Michael Kellett
Electronics and Software Consultant

Comparing AD218X and TI54XX family DSP performance

Introduction

The aim of this experiment is to compare the performance of the processors and manufacturers C compiler tools using a very simple benchmark. The idea is to use a simple C coding style and see how well the compilers and the chips cope with it.

The end result is the number of *effective* MAC (multiply accumulate) operations the chips achieve per clock cycle – both families can perform one MAC every cycle as a peak rate.

The Benchmark Code

The plan was to call a two section biquad filter function (from the manufacturers library) from a for() loop 100 times and use the manufacturers simulators to count the number of cycles used between two marker statements.

This is not an efficient coding style in terms of performance (due to the overhead involved in calling the library function) but it gives a good idea of the performance that will be achieved without a lot of processor and tool specific tuning.

The experiment went well as far as the AD218X was concerned but the TI tools available (from the ‘5402 DSP Starter Kit) did not include a simulator. The TI code was timed running on real hardware using profiling from the Code Composer development suite. To confuse matters a little more TI do not supply a straight single sample biquad filter routine but one which takes a vector input. This is slower for single sample operation but faster when burst processing is allowable.

Each biquad section needs 5 MAC operations to compute so really efficient code should approach 7 or 8 cycles per biquad or better than 0.5 MACs per cycle.

```
AD2181 code

#include <filters.h>
int pm coeffs[] = {1,2,3,4,5,6,7,8,9,10};                    /* set up filter coefficients, meaningless filter
                                                             data */
int input, output, state[5];                                 /* declare globals for input, output and the filter
                                                             state */
int main(void)
{
int i;
int j;
for(i=0;i < 5;i++)                                   /* initialise the filter state */
        {
        state[i] = 0;
        }
j = 0;                                               /* place to set break point */
for(i = 0; i < 100; i++)
        {
        input = i;
        output = biquad(input, coeffs, state, 2);            /* call the filter 100 times */
        }
j = 1;                                               /* place to set breakpoint */
}
```

```
TI5402 Code

#include <board.h>
#include <math.h>
#include <stdio.h>
#include <tms320.h>
#include <dsplib.h>


void delay(int);


#define NX 1
#define NBIQ 2

DATA x[NX] ={
-617};

#pragma DATA_SECTION (h,".coeffs")
DATA h[5*NBIQ] ={ /* C54x: a1 a2 b2 b0 b1 ... */
30857,
30152,
-26684,
-18838,
4924,
32118,
14852,
-30453,
22232,
-3184,
};


#pragma DATA_SECTION (dbuffer,".dbuffer")
DATA dbuffer[2*NBIQ];
DATA *dp = dbuffer;
DATA r[NX];

void main()
{
short i;


for (i=0;i<NX;i++) r[i] =0;                     // clear output buffer (optional)
for (i=0; i<2*NBIQ; i++) dbuffer[i] = 0;        // clear delay buffer (a must)

brd_init(100);                                  /* initialise the development board */


while (1)
        {
        brd_led_toggle(BRD_LED0);               /* twinkles an LED to show its running */
        delay(100);
        }
}


void delay(int period)
{
int i;

for(i=0; i<period; i++)
        {
        iircas5(x,h,r,&dp,NBIQ, NX);
    }
}
```

Results

The AD2181 simulation required 5913 cycles for 100 loops which is 5.9 cycles per MAC or 0.17 MACs per cycle.

The TI5402 processor required:

NX	Cycles	Total MACs	MAC/cycle
1	11212	1000	0.089
4	20812	4000	0.192
8	33612	8000	0.238
16	59212	16000	0.270

Summary

In this simple test the AD218X toolset and chip gave better straight out of the box results than the TI equivalents.

The TI vectorised filter gave much better results when samples were processed in a block (as would be expected).

The same technique could be adopted with the AD processor but there is no standard library function to implement it.

The conclusion from this simple experiment is that to get anything like the best performance from these processors at least some hand crafting of assembler code will be required.

In order to investigate the possibilities further a more realistic benchmark task is required (because the present simple one can be optimised to nothing !).

If you would like to discuss low power DSP techniques or low power low frequency analogue design problems please contact me.

Michael Kellett,

Electronics and Software Consultant.

mk@mkesc.co.uk   www.mkesc.co.uk