or more times
*                       - for parameters if they are used (dynamically)
*                         six or more times
*                       Note that an optimal "register" strategy is
*                       compiler-dependent, and that "register" declarations
*                       do not necessarily lead to faster execution.
*               -DNOSTRUCTASSIGN        (default: Not defined)
*                       Define if the C compiler does not support
*                       assignment of structures.
*               -DNOENUMS               (default: Not defined)
*                       Define if the C compiler does not support
*                       enumeration types.
*               -DTIMES                 (default)
*               -DTIME
*                       The "times" function of UNIX (returning process times)
*                       or the "time" function (returning wallclock time)
*                       is used for measurement.
*                       For single user machines, "time ()" is adequate. For
*                       multi-user machines where you cannot get single-user
*                       access, use the "times ()" function. If you have
*                       neither, use a stopwatch in the dead of night.
*                       "printf"s are provided marking the points "Start Timer"
*                       and "Stop Timer". DO NOT use the UNIX "time(1)"
*                       command, as this will measure the total time to
*                       run this program, which will (erroneously) include
*                       the time to allocate storage (malloc) and to perform
*                       the initialization.
*               -DHZ=nnn
*                       In Berkeley UNIX, the function "times" returns process
*                       time in 1/HZ seconds, with HZ = 60 for most systems.
*                       CHECK YOUR SYSTEM DESCRIPTION BEFORE YOU JUST APPLY
*                       A VALUE.
*
*****************************************************************************
*
*  Compilation model and measurement (IMPORTANT):
*
*  This C version of Dhrystone consists of three files:
*  - dhry.h (this file, containing global definitions and comments)
*  - dhry_1.c (containing the code corresponding to Ada package Pack_1)
*  - dhry_2.c (containing the code corresponding to Ada package Pack_2)
```

```
*
*  The following "ground rules" apply for measurements:
*  - Separate compilation
*  - No procedure merging
*  - Otherwise, compiler optimizations are allowed but should be indicated
*  - Default results are those without register declarations
*  See the companion paper "Rationale for Dhrystone Version 2" for a more
*  detailed discussion of these ground rules.
*
*  For 16-Bit processors (e.g. 80186, 80286), times for all compilation
*  models ("small", "medium", "large" etc.) should be given if possible,
*  together with a definition of these models for the compiler system used.
*
**************************************************************************
*
*  Dhrystone (C version) statistics:
*
*  [Comment from the first distribution, updated for version 2.
*   Note that because of language differences, the numbers are slightly
*   different from the Ada version.]
*
*  The following program contains statements of a high level programming
*  language (here: C) in a distribution considered representative:
*
*    assignments                   52 (51.0 %)
*    control statements            33 (32.4 %)
*    procedure, function calls     17 (16.7 %)
*
*  103 statements are dynamically executed. The program is balanced with
*  respect to the three aspects:
*
*    - statement type
*    - operand type
*    - operand locality
*          operand global, local, parameter, or constant.
*
*  The combination of these three aspects is balanced only approximately.
*
*  1. Statement Type:
*  -----------------             number
*
*     V1 = V2                      9
*       (incl. V1 = F(..)
*     V = Constant                12
*     Assignment,                  7
*       with array element
*     Assignment,                  6
*       with record component
*                                 --
*                                 34       34
*
*     X = Y +|-|"&&"|"|" Z         5
*     X = Y +|-|"==" Constant      6
*     X = X +|- 1                  3
*     X = Y *|/ Z                  2
*     X = Expression,              1
*           two operators
```

```
*     X = Expression,             1
*           three operators
*                                --
*                         18      18
*
*     if ....                    14
*       with "else"       7
*       without "else"    7
*           executed        3
*           not executed    4
*     for ...                     7  |  counted every time
*     while ...                   4  |  the loop condition
*     do ... while                1  |  is evaluated
*     switch ...                  1
*     break                       1
*     declaration with            1
*       initialization
*                                --
*                                34      34
*
*     P (...)  procedure call    11
*       user procedure    10
*       library procedure 1
*     X = F (...)
*             function  call      6
*       user function     5
*       library function      1
*                                --
*                                17      17
*                                       ---
*                                       103
*
*    The average number of parameters in procedure or function calls
*    is 1.82 (not counting the function values aX *
*
*  2. Operators
*  ------------
*                           number    approximate
*                                     percentage
*
*    Arithmetic              32     50.8
*
*       +                      21   33.3
*       -                       7   11.1
*       *                       3    4.8
*       / (int div)             1    1.6
*
*    Comparison              27     42.8
*
*       ==                      9   14.3
*       /=                      4    6.3
*       >                       1    1.6
*       <                       3    4.8
*       >=                      1    1.6
*       <=                      9   14.3
*
*    Logic                    4      6.3
```

