MOTOROLA
SEMICONDUCTOR
APPLICATION NOTE

Order this document by: AN1215/D

PID Routines for MC68HC11K4 and MC68HC11N4 Microcontrollers

By James W. Gray

INTRODUCTION

PID (proportional, integral, derivative) compensation is one of the most common forms of closed-loop control. Control of closed-loop systems that require compensation is a growing area of application for embedded microprocessors. In these systems, analog signals must be converted into discrete digital samples before compensation or filtering can take place. Loop performance dictates the sampling rate, and calculations must be complete before the next sample time begins. These loop-related constraints and the Nyquist frequency requirement place an upper bound on digital control of closed systems with feedback error. If the controlled system has a resonance or other behavior with a time constant shorter than the sample and calculation time, chaos is the most likely outcome. Despite these limitations, increases in microprocessor clock rates and the addition of on-chip control-oriented hardware are expanding the number of medium performance control applications handled by 8-bit machines. While an expensive DSP-class processor is the correct choice for the most demanding applications, several members of the M68HC11 family have the speed and resources to control multiple PWM channels.

This note provides two working examples of PID control-loop software. The first example, written primarily in C, shows a PID algorithm in a straightforward way using floating-point math. Key features of the C environment are covered for readers who are more used to assembly language. The second example implements a PID algorithm in assembly language. It uses the MC68HC11N4 on-chip math coprocessor to speed up arithmetic operations.

Both examples are complete and ready to run on a Motorola M68HC11EVS evaluation board. External interfacing is identical for both examples — an 8-bit analog to digital converter is used for input, and an 8-bit PWM waveform is output. Because the code in both examples carries more than 16 bits of precision, and because both processors support 16-bit PWM, only minor changes are needed to increase precision. Power amplifiers, sensors, and other interface circuitry must be supplied in order to experiment with real-world systems — a simple RC circuit is used for software checkout.

C and assembly language source code and loadable object code can be obtained from Motorola Freeware Data Systems. Modem access: (512) 891-3733. Internet access: freeware.aus sps.mot.com. Web access: http:///www.freeware.aus.sps.mot.com.

THE MICROCONTROLLERS

The MC68HC11K4 and MC68HC11N4 are 16-MHz devices with nonmultiplexed external address and data buses. Each has 24 Kbytes of on-chip ROM or EPROM. Both devices have multiple PWM channels with programmable period, duty cycle, polarity, and clock source. In both, two 8-bit channels can be concatenated to generate a 16-bit PWM output. The MC68HC11N4 also has two additional 12-bit PWM channels and two digital to analog converter channels with 8-bit resolution.

The MC68HC11N4 is particularly well-suited to PID computation because it has an on-chip math coprocessor that performs 16- and 32-bit multiplication and division, fractional division, and multiply-and-accumulate operations. All operations are done by means of memory-mapped registers, thus preserving the standard M68HC11 family instruction set. Multiplication and fractional division are complete in a maximum of 5 microseconds while integer division requires a maximum of 8.75 microseconds.

PID ALGORITHM

The general flow of a controlled system with PID compensation is shown in **Figure 1**. An informal review of the derivation of the discrete form of each term and how they function follows. **REFERENCES** 2 and 3 provide a complete analysis of digital PID control.

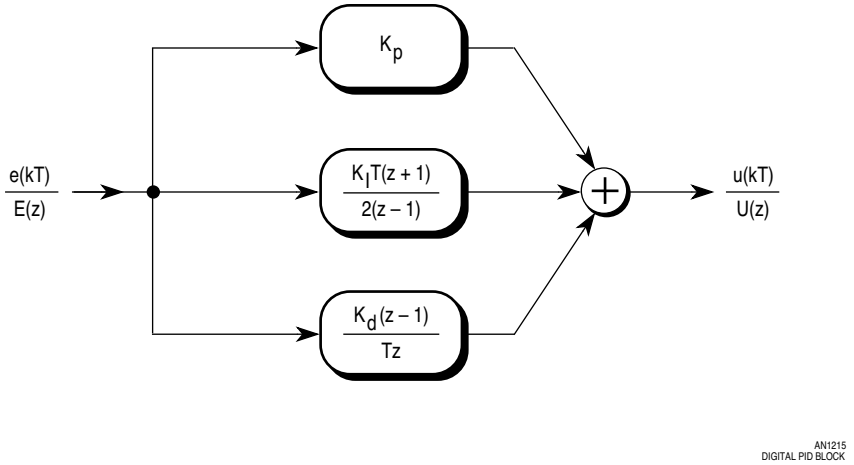

Figure 1 PID Flow Diagram

There is a desired setpoint in our process (Gd) and a measurement of the actual value G(t) in time. Error is:

$$e(t) = Gd - G(t)$$

Output correction x(t) for the PID controller is:

$$x(t) = KP\ e(t) + KI\int e(t) + KD\frac{de(t)}{dt} \mid t = T$$

where KP, KI, and KD are constants.

