ANALOG DEVICES **Engineer To Engineer Note** **EE-141**

Technical Notes on using Analog Devices' DSP components and development tools
Phone: (800) ANALOG-D, FAX: (781) 461-3010, EMAIL: dsp.support@analog.com, FTP: ftp.analog.com, WEB: www.analog.com/dsp

Benchmarking C Code on the ADSP-2106x and the ADSP-2116x Family of DSPs.

Submitted 5/30/2001 by DTL

This application note will detail the process of benchmarking C routines using simple macro functions.

Benchmarking is done to measure the performance of a C compiler, or to understand how many DSP clock cycles a specific section of code will take. Once the number of clock cycles is know, the amount of time that function will take to execute can be quickly calculated using the instruction rate of that processor. For example, the segment of example code on the following page performs the multiplication of 2 vectors, each containing 256 points. When we enable the optimizer and run this section of code through the simulator, we learn that it takes a total of 520 DSP clock cycles. If our DSP is running at 60Mhz, then the total amount of time it takes to complete this loop is 520 (Clock Cycles)/60,000,000 (Clock Cycles/Second) which equals about 8.6 µseconds.

The ADSP-21xxx family of DSPs have a set of registers called EMUCLK and EMUCLK2 which make up a 64-bit counter. This counter is unconditionally incremented during every instruction cycle on the DSP and is not affected by cache-misses, wait-states, etc. Every time EMUCLK wraps back to zero, EMUCLK2 is incremented by one. These registers, while not documented, are great for benchmarking purposes and can be accessed like any universal register (i.e. r0 = EMUCLK;).

Unfortunately, these variables are not directly accessible from C as they are not memory-mapped so we have provided two macros to retrieve the EMUCLK values and compute cycle counts.

The two macros defined below can be copied and pasted into any C file and used freely. The following page contains an example demonstrating their use.

```
#define CYCLE_COUNT_START( cntr ) asm("r0 = emuclk; %0 = r0;":              \
                                   "=k" (cntr):"d" (cntr):            \
                                  "r0")
#define CYCLE_COUNT_STOP( cntr ) asm("r0 = emuclk; r1 = %1; r2 = 4;          \
                                   r0 = r0 - r2; r0 = r0 - r1; %0 = r0;" : \
                                  "=k" (cntr) :                          \
                                  "d" (cntr) : "r0", "r1")
```

The first macro, CYCLE_COUNT_START, copies the contents of EMUCLK into an integer variable. The second macro, CYCLE_COUNT_STOP, subtracts the stored value of EMUCLK from the current value. It then subtracts a value of 4 representing the overhead incurred in these two functions. The final value is then stored back into the integer variable.

The code example on the following page demonstrates the usage of these macros. Try running this code through the simulator with and without the C optimizer enabled!

```
#include <stdio.h>
/* Cycle Count Example Code */


/* Infamous cycle count macros */
#define CYCLE_COUNT_START( cntr ) asm("r0 = emuclk; %0 = r0;":              \
                                      "=k" (cntr):"d" (cntr):              \
                                      "r0")
#define CYCLE_COUNT_STOP( cntr ) asm("r0 = emuclk; r1 = %1; r2 = 4;            \
                                      r0 = r0 - r2; r0 = r0 - r1; %0 = r0;" : \
                                     "=k" (cntr) :                           \
                                     "d" (cntr) : "r0", "r1")

// test vectors
float dm Vector_A[256];
float pm Vector_B[256];
float dm Vector_C[256];

int cnt0;           // does not have to be global

main()
{
       int i;

       // read contents of EMUCLK and store in cnt0
       CYCLE_COUNT_START(cnt0);

       // perform loop
       for (i=0; i<256;i++)
       {
             Vector_C[i] = Vector_A[i] * Vector_B[i];
       }

       // calculate total number of cycles and store result in cnt0
       CYCLE_COUNT_STOP(cnt0);

       // print the results of the benchmark
       printf("The cycle count for vector multiplication execution was %d cycles.\n",cnt0);
       printf("The cycle count for each vector element was %f.\n",((float)cnt0*(1.0/256.0)));
}
```

These macros do not take into account the potential wrapping of the EMUCLK register back to zero and the subsequent increment of the EMUCLK2 register in order to save space and execution time. If benchmarking is to be done on a free-running system, the EMUCLK register will wrap once every 71 seconds on a 60Mhz DSP. However, since most benchmarking is typically done in a simulation environment or in a controlled emulation environment, a EMUCLK wrap is typically never encountered as these registers are reset with the processor.