```
*
*       && (AND-THEN)           1    1.6
*       |  (OR)                 1    1.6
*       !  (NOT)                2    3.2
*
*                             --          -----
*                             63          100.1
*
*
*  3. Operand Type (counted once per operand reference):
*  ---------------
*                           number    approximate
*                                     percentage
*
*     Integer                175        72.3 %
*     Character               45        18.6 %
*     Pointer                 12         5.0 %
*     String30                 6         2.5 %
*     Array                    2         0.8 %
*     Record                   2         0.8 %
*                            ---       -------
*                            242       100.0 %
*
*  When there is an access path leading to the final operand (e.g. a record
*  component), only the final data type on the access path is counted.
*
*
*  4. Operand Locality:
*  --------------------
*                                  number    approximate
*                                            percentage
*
*     local variable               114        47.1 %
*     global variable               22         9.1 %
*     parameter                     45        18.6 %
*        value            23         9.5 %
*        reference        22         9.1 %
*     function result               6         2.5 %
*     constant                     55        22.7 %
*                                  ---       -------
*                                  242       100.0 %
*
*
*  The program does not compute anything meaningful, but it is syntactically
*  and semantically correct. All variables have a value assigned to them
*  before they are used as a source operand.
*
*  There has been no explicit effort to account for the effects of a
*  cache, or to balance the use of long or short displacements for code or
*  data.
*
*****************************************************************************
 */

/* Compiler and system dependent definitions: */

/********* begin GJ changes **************************/
```

```
/******* Commenting out non PPC timing variables ***/

// Remove non-used timing variables:
//#ifndef TIME
//#undef TIMES
//#define TIMES
//#endif
                /* Use times(2) time function unless    */
                /* explicitly defined otherwise         */

//#ifdef MSC_CLOCK
//#undef HZ
//#undef TIMES
//#include <time.h>
//#define HZ     CLK_TCK
//#endif
               /* Use Microsoft C hi-res clock */


//#ifdef TIMES
//#include <sys/types.h>
//#include <sys/times.h>
                /* for "times" */
//#endif

//#define Mic_secs_Per_Second     1000000.0
       /* Berkeley UNIX C returns process times in seconds/HZ */

/*******end GJ changes ***************************/


#ifdef  NOSTRUCTASSIGN
#define structassign(d, s)      memcpy(&(d), &(s), sizeof(d))
#else
#define structassign(d, s)      d = s
#endif

#ifdef  NOENUM
#define Ident_1 0
#define Ident_2 1
#define Ident_3 2
#define Ident_4 3
#define Ident_5 4
  typedef int   Enumeration;
#else
  typedef       enum    {Ident_1, Ident_2, Ident_3, Ident_4, Ident_5}
                Enumeration;
#endif
        /* for boolean and enumeration types in Ada, Pascal */

/* General definitions: */

#include <stdio.h>
                /* for strcpy, strcmp */

#define Null 0
                /* Value of a Null pointer */
#define true  1
```

```
#define false 0

typedef int     One_Thirty;
typedef int     One_Fifty;
typedef char    Capital_Letter;
typedef int     Boolean;
typedef char    Str_30 [31];
typedef int     Arr_1_Dim [50];
typedef int     Arr_2_Dim [50] [50];

typedef struct record
    {
    struct record *Ptr_Comp;
    Enumeration    Discr;
    union {
          struct {
                  Enumeration Enum_Comp;
                  int         Int_Comp;
                  char        Str_Comp [31];
                  } var_1;
          struct {
                  Enumeration E_Comp_2;
                  char        Str_2_Comp [31];
                  } var_2;
          struct {
                  char        Ch_1_Comp;
                  char        Ch_2_Comp;
                  } var_3;
          } variant;
      } Rec_Type, *Rec_Pointer;
```

6.2 Listing 5: Dhry1.c

```
/*************************************************************************
 *
 *                   "DHRYSTONE" Benchmark Program
 *                   -----------------------------
 *
 *  Version:    C, Version 2.1
 *
 *  File:       dhry_1.c (part 2 of 3)
 *
 *  Date:       May 25, 1988
 *
 *  Author:     Reinhold P. Weicker
 *
 * Changes: 11/29/01 G. J. Added variables for Power PC compatible MPC500
 *                   timings. Also commented out printf statements.
 *
 *************************************************************************
 */

#define HZ 60
#include "Dhry.h"

/* Global Variables: */
```

```
Rec_Pointer     Ptr_Glob,
                Next_Ptr_Glob;
int             Int_Glob;
Boolean         Bool_Glob;
char            Ch_1_Glob,
                Ch_2_Glob;
int             Arr_1_Glob [50];
int             Arr_2_Glob [50] [50];

/*********** begin GJ addition******************/
unsigned long BeginTime;
unsigned long EndTime;
unsigned long Dhry_Ticks;
float clock_val;
float Seconds;
/************* end GJ addition *****************/

extern char     *malloc ();
Enumeration     Func_1 ();
  /* forward declaration necessary since Enumeration may not simply be int */