Now, rewriting the integral:

$$x(t) = KP\ e(t) + KI\int_{t=0}^{t} [Gd - G(t)]dt + KD\frac{de(t)}{dt} \mid t = T$$

To introduce discrete time, let t = kT where k = 1,2,...,n and T = the sampling and control update period. Now, t0 = (k – 1)T. The integral evaluated from (k – 1)T to kT can be approximated using the trapezoidal integration rule. The derivative of the error term is simply the rate of change of error, but this can be noisy over one period. Using a four-point central-weighted average for the difference term is a practical way to deal with this on a microprocessor.

The form which can be executed directly on the microprocessor is:

$$x(t) = KP\ e(t) + KI\ (Gdt - \frac{T}{2}(G(Kt) + G[(k-1)T])) + \frac{KD}{6T}\ ((e(kT) - e(k-3) + 3(e(k-1) - e(k-2)))$$

This term is added to the current output and put into the PWM control register at the beginning of the next calculation cycle. Substituting the microcode labels for constants and variables into EQ. 4 and using C language operator notation gives:

NEWDTY = KP ∗ (ERRX) + KI ∗ PERDT ∗ (CMNDX - (ADRCX + ADRCXM1) / 2) + (KD / (6 ∗ PERDT)) ∗ ((ERRX - ERRM3X) + 3 ∗ (ERRM1X - ERRM2X)) + OLDDTY

The function of the proportional term is clear, but the derivative and integral terms may need a brief explanation. When a system with only proportional control is off the specified setpoint, the controller will increase the control voltage until the error signal is zero, and the system thus returns to the setpoint with more applied voltage than is required for maintaining equilibrium. This causes overshoot and, as the process continues, under-damped ringing. The derivative term contributes proportionally to the error rate of change, but with the opposite sign of the proportional term. If the proper constants are chosen, critical damping can be achieved. The role of the integral term is to eliminate steady state error. A system that has a steady state error when tracking a ramping input function can use an integral term to integrate the error over time and compensate for it.

C LANGUAGE IMPLEMENTATION

This version of the PID control routine illustrates use of high-level language for control applications. High-level language offers many conveniences not available in assembly language. Here are a few instances:

Memory-mapped registers can be defined in a single file, which can be included in any function.

Since C is a strongly-typed language, the compiler can identify data-type mismatches, such as writing a 16-bit integer to an 8-bit port.

Most C compilers for the M68HC11 family allow direct vectoring to interrupt service routines. Interrupt functions are usually defined for a special memory segment with a base address and a function name:

```
interrupt [INTVEC_START + 26] void RTI_interrupt(void);
```

When an interrupt service routine is written it can be declared as an interrupt function instead of a normal subroutine:

```
interrupt void RTI_interrupt(void)
```

The function is compiled with a terminating RTI instruction, eliminating the need for a patch between hardware interrupts and C subroutines. The void statements indicate that no arguments are being passed to or from the interrupt routine.

One of the most attractive features of contemporary C compilers is the ability to add floating point math with an “include math.h” statement. Even when the final application can't afford the time or code space for floating point calculations, use of floating point math during debugging provides an excellent means of testing new algorithms.

Now to examine the control routine itself (refer to Appendix A for a complete C listing). After necessary files are included and floating point variables are declared, a prototype is given to define an assembly language function that is used later.

The main program initializes constants and variables, sets up the required on-chip peripherals, and waits for interrupts to occur. The M68HC11 real-time interrupt (RTI) is used to establish a precise time base for performing PID compensation. The period (T or PERDT) is determined by the RTI rate. In these examples, the period is 16.383 milliseconds, but this value is arbitrary — in real applications, the performance of the controlling microprocessor and the requirements of the controlled system determine the period.

The RTI_interrupt function is a workhorse. It does PID loop PWM duty cycle calculations, performs I/O using the DOIO assembly language function, and checks the results against the usable PWM output range of $00 to $FF. If a floating-point result is out of range, the closest limit is substituted. This “saturation arithmetic” prevents out-of-range results from causing sign-reversals in the PWM output.

Details of error-checking and the DOIO subroutine are best understood by looking at the format of the floating-point variables. The four byte format is:

SEEEEEEE EMMMMMMM MMMMMMMM MMMMMMMM

S represents a sign bit, E represents an 8-bit exponent biased by 127, and M represents a 23-bit fractional mantissa (significand) with an implicit leading 1. The I/O range of $00 to $FF is scaled into the eight most significant bits of a floating-point variable, giving a floating-point range of 1.0 ($3F800000) to 1.998046875 ($3FFF8000). The following expression is used to evaluate a floating-point number:

$$F = [(-1)^S] [2^{(E-127)}] [1.M]$$

When the RTI_interrupt function returns control to the C routine, the last task the routine must perform is preparation for the next period. A two-element pipeline of the A/D reading and a four-element error pipeline are updated. Finally the “old” duty cycle value is copied into OLDDTY.