#ifndef REG
        Boolean Reg = false;
#define REG
        /* REG becomes defined as empty */
        /* i.e. no register variables   */
#else
        Boolean Reg = true;
#endif

/* variables for time measurement: */
#ifndef ETAS
#ifdef TIMES
struct tms      time_info;
extern  int     times();

                /* see library function "times" */
#define Too_Small_Time (2*HZ)
                /* Measurements should last at least about 2 seconds */
#endif
#endif

#ifdef TIME
extern long     time();
                /* see library function "time"  */
#define Too_Small_Time 2
                /* Measurements should last at least 2 seconds */
#endif
#ifdef MSC_CLOCK
extern clock_t clock();
#undef HZ
#define HZ 1000000
#define Too_Small_Time (2*HZ)
#endif

/********* begin GJ change ************/
```

```
//    Comment out original timing variables ...
//long            Begin_Time,
//                End_Time,
//                User_Time;
//float           Microseconds,
//                Dhrystones_Per_Second;

/* end of variables for time measurement */
/************ end GJ change *************/


main ()
/*****/

  /* main program, corresponds to procedures        */
  /* Main and Proc_0 in the Ada version             */
{
        One_Fifty       Int_1_Loc;
  REG   One_Fifty       Int_2_Loc;
        One_Fifty       Int_3_Loc;
  REG   char            Ch_Index;
        Enumeration     Enum_Loc;
        Str_30          Str_1_Loc;
        Str_30          Str_2_Loc;
  REG   int             Run_Index;
  REG   int             Number_Of_Runs;

  /* Initializations */
#ifdef ETAS
         setbuf(stdout,0);
         setbuf(stderr,0);
         setbuf(stdin,0);
#endif

  Next_Ptr_Glob = (Rec_Pointer) malloc (sizeof (Rec_Type));
  Ptr_Glob = (Rec_Pointer) malloc (sizeof (Rec_Type));

  Ptr_Glob->Ptr_Comp                    = Next_Ptr_Glob;
  Ptr_Glob->Discr                       = Ident_1;
  Ptr_Glob->variant.var_1.Enum_Comp     = Ident_3;
  Ptr_Glob->variant.var_1.Int_Comp      = 40;
  strcpy (Ptr_Glob->variant.var_1.Str_Comp,
          "DHRYSTONE PROGRAM, SOME STRING");
  strcpy (Str_1_Loc, "DHRYSTONE PROGRAM, 1'ST STRING");

  Arr_2_Glob [8][7] = 10;
        /* Was missing in published program. Without this statement,    */
        /* Arr_2_Glob [8][7] would have an undefined value.             */
        /* Warning: With 16-Bit processors and Number_Of_Runs > 32000,  */
        /* overflow may occur for this array element.                   */

/********** begin GJ changes **************************/
/* comment out printf statements   ******/
//  printf ("\n");
//  printf ("Dhrystone Benchmark, Version 2.1 (Language: C)\n");
//  printf ("\n");
//  if (Reg)
```

```
//  {
//    printf ("Program compiled with 'register' attribute\n");
//    printf ("\n");
//  }
//  else
//  {
//    printf ("Program compiled without 'register' attribute\n");
//    printf ("\n");
//  }
//  printf ("Please give the number of runs through the benchmark:\n ");
//  {
//    int n;
// Comment out and bypass the scanf input --GJ 112901
//    scanf ("%d", &n);
    Number_Of_Runs = 1000000;
//  }
//  printf ("\n");

//  printf ("Execution starts, %d runs through Dhrystone\n", Number_Of_Runs);
/********** end GJ changes ***************************/

/***************/
/* Start timer */
/***************/

#ifdef TIMES
times (&time_info);
Begin_Time = (long) time_info.tms_utime;
#endif
#ifdef TIME
Begin_Time = time ( (long *) 0);
#endif

#ifdef MSC_CLOCK
InitTimer();
Begin_Time = clock();
#endif

/********** begin GJ additions for PowerPC Architecture ************/
/*** Conditional printout to mark beginning of benchmark ***/
#ifdef PRINTOUT
        printf("Dhrystone BENCHMARKS STARTS >>>>>>>>>>>\n\n");
#endif
/***    Initialize timer and get start time value   ***/

        InitTimer();
        BeginTime = clock();

/********** end GJ changes ***************************/

/********* START Dhrystone Benchmark ***********/
  for (Run_Index = 1; Run_Index <= Number_Of_Runs; ++Run_Index)
  {

    Proc_5();
    Proc_4();
      /* Ch_1_Glob == 'A', Ch_2_Glob == 'B', Bool_Glob == true */
```

```
    Int_1_Loc = 2;
    Int_2_Loc = 3;
    strcpy (Str_2_Loc, "DHRYSTONE PROGRAM, 2'ND STRING");
    Enum_Loc = Ident_2;
    Bool_Glob = ! Func_2 (Str_1_Loc, Str_2_Loc);
      /* Bool_Glob == 1 */
    while (Int_1_Loc < Int_2_Loc)  /* loop body executed once */
    {
      Int_3_Loc = 5 * Int_1_Loc - Int_2_Loc;
        /* Int_3_Loc == 7 */
      Proc_7 (Int_1_Loc, Int_2_Loc, &Int_3_Loc);
        /* Int_3_Loc == 7 */
      Int_1_Loc += 1;
    } /* while */
      /* Int_1_Loc == 3, Int_2_Loc == 3, Int_3_Loc == 7 */
    Proc_8 (Arr_1_Glob, Arr_2_Glob, Int_1_Loc, Int_3_Loc);
      /* Int_Glob == 5 */
    Proc_1 (Ptr_Glob);
    for (Ch_Index = 'A'; Ch_Index <= Ch_2_Glob; ++Ch_Index)
                             /* loop body executed twice */
    {
      if (Enum_Loc == Func_1 (Ch_Index, 'C'))
          /* then, not executed */
        {
        Proc_6 (Ident_1, &Enum_Loc);
        strcpy (Str_2_Loc, "DHRYSTONE PROGRAM, 3'RD STRING");
        Int_2_Loc = Run_Index;
        Int_Glob = Run_Index;
        }
    }
      /* Int_1_Loc == 3, Int_2_Loc == 3, Int_3_Loc == 7 */
    Int_2_Loc = Int_2_Loc * Int_1_Loc;
    Int_1_Loc = Int_2_Loc / Int_3_Loc;
    Int_2_Loc = 7 * (Int_2_Loc - Int_3_Loc) - Int_1_Loc;
      /* Int_1_Loc == 1, Int_2_Loc == 13, Int_3_Loc == 7 */
    Proc_2 (&Int_1_Loc);
      /* Int_1_Loc == 5 */

  } /* loop "for Run_Index" */

  /**************/
  /* Stop timer */
  /**************/

#ifdef TIMES
  times (&time_info);
  End_Time = (long) time_info.tms_utime;
#endif
#ifdef TIME
  End_Time = time ( (long *) 0);
#endif
#ifdef MSC_CLOCK
  End_Time = clock();
#endif


/********** begin GJ changes **************************/
```

```
/***** New function for PPC compatible clock *********/
  EndTime = clock();

/* Look at register value for Dhry_Ticks and Seconds to see the value. */
    Dhry_Ticks = (BeginTime - EndTime);

/* The Crystal speed will effect the number of clock   */
/* ticks per second. From table 6-6 of the MPC565      */
/* User's manual, a 4 MHz crystal will generate        */
/* 999,999 ticks per second. A 20 MHz crystal will     */
/* generate a period that is 5 times smaller which is  */
/* 5,000,000 ticks per second. Select the appropriate  */
/* value below for the crystal speed in use.           */

   clock_val = 1000000; /* For a 4 MHz crystal */
// clock_val = 5000000; /* For a 20 MHz crystal */

/* Different internal clock speeds (MF bit=x) will     */
/* have no effect on the number of recorded Dhry_Ticks */
/* since the Dhry_Ticks are recorded off of the        */
/* crystal speed. (If the TBS bit were set to one,     */
/* Then a separate calculation would have to be made   */
/* for each clock speed(ie. 20MHz, 40MHz, 56MHz, etc.) */

  Seconds = ((float) Dhry_Ticks / clock_val);

/*** Print results if PRINTOUT is defined.  ***/
#ifdef PRINTOUT

    printf("Total Execution time of Dhrystone in clock ticks = %d\n", Dhry_Ticks);
  printf("Total Execution time of Dhrystone in seconds = %f\n", Seconds);

  printf("Dhrystone benchmark ends.\n\n");
#endif

/********** end GJ changes ***************************/
/**********Comment out unused printf statements ********/
//  printf ("Execution ends\n");
//  printf ("\n");
//  printf ("Final values of the variables used in the benchmark:\n");
//  printf ("\n");
//  printf ("Int_Glob:            %d\n", Int_Glob);
//  printf ("        should be:   %d\n", 5);
//  printf ("Bool_Glob:           %d\n", Bool_Glob);
//  printf ("        should be:   %d\n", 1);
//  printf ("Ch_1_Glob:           %c\n", Ch_1_Glob);
//  printf ("        should be:   %c\n", 'A');
//  printf ("Ch_2_Glob:           %c\n", Ch_2_Glob);
//  printf ("        should be:   %c\n", 'B');
//  printf ("Arr_1_Glob[8]:       %d\n", Arr_1_Glob[8]);
//  printf ("        should be:   %d\n", 7);
//  printf ("Arr_2_Glob[8][7]:    %d\n", Arr_2_Glob[8][7]);
//  printf ("        should be:   Number_Of_Runs + 10\n");
//  printf ("Ptr_Glob->\n");
//  printf ("  Ptr_Comp:          %d\n", (int) Ptr_Glob->Ptr_Comp);
//  printf ("        should be:   (implementation-dependent)\n");
//  printf ("  Discr:             %d\n", Ptr_Glob->Discr);
```