MC68HC11N4 MATH COPROCESSOR

The assembly PID routine uses the MC68HC11N4 math coprocessor, which is commonly referred to as an arithmetic logic unit, or ALU. The ALU performs 32/16-bit division, 16/16 multiplication, multiply-and-accumulate operations, and 16/16-bit fractional division without CPU intervention. As **Figure 2** shows, the coprocessor has one control register, one status register, and three data registers. Arrows indicate the most convenient order of writing the registers.

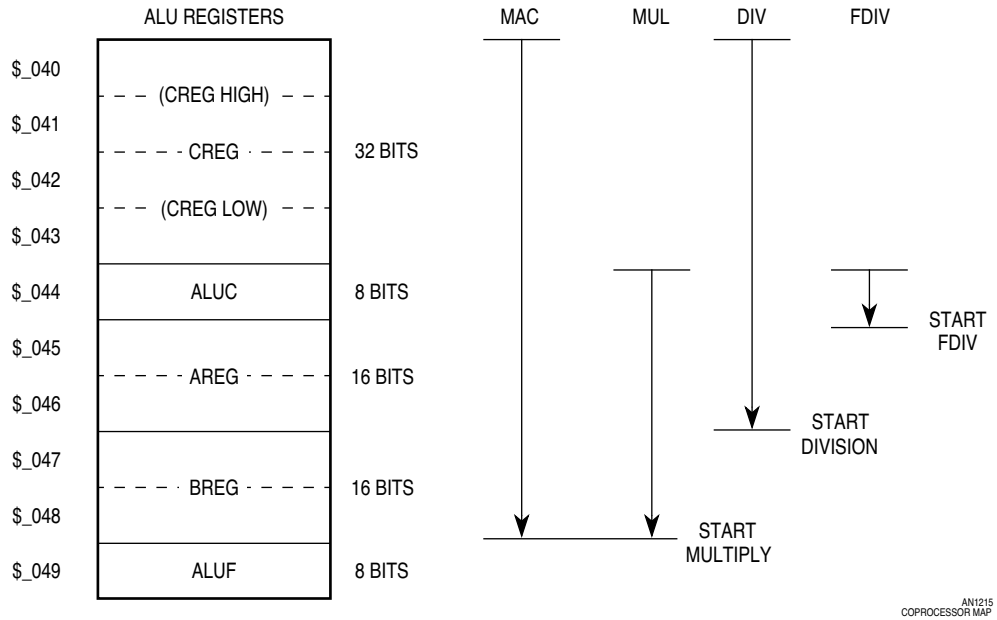

Figure 2 Coprocessor Registers and Operations

The 8-bit ALU control register (ALUC) controls ALU operation. **Table 1** summarizes ALUC bit functions.

ALUC — Arithmetic Logic Unit Control **$0044**

	Bit 7	6	5	4	3	2	1	Bit 0
	SIG	DIV	MAC	DCC	TRG	—	—	—
RESET:	0	0	0	0	0	0	0	0

SIG — Signed Number Enable

0 = AREG, BREG, and CREG contents are unsigned numbers

1 = AREG, BREG, and CREG contents are signed numbers

DIV — Division Enable

MAC — Multiply with Accumulated Product Enable

DCC — Division Compensation for Concatenated Quotient Enable

TRG — Function Start Trigger Bit

Always reads zero

0 = No effect

1 = Start ALU

Bits [2:0] — Not implemented

Always read zero

Table 1 ALUC Bit Function

SIG	DIV	MAC	DCC	FUNCTION	START TRIGGERS
0	0	0	X	Unsigned MUL	Write BREG or set TRG
1	0	0	X	Signed MUL	Write BREG or set TRG
0	0	1	X	Unsigned MAC	Write BREG or set TRG
1	0	1	X	Signed MAC	Write BREG or set TRG
0	1	0	X	Unsigned IDIV	Write AREG or set TRG
1	1	0	0	Signed IDIV	Write AREG or set TRG
1	1	0	1	Signed IDIV DCC	Write AREG or set TRG
0	1	1	X	Unsigned FDIV	Set TRG
1	1	1	0	Signed FDIV	Set TRG
1	1	1	1	Signed FDIV DCC	Set TRG

The 8-bit ALU status register (ALUF) signals when an operation is complete and indicates the status of the completed operation.

ALUF — Arithmetic Logic Unit Status Flag Register **$0049**

	Bit 7	6	5	4	3	2	1	Bit 0
	NEG	RZF	—	—	—	OVF	DZF	ACF
RESET:	0	0	0	0	0	0	0	0

NEG — Negative Result

NEG is set if the result is a negative value. This is a read-only-bit. Writes to this bit do not affect the value.

RZF — Remainder Equals Zero Flag

RZF is set if the remainder is zero.

Bits [5:3] — Not implemented

Always read zero

OVF — Overflow Flag

OVF is set if overflow from MSB on CREG is detected. This bit is cleared automatically by a write to this register with bit 2 set.

DZF — Divide by Zero Flag

DZF is set if a divide by zero condition is detected. DZF is cleared automatically by a write to this register with bit 1 set.

ACF — Arithmetic Completion Flag

ACF is set by completion of the arithmetic operation. ACF is cleared automatically by a write to this register with bit 0 set.