```
//  printf ("        should be:   %d\n", 0);
//  printf ("  Enum_Comp:         %d\n", Ptr_Glob->variant.var_1.Enum_Comp);
//  printf ("        should be:   %d\n", 2);
//  printf ("  Int_Comp:          %d\n", Ptr_Glob->variant.var_1.Int_Comp);
//  printf ("        should be:   %d\n", 17);
//  printf ("  Str_Comp:          %s\n", Ptr_Glob->variant.var_1.Str_Comp);
//  printf ("        should be:   DHRYSTONE PROGRAM, SOME STRING\n");
//  printf ("Next_Ptr_Glob->\n");
//  printf ("  Ptr_Comp:          %d\n", (int) Next_Ptr_Glob->Ptr_Comp);
//  printf ("        should be:   (implementation-dependent), same as above\n");
//  printf ("  Discr:             %d\n", Next_Ptr_Glob->Discr);
//  printf ("        should be:   %d\n", 0);
// printf ("  Enum_Comp:         %d\n", Next_Ptr_Glob->variant.var_1.Enum_Comp);
//  printf ("        should be:   %d\n", 1);
//  printf ("  Int_Comp:          %d\n", Next_Ptr_Glob->variant.var_1.Int_Comp);
// printf ("        should be:   %d\n", 18);
//  printf ("  Str_Comp:          %s\n",
//                                Next_Ptr_Glob->variant.var_1.Str_Comp);
//  printf ("        should be:   DHRYSTONE PROGRAM, SOME STRING\n");
//  printf ("Int_1_Loc:           %d\n", Int_1_Loc);
//  printf ("        should be:   %d\n", 5);
//  printf ("Int_2_Loc:           %d\n", Int_2_Loc);
//  printf ("        should be:   %d\n", 13);
//  printf ("Int_3_Loc:           %d\n", Int_3_Loc);
//  printf ("        should be:   %d\n", 7);
//  printf ("Enum_Loc:            %d\n", Enum_Loc);
//  printf ("        should be:   %d\n", 1);
//  printf ("Str_1_Loc:           %s\n", Str_1_Loc);
//  printf ("        should be:   DHRYSTONE PROGRAM, 1'ST STRING\n");
//  printf ("Str_2_Loc:           %s\n", Str_2_Loc);
//  printf ("        should be:   DHRYSTONE PROGRAM, 2'ND STRING\n");
//  printf ("\n");

//  Begin GJ changes to Comment out original timing effort
//  User_Time = Begin_Time - End_Time;
//       printf("TotalTime : %X\n", User_Time);
//       printf("Too_Small_Time : %X\n", Too_Small_Time);
//       printf("HZ : %X\n", HZ);
//  if (User_Time < Too_Small_Time)
//  {
//    printf ("Measured time too small to obtain meaningful results\n");
//    printf ("Please increase number of runs\n");
//    printf ("\n");
//  }
//  else
//  {
//#ifdef TIME
//    Microseconds = (float) User_Time * Mic_secs_Per_Second
//                         / (float) Number_Of_Runs;
//    Dhrystones_Per_Second = (float) Number_Of_Runs / (float) User_Time;
//#else
//    Microseconds = (float) User_Time * Mic_secs_Per_Second
//                         / ((float) HZ * ((float) Number_Of_Runs));
//    Dhrystones_Per_Second = ((float) HZ * (float) Number_Of_Runs)
//                         / (float) User_Time;
//#endif
```

```
//
//    printf ("Microseconds for one run through Dhrystone: ");
//    printf ("%6.1f \n", Microseconds);
//    printf ("Dhrystones per Second:                      ");
//    printf ("%6.1f \n", Dhrystones_Per_Second);
//    printf ("\n");
//  }
//     end GJ changes to comment out original timing effort
/********** end GJ changes ***************************/

}

Proc_1 (Ptr_Val_Par)
/******************/

REG Rec_Pointer Ptr_Val_Par;
    /* executed once */
{
  REG Rec_Pointer Next_Record = Ptr_Val_Par->Ptr_Comp;
                                        /* == Ptr_Glob_Next */
  /* Local variable, initialized with Ptr_Val_Par->Ptr_Comp,    */
  /* corresponds to "rename" in Ada, "with" in Pascal           */

  structassign (*Ptr_Val_Par->Ptr_Comp, *Ptr_Glob);
  Ptr_Val_Par->variant.var_1.Int_Comp = 5;
  Next_Record->variant.var_1.Int_Comp
        = Ptr_Val_Par->variant.var_1.Int_Comp;
  Next_Record->Ptr_Comp = Ptr_Val_Par->Ptr_Comp;
  Proc_3 (&Next_Record->Ptr_Comp);
    /* Ptr_Val_Par->Ptr_Comp->Ptr_Comp
                        == Ptr_Glob->Ptr_Comp */
  if (Next_Record->Discr == Ident_1)
    /* then, executed */
  {
    Next_Record->variant.var_1.Int_Comp = 6;
    Proc_6 (Ptr_Val_Par->variant.var_1.Enum_Comp,
           &Next_Record->variant.var_1.Enum_Comp);
    Next_Record->Ptr_Comp = Ptr_Glob->Ptr_Comp;
    Proc_7 (Next_Record->variant.var_1.Int_Comp, 10,
           &Next_Record->variant.var_1.Int_Comp);
  }
  else /* not executed */
    structassign (*Ptr_Val_Par, *Ptr_Val_Par->Ptr_Comp);
} /* Proc_1 */


Proc_2 (Int_Par_Ref)
/******************/
    /* executed once */
    /* *Int_Par_Ref == 1, becomes 4 */

One_Fifty   *Int_Par_Ref;
{
  One_Fifty  Int_Loc;
  Enumeration   Enum_Loc;

  Int_Loc = *Int_Par_Ref + 10;
```

```
  do /* executed once */
    if (Ch_1_Glob == 'A')
      /* then, executed */
    {
      Int_Loc -= 1;
      *Int_Par_Ref = Int_Loc - Int_Glob;
      Enum_Loc = Ident_1;
    } /* if */
  while (Enum_Loc != Ident_1); /* true */
} /* Proc_2 */


Proc_3 (Ptr_Ref_Par)
/******************/
    /* executed once */
    /* Ptr_Ref_Par becomes Ptr_Glob */

Rec_Pointer *Ptr_Ref_Par;

{
  if (Ptr_Glob != Null)
    /* then, executed */
    *Ptr_Ref_Par = Ptr_Glob->Ptr_Comp;
  Proc_7 (10, Int_Glob, &Ptr_Glob->variant.var_1.Int_Comp);
} /* Proc_3 */


Proc_4 () /* without parameters */
/*******/
    /* executed once */
{
  Boolean Bool_Loc;

  Bool_Loc = Ch_1_Glob == 'A';
  Bool_Glob = Bool_Loc | Bool_Glob;
  Ch_2_Glob = 'B';
} /* Proc_4 */


Proc_5 () /* without parameters */
/*******/
    /* executed once */
{
  Ch_1_Glob = 'A';
  Bool_Glob = false;
} /* Proc_5 */


        /* Procedure for the assignment of structures,          */
        /* if the C compiler doesn't support this feature       */
#ifdef  NOSTRUCTASSIGN
memcpy (d, s, l)
register char   *d;
register char   *s;
register int    l;
{
        while (l--) *d++ = *s++;
```

```
}
#endif

};
```

6.3 Listing 6: Dhry2.c

```
/*
 **************************************************************************
 *
 *                   "DHRYSTONE" Benchmark Program
 *                   -----------------------------
 *
 *  Version:    C, Version 2.1
 *
 *  File:       dhry_2.c (part 3 of 3)
 *
 *  Date:       May 25, 1988
 *
 *  Author:     Reinhold P. Weicker
 *
 **************************************************************************
 */

#include "dhry.h"

#ifndef REG
#define REG
        /* REG becomes defined as empty */
        /* i.e. no register variables   */
#endif

extern  int     Int_Glob;
extern  char    Ch_1_Glob;


Proc_6 (Enum_Val_Par, Enum_Ref_Par)
/*********************************/
    /* executed once */
    /* Enum_Val_Par == Ident_3, Enum_Ref_Par becomes Ident_2 */

Enumeration  Enum_Val_Par;
Enumeration *Enum_Ref_Par;
{
  *Enum_Ref_Par = Enum_Val_Par;
  if (! Func_3 (Enum_Val_Par))
    /* then, not executed */
    *Enum_Ref_Par = Ident_4;
  switch (Enum_Val_Par)
  {
    case Ident_1:
      *Enum_Ref_Par = Ident_1;
      break;
    case Ident_2:
      if (Int_Glob > 100)
        /* then */
```

```
      *Enum_Ref_Par = Ident_1;
      else *Enum_Ref_Par = Ident_4;
      break;
    case Ident_3: /* executed */
      *Enum_Ref_Par = Ident_2;
      break;
    case Ident_4: break;
    case Ident_5:
      *Enum_Ref_Par = Ident_3;
      break;
  } /* switch */
} /* Proc_6 */