Data register A (AREG) can hold either a 16-bit multiplicand or a 16-bit divisor. Data register B (BREG) can hold either a 16-bit multiplier or a 16-bit remainder after division. Data register C (CREG), which is treated as two 16-bit registers (CREG High and CREG Low), can hold a 32-bit product or accumulated product after multiplication, or it can hold a 32-bit numerator before division and a 32-bit quotient after division. There is an implied fixed radix point to the right of bits AREG0, BREG0, and CREG0.

Table 2 shows numeric ranges of ALU registers. **Table 3** shows signed expression of numbers. Table 4 shows fractional numeric representation. **Figure 3** shows typical data register formats.

Table 2 Numeric Ranges of ALU Registers

Register	Size	Unsigned	Signed
AREG	16 bits	0 to 65535	–32,768 to +32,767
BREG	16 bits	0 to 65535	–32,768 to +32,767
CREG	32 bits	0 to 4,294,967,295	–2,147,483,648 to +2,147,483,647

Table 3 Representation of Signed Numbers

Decimal	16-Bit Hexadecimal	32-Bit Hexadecimal
+2,147,483,647	—	$7FFF FFFF
•	—	•
•	—	•
+32,767	$7FFF	$0000 7FFF
•	•	•
•	•	•
+1	$0001	$0000 0001
0	$0000	$0000 0000
–1	$FFFF	$FFFF FFFF
–2	$FFFE	$FFFF FFFE
•	•	•
•	•	•
–32,768	$8000	$FFFF 8000
•	—	•
•	—	•
–2,147,483,647	—	$8000 0000

Table 4 Representation of Fractions

Decimal	16-Bit Hexadecimal
+0.99998	$FFFF
•	•
•	•
+0.5	$8000
+0.25	$4000
+0.125	$2000
+0.0625	$1000
+0.03125	$0800
+0.015625	$0400
•	•
•	•
+0.0000153	$0001
–0.99998	$0001
•	•
•	•
–0.5	$8000
–0.25	$C000
–0.125	$E000
–0.0625	$F000
–0.03125	$F800
–0.015625	$FC00
•	•
•	•
–0.0000153	$FFFF