Proc_7 (Int_1_Par_Val, Int_2_Par_Val, Int_Par_Ref)
/*********************************************/
    /* executed three times                                      */
    /* first call:      Int_1_Par_Val == 2, Int_2_Par_Val == 3,  */
    /*                  Int_Par_Ref becomes 7                    */
    /* second call:     Int_1_Par_Val == 10, Int_2_Par_Val == 5, */
    /*                  Int_Par_Ref becomes 17                   */
    /* third call:      Int_1_Par_Val == 6, Int_2_Par_Val == 10, */
    /*                  Int_Par_Ref becomes 18                   */
One_Fifty       Int_1_Par_Val;
One_Fifty       Int_2_Par_Val;
One_Fifty      *Int_Par_Ref;
{
  One_Fifty Int_Loc;

  Int_Loc = Int_1_Par_Val + 2;
  *Int_Par_Ref = Int_2_Par_Val + Int_Loc;
} /* Proc_7 */


Proc_8 (Arr_1_Par_Ref, Arr_2_Par_Ref, Int_1_Par_Val, Int_2_Par_Val)
/*********************************************************************/
    /* executed once      */
    /* Int_Par_Val_1 == 3 */
    /* Int_Par_Val_2 == 7 */
Arr_1_Dim       Arr_1_Par_Ref;
Arr_2_Dim       Arr_2_Par_Ref;
int             Int_1_Par_Val;
int             Int_2_Par_Val;
{
  REG One_Fifty Int_Index;
  REG One_Fifty Int_Loc;

  Int_Loc = Int_1_Par_Val + 5;
  Arr_1_Par_Ref [Int_Loc] = Int_2_Par_Val;
  Arr_1_Par_Ref [Int_Loc+1] = Arr_1_Par_Ref [Int_Loc];
  Arr_1_Par_Ref [Int_Loc+30] = Int_Loc;
  for (Int_Index = Int_Loc; Int_Index <= Int_Loc+1; ++Int_Index)
    Arr_2_Par_Ref [Int_Loc] [Int_Index] = Int_Loc;
  Arr_2_Par_Ref [Int_Loc] [Int_Loc-1] += 1;
  Arr_2_Par_Ref [Int_Loc+20] [Int_Loc] = Arr_1_Par_Ref [Int_Loc];
  Int_Glob = 5;
} /* Proc_8 */
```

```
Enumeration Func_1 (Ch_1_Par_Val, Ch_2_Par_Val)
/*************************************************/
    /* executed three times                                         */
    /* first call:      Ch_1_Par_Val == 'H', Ch_2_Par_Val == 'R'    */
    /* second call:     Ch_1_Par_Val == 'A', Ch_2_Par_Val == 'C'    */
    /* third call:      Ch_1_Par_Val == 'B', Ch_2_Par_Val == 'C'    */

Capital_Letter   Ch_1_Par_Val;
Capital_Letter   Ch_2_Par_Val;
{
  Capital_Letter        Ch_1_Loc;
  Capital_Letter        Ch_2_Loc;

  Ch_1_Loc = Ch_1_Par_Val;
  Ch_2_Loc = Ch_1_Loc;
  if (Ch_2_Loc != Ch_2_Par_Val)
    /* then, executed */
    return (Ident_1);
  else  /* not executed */
  {
    Ch_1_Glob = Ch_1_Loc;
    return (Ident_2);
   }
} /* Func_1 */


Boolean Func_2 (Str_1_Par_Ref, Str_2_Par_Ref)
/*************************************************/
    /* executed once */
    /* Str_1_Par_Ref == "DHRYSTONE PROGRAM, 1'ST STRING" */
    /* Str_2_Par_Ref == "DHRYSTONE PROGRAM, 2'ND STRING" */

Str_30  Str_1_Par_Ref;
Str_30  Str_2_Par_Ref;
{
  REG One_Thirty        Int_Loc;
      Capital_Letter    Ch_Loc;

  Int_Loc = 2;
  while (Int_Loc <= 2) /* loop body executed once */
    if (Func_1 (Str_1_Par_Ref[Int_Loc],
                Str_2_Par_Ref[Int_Loc+1]) == Ident_1)
      /* then, executed */
    {
      Ch_Loc = 'A';
      Int_Loc += 1;
    } /* if, while */
  if (Ch_Loc >= 'W' && Ch_Loc < 'Z')
    /* then, not executed */
    Int_Loc = 7;
  if (Ch_Loc == 'R')
    /* then, not executed */
    return (true);
  else /* executed */
  {
```

```
    if (strcmp (Str_1_Par_Ref, Str_2_Par_Ref) > 0)
      /* then, not executed */
    {
      Int_Loc += 7;
      Int_Glob = Int_Loc;
      return (true);
    }
    else /* executed */
      return (false);
  } /* if Ch_Loc */
} /* Func_2 */


Boolean Func_3 (Enum_Par_Val)
/***************************/
    /* executed once        */
    /* Enum_Par_Val == Ident_3 */
Enumeration Enum_Par_Val;
{
  Enumeration Enum_Loc;

  Enum_Loc = Enum_Par_Val;
  if (Enum_Loc == Ident_3)
    /* then, executed */
    return (true);
  else /* not executed */
    return (false);
} /* Func_3 */
```

6.4 Listing 7: clock.c