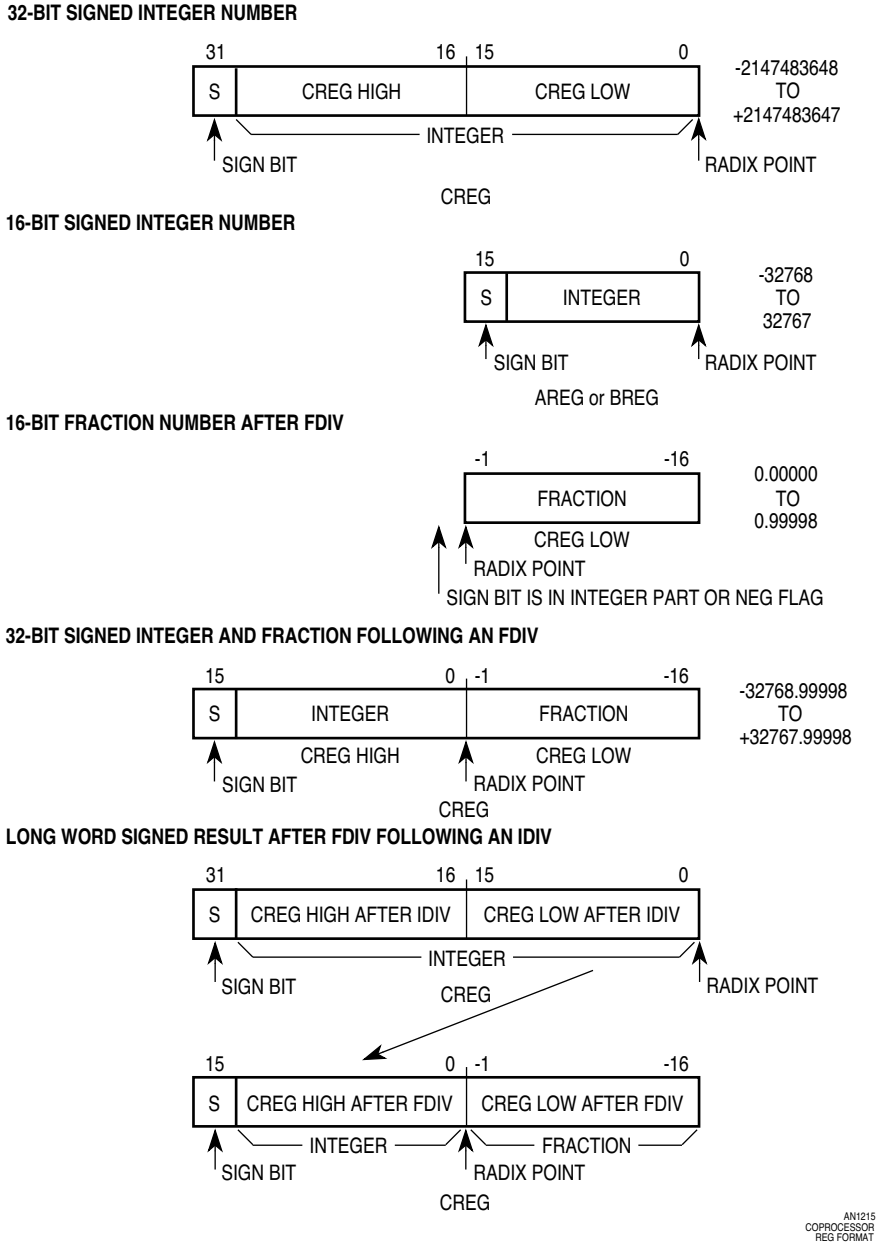

Figure 3 ALU Data Register Format

ASSEMBLY LANGUAGE IMPLEMENTATION

The assembly language version of the PID control routine consists of a C master program and an assembly subroutine (DOPID) that carries out all time-critical tasks (see Appendix B for a complete listing). The C master program is used for convenience — DOPID can be made to stand alone with minor changes in variable definition.

The constant and variable number representation for the assembly version of the PID loop is a four-byte number consisting of a 16-bit integer, an implied decimal point, and a 16-bit fraction. This representation was chosen to provide relatively high performance while giving enough precision to support MC68HC11N4 8-, 12-, and 16-bit PWM resolutions. Many other formats could be used, each with a particular trade-off between precision and time.

Although the sampling period constant and the constants of each term of the PID algorithm could be calculated during the assembly phase and set in the final code, these values are instead initialized as ratios of hexadecimal integers, then calculated in real time in the loop. This arrangement allows easy experimentation with these values without reassembling each time a value is changed. The initial format of these values (numerator and denominator) is a consequence of the four-byte numeric representation.

Here are some examples of ALU operations performed during PID computation.

To perform a signed integer divide followed by a signed fractional divide, the numerator is written to CREG, $C0 is written to ALUC, and the divisor is written to AREG. After the completion flag in ALUF is set ($49), $E8 is written to ALUC. Following integer division, the quotient is in CREG and the remainder is in BREG. Fractional division moves the content of CREG Low to CREG High, then places a 16-bit fraction in CREG Low. The final result is a 16-bit quotient in CREG High and a 16-bit fraction in CREG Low. There is an implicit decimal point between CREG High and CREG Low. More precision can be obtained by concatenating fractional divisions.

To perform a signed multiply, $80 is written to ALUC and multiplicands are written to AREG and BREG. When the operation is complete, a 32-bit result is in CREG.

The actual range used for control is hexadecimal $0000.0000 to $00FF.FFFF (decimal 0 to 255.99998). While the error term can be positive or negative, PWM output and feedback voltage are always positive. A result greater than $FF is treated as an overflow, and a result less than $00 is treated as an underflow. This effects a saturated value of the correct sign as in the C version. Except for the expression of initial constants as ratios, formula 5 is not changed. In the derivative term the factor

(KDNUM / KDDEN) / ((6 ∗ (PERDTNUM / PERDTDEN))

is rearranged to

((KDNUM / KDDEN) ∗ PERDTDEN) / (6 ∗ PERDTNUM).

The DOPID routine is written as straight-line code. Only two subroutines and a limit-checking section are shared by the proportional, integral, and derivative terms. The subroutines MULLNG and ADLNG are used to multiply and add terms and factors expressed in the special four-byte format explained previously. Each of the P, I, and D terms use ADLNG to contribute to the new PWM duty cycle (NEWDTY), but, as in the C version of the PID routine, NEWDTY is not output until the beginning of the next period. Only the 8-bit integer portion is used, and the 1-bit round-off error this causes is not corrected — the effect is negligible in this 8-bit example. After all three terms are calculated, control is returned to the master C routine. The master routine updates the error and A to D pipelines, then enters the main wait loop.

During program execution, all results and most intermediate values are kept in RAM, rather than on the stack. Controller state can easily be inspected by means of a single breakpoint and a dump of the appropriate variable address. Variable addresses are provided in the C startup code for the assembly routine — the addresses are only valid for this compilation and can change with code revision.

HARDWARE PLATFORM

The Motorola M68HC11KMNPEVS Evaluation System can be used to run both versions of the PID routine. With an MC68HC11K4 inserted in the emulator module, only the floating point version will execute. With an MC68HC11N4 inserted, both versions can be executed — the floating point version simply does not use the math coprocessor.

Both versions of the PID routine utilize special test mode in the EVS system. This means that the M68HC11 processor vectors are mapped from $BFD6 to $BFFF instead of from $FFD6 to $FFFF, and can be placed in user RAM or in emulation RAM, making experimentation with varied processor configuration options easier. Refer to the *M68HC11KMNPEVS Evaluation System User's Manual* for more information.

S-record formatted object code can be loaded and run on the EVS using appropriate serial communications software. The default EVS Baud rate is 19200 Baud.

A simple RC circuit is used to provide the feedback necessary for the software PID loop to operate. The connections to the EVS are shown in **Figure 4**. PH0 is the processor PWM output and PE0 is the channel 1 A/D input. Note that the A/D reference inputs must be connected to appropriate supplies and the SUP6(F(,IRQ)) line must also be tied high. A two-channel oscilloscope can be used to observe PWM output and controlled voltage. Both versions of the code write $FF to PORTA just before performing PID loop calculations and then write $00 to the port just after calculations are complete, allowing execution times to be observed with a scope.

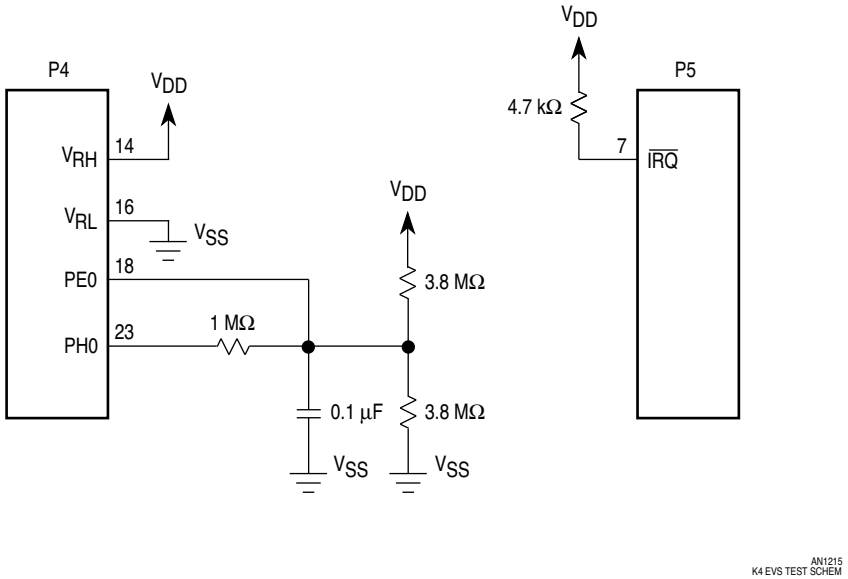

Figure 4 Evaluation System Schematic

PERFORMANCE

Since two very different approaches were taken to implement the PID loop algorithm, it is not surprising that the performance and code sizes of the two routines differ significantly. The C language version contains over 1500 bytes, much of which consists of floating point runtime support. Because it carries full floating-point precision and does not use math support hardware, the C version takes approximately 6 milliseconds to complete the loop. The assembly version contains approximately 1000 bytes, and could be reduced to about 800 bytes if features added for clarity and experimentation were removed. Because it uses a tailored arithmetic format and gets a hardware assist from the math coprocessor, the assembly version completes the loop in approximately 700 microseconds. In both cases, performance could be improved by precomputing all the results with constant factors.

EXPERIMENTS AND EXTENSIONS

To develop a more intuitive understanding of PID loop function, try varying the term constants in the object code and observing the effect on controller performance. For the C version, it is best to change the values in the C source and recompile. For the assembly version, it is easy to recalculate and change denominators of the constants: only decimal-to-hex conversion is required. The PERDT constant must be changed when the RTI interrupt rate is changed, or the time base of the algorithm will be destroyed. Try adjusting the proportional and derivative constants to give good observable control and then start substituting smaller value resistors for R1. Eventually, the substitution will cause an unstable underdamped system.

Many extensions to the routines are simple yet interesting. The command voltage level can be read from one of the seven unused A/D channels to give real-time control points. The MC68HC11N4 version could have analog output instead of PWM with the use of an 8-bit D/A channel. The N4 version could also use an infinite impulse response (IIR) filter implemented by means of the math coprocessor multiply-and-accumulate capability, rather than utilizing the noise-canceling effect of the derivative term's four point central difference.

CONCLUSION

With the 16.383 millisecond sampling rate chosen, the floating point C routine cannot service all four MC68HC11K4 PWM channels. It could, however, service one higher-rate channel and several less time-critical processes. The assembly math coprocessor routine can service all the MC68HC11N4 PWM channels (four 8-bit and two 12-bit) in 4.2 milliseconds, leaving 75% of execution time for other tasks. If the MC68HC11N4 D/A channels were also used, six independent PID controllers could be run in 5.6 milliseconds — this routine would use less than a third of the allowable processing time and require only half of the twelve MC68HC11N4 A/D input channels.

The two approaches to PID control outlined in this note encompass a wide range of useful and cost-effective applications. The simplicity of the coded C algorithm is very appealing. Since the C routine carries virtually no code-space overhead, it is very well-suited to an application where floating-point math is already required, and a sampling rate in the 20-millisecond range is acceptable. Because it uses a number format adapted to the application and to use of on-chip resources, the assembly routine generally yields higher, more cost-effective performance than the C routine.

A final word of caution. There is no substitute for thorough mathematical analysis of the system to be controlled in the discrete time domain. References 2 and 3 contain detailed discussions of control systems and analytic techniques.