```
/*****************************************************************
 * FILE: CLOCK.C
 * Author: Glenn J.
 * Motorola
 * Date: 12/07/01
 * Copyright @2001 Motorola
 * All Rights Reserved
 *****************************************************************/


void InitTimer();
unsigned long  clock();
void IntHdlr();

static long counter;

// The interrupt handler is not implemented yet.
void IntHdlr()
{ /* Interrupt handler to be located at $900 for ppc decrementer. */

  counter ++;    /* increment roll over count. */
}

void InitTimer()
{ /* Set decrementer to max value. */
```

```
  /* May want this to be a macro. */
  asm ("        lis         r3,0xFFFF");
  asm ("        ori         r3, r3, 0xFFFF");
  asm ("        mtspr       22,r3");
}

unsigned long clock()
{ /* Read PPC decrementer and return count. */
        asm("     mfspr r3,22");
}
```

6.5 Listing 8: Crt0.s

```
;# FILE: CRT0.S: startup for an embedded environment: PowerPC
;# Author: Djonli
;# Motorola GSD (Global Software Division) -- Austin Center
;# Changes: 12/07/01 GJ -- Shortened the file for Dhrystone
;# Date: 03/30/99
;# Copyright @1999 Motorola
;# All Rights Reserved

        .file    "crt0.c"
        .text
        .globl   _start
        .align   2
        addi     r0,r0,0    ;# Debuggers may object to starting at 0.

_start:
;# Disable Watchdog
        addis          r11,r0,0x002F
        ori                        r11,r11,0xC004
        addis    r12,r0,0x0
        ori      r12,r12,0xFF00
        stw      r12,0(r11)

;# Set IMB to Full Speed
        addis    r11,r0,0x0030
        ori      r11,r11,0x7f80
        addis    r12,r0,0x0
        stw      r12,0(r11)

;# Turn on the Time Base
        addis    r11,r0,0x002F
        ori      r11,r11,0xC200
        addis    r12,r0,0x0
        ori      r12,r12,0x1
        sth      r12,0(r11)

;# Turn off serialization for benchmarking purposes
;# Can't modify ICTRL from running code while using a debugger
;# Has to set ICTRL from a debugger !!!
;# Set ICTRL to 0x7 to turn off the show cycles.
        addis    r12,r0,0x0
        ori      r12,r12,0x7
        mtspr    158,r12
        isync
```

```
        nop

;# Enable the Floating Point
    mfmsr  r3
    ori    r3,r3,0x2000
    mtmsr  r3
    isync
    nop

        isync
        sync

        addis    r11,r0,__SP_INIT@ha;# Initialize stack pointer r1 to
        ori      r1,r11,__SP_INIT@l;# value in linker command file.
        addis    r13,r0,_SDA_BASE_@ha;# Initialize r13 to sdata base
        ori      r13,r13,_SDA_BASE_@l;# (provided by linker).
        addis    r2,r0,_SDA2_BASE_@ha;# Initialize r2 to sdata2 base
        ori      r2,r2,_SDA2_BASE_@l;# (provided by linker).
        addi     r0,r0,0  ;# Clear r0.
        stwu     r0,-64(r1);# Terminate stack.

;#
;# Insert other initialize code here.
;#
        bl       __init_main        ;# Finish initialization; call main().
        b        exit           ;# Never returns.
        bl       main             ;# Dummy to pull in main() as soon as
                                  ;# possible.

;#--------------- .init section -----------------------------
        .section .init$00,2,C
        .globl   __init
__init:                   ;# Entry to __init, called by
        mfspr    r0,8     ;# __init_main called above.
        stwu     r1,-64(r1)
        stw      r0,68(r1)

        ;# Linker places .init sections from other modules, containing
        ;# calls to initialize global objects, here.

        .section .init$99,2,C
        lwz      r0,68(r1);# Return from __init.
        addi     r1,r1,64
        mtspr    8,r0
        blr

;#--------------- .fini section -----------------------------
        .section .fini$00,2,C
        .globl   __fini
__fini:                           ;# Entry to __fini, called by exit().
        mfspr    r0,8
        stwu     r1,-64(r1)
        stw      r0,68(r1)

        ;# Linker places .fini sections from other modules, containing
        ;# calls to destroy global objects, here.
```

```
.section .fini$99,2,C
lwz      r0,68(r1);# Return from __fini.
addi     r1,r1,64
mtspr    8,r0
blr
```

7 Document Revision History

Table 2 provides a revision history for this application note.

Table 2. Document Revision History

Rev. No.	Substantive Change(s)	Date of Release
0	Initial release.	January 2003
1	Corrected value in equation in section “Calculating the Results.” Reformatted document to Freescale look and feel.	August 2008

THIS PAGE IS INTENTIONALLY BLANK

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: AN2354
Rev. 1
08/2008

freescale™
semiconductor