REFERENCES

1. Andrews, Michael. Programming Microprocessor Interfaces for Control and Instrumentation. Prentice Hall; 1982
2. Kuo, Benjamin. Digital Control Systems. Holt, Rinehart and Winston Inc.; 1980
3. 3.Kuo, Benjamin. Automatic Control Systems. Prentice Hall; 1987
4. 4.Motorola. MC68HC11K4 Technical Summary, BR751/D
5. 5.Motorola. MC68HC11N4 Technical Summary, MC68HC11N4TS/D
6. 6.Motorola. M68HC11 Reference Manual, M68HC11RM/AD Rev. 3
7. 7.Motorola. M68HC11K4EVS Evaluation System User's Manual, M68HC11K4EVS/D1

APPENDIX A
C LANGUAGE PID ROUTINE

A.1 Main C Routine

```
#include <stdio.h>
#include <io6811k4.h>
#include <int6811k.h>
#include <math.h>


zpage   unsigned int    TOFCOUNT;               /* declare variables */
zpage   float           CMNDVX;
zpage   float           ADRCX;
zpage   float           ADRCXM1;
zpage   float           ADRCXM2;
zpage   float           ADRCXM3;
zpage   float           ERRX;
zpage   float           ERRM1X;
zpage   float           ERRM2X;
zpage   float           ERRM3X;
zpage   float           PERDT;
zpage   float           NEWDTY;
zpage   float           OLDDTY;
zpage   float           KP;
zpage   float           KD;
zpage   float           KI;


extern int   DOIO (void);                   /* prototype for assembly
                                               routine */


void main()             /* main program */
{
CMNDVX = 1.5;
PERDT = 0.016383;       /* RTI and therefore PID loop period = 16.383 ms */
KP = 0.18;              /* kp = .12, ki = 6.0, kd = .006, for 1 M ohm drive */
KI = 6.0;
KD = 0.009;
OLDDTY = 1.9;           /* start out with pwm set fairly high */
PORTA = 0x00;           /* this will be used for a scope trigger */
DDRA =  0xFF;           /* set PORTA as output */
PACTL = 0x03;           /* set RTI to 16.383 ms (E = 4 MHz) */
TMSK2 = 0x40;           /* enable RTI interrupts */
OPTION = 0x90;          /* enable A/D charge pump */
PWPER1 = 0xFF;          /* set up PWM channel 1 at 15.625 kHz */
PWDTY1 = 0xFF;          /* with positive polarity */
PWPOL = 0x01;
DDRH = 0x00;
PWEN = 0x01;
TFLG2 = 0x40;
enable_interrupt();     /* wait here for RTI to cause loop execution */
wait_for_interrupt();

for (;;) {
  ; }
}

interrupt void IRQ_interrupt(void)  /* should initialize all interrupts... */

{
  PORTA = 0xFF;
  PORTA = 0x00;
}

interrupt void TO_interrupt(void)

{
  TOFCOUNT++ ;
}
```

```
interrupt void RTI_interrupt(void)      /*PID LOOP/PWM routine */

{
  PORTA = 0xFF;                         /* scope strobe */

  DOIO();                               /* read A to D and output the duty
                                        cycle calculated last period */

  ADCTL = 0x10;                         /* begin new conversion cycle */

  ERRX = (CMNDVX - ADRCX);              /* calculate current error */

  /* The statement below is the entire floating point PID loop */

  NEWDTY = KP*(ERRX) +  KI*PERDT*(CMNDVX - (ADRCX + ADRCXM1)/2)
           + (KD/(6*PERDT))*((ERRX - ERRM3X) + 3*(ERRM1X - ERRM2X))
           + OLDDTY;

  if (NEWDTY > 1.99609)                 /* test for result being in usable */
      NEWDTY = 1.99609;                 /* limits and set PWM duty cycle if */
  else if (NEWDTY < 1.0)                /* beyond saturation */
            NEWDTY = 1.0;

  TFLG2 = 0x40;                         /* clear RTI flag */
  ADRCXM1 = ADRCX;                      /* update A/D result for next cycle */
  ERRM3X = ERRM2X;                      /* update error pipeline */
  ERRM2X = ERRM1X;
  ERRM1X = ERRX;
  OLDDTY = NEWDTY;                      /* update old duty cycle for next
                                           calculation period */

  PORTA = 0x00;                         /* scope strobe */
}
```

A.2 DOIO Subroutine Assembler Listing

```
 1                ********************************************************
 2                *       DOIO assembly function                         *
 3                *       This routine handles the conversion between    *
 4                *       8 bit register values and the C float variables *
 5                *                                                      *
 6                ********************************************************
 7
 8  0000                  MODULE  DOIO
 9  0000                  PUBLIC  DOIO
10
11  0000                  P68H11
12  0000                  RSEG    CODE
13
14  006C          PWDTY1  set     $006c                   REGISTER LOCATIONS
15  0031          ADDR1   set     $0031
16  0000                  EXTERN  ADRCX:ZPAGE             EXTERNAL VARIABLE LOCATIONS
17  0000                  EXTERN  NEWDTY:ZPAGE
18  0000          DOIO:
19  0000 863F             LDAA    #$3F
20  0002 5F               CLRB
21  0003 DD00             STD     ADRCX                   INITIALIZE FLOAT LOCATION.
22  0005 9631             LDAA    ADR1                    GET CHANNEL 1 A/D RESULT.
23  0007 04               LSRD                            SHIFT TO FLOAT MANTISSA POSITION.
24  0008 8A80             ORAA    #$80                    OR IN LEAST SIGNIFICANT EXP BIT
25  000A DD01             STD     ADRCX+1                 AND STORE IT IN FLOAT VARIABLE.
26  000C 5F               CLRB                            CLEAR LEAST SIGNIFICANT
27  000D D703             STAB    ADRCX+3                 FLOAT BYTE.
28  000F DC01             LDD     NEWDTY+1                GET TWO BYTES OF FLOAT MANTISSA.
29  0011 05               LSLD                            SHIFT TO CORRECT REGISTER POSITION.
30  0012 976C             STAA    PWDTY1                  OUTPUT TO PWM DUTY REGISTER.
31  0014 39               RTS
32  0015                  END
```

APPENDIX B
ASSEMBLY LANGUAGE PID ROUTINE

B.1 Master C Routine

```
/* This code just sets up variables and does some updates
   after the assembly PID loop is called. */

#include <stdio.h>
#include <io6811N4.h>
#include <int6811N.h>


zpage   unsigned int         TOFCOUNT;            /* declare variables */
zpage   signed int           CMNDVX;
zpage   signed int           ADRCX;
zpage   signed int           ADRCXM1;
zpage   signed int           ERRX;
zpage   signed int           ERRM1X;
zpage   signed int           ERRM2X;
zpage   signed int           ERRM3X;
zpage   signed int           KPNUM;
zpage   signed int           KPDEN;
zpage   signed int           KINUM;
zpage   signed int           KIDEN;
zpage   signed int           KDNUM;
zpage   signed int           KDDEN;
zpage   signed int           PERDTNUM;
zpage   signed int           PERDTDEN;
zpage   signed int           INT56;
zpage   signed int           FC56;
zpage   signed int           TEMP1;
zpage   signed int           TEMP2;
zpage   signed int           TEMP3;
zpage   signed int           TEMP4;
zpage   long                 NEWDTY;
zpage   long                 OLDDTY;
zpage   long                 KPTRM;
zpage   long                 KDTRM;
zpage   long                 KITRM;
zpage   long                 LTEMP1;
zpage   long                 LTEMP2;
zpage   long                 LTEMP3;
zpage   long                 LTEMP4;
zpage   long                 LTEMP5;
zpage   long                 LTEMP6;
zpage   long                 LTEMP7;
zpage   long                 LTEMP8;
zpage   long                 LTEMP9;
zpage   long                 LTEMPA;
zpage   long                 FCINT56;
zpage   long                 INTFC56;

extern int   DOPID (void);                  /* prototype for assembly
                                               routine */
```

```
void main()             /* main program */
{
CMNDVX = 0x0080;
PERDTNUM = 0x00A4;
PERDTDEN = 0x2710;      /* PERIOD = 164/10000 decimal */
KPNUM = 0x000C;
KPDEN = 0x0064;         /* kp = .12, ki = 6.0, kd = .006, for 1M ohm  drive */
KINUM = 0x0006;
KIDEN = 0x0001;
KDNUM = 0x0006;
KDDEN = 0x03E8;
OLDDTY =0x00FF0000;
PORTA = 0x00;
DDRA =  0xFF;           /* set PORTA as output */
PACTL = 0x03;           /* set RTI to 16.383 ms (E = 4MHz) */
TMSK2 = 0x40;           /* enable RTI interrupts */
OPTION = 0x90;          /* enable A/D charge pump */
PWPER1 = 0xFF;          /* set up PWM channel 1 at 15.625 kHz */
PWDTY1 = 0xFF;          /* with positive polarity */
PWPOL = 0x01;
DDRH = 0x00;
PWEN = 0x01;
TFLG2 = 0x40;

enable_interrupt();     /* wait for RTI to execute assembly PID routine */

wait_for_interrupt();

for (;;) {
  ; }
}

interrupt void IRQ_interrupt(void)     /* Just some traps for unexpected */
{                                      /* interrupts */

  PORTA = 0xFF;
  PORTA = 0x00;
}

interrupt void TO_interrupt(void)
{
  TOFCOUNT++ ;
}

interrupt void RTI_interrupt(void)      /* PWM routine */
{
  PORTA = 0xFF;                         /* scope strobe */
  DOPID();                              /*DO THE PID LOOP USING THE MATH
                                          COPROCESSOR */

/*  NEWDTY = KP*(ERRX) +  KI*PERDT*(CMNDVX - (ADRCX + ADRCXM1)/2)
           + (KD/(6*PERDT))*((ERRX - ERRM3X) + 3*(ERRM1X - ERRM2X))
           + OLDDTY; */


  TFLG2 = 0x40;                         /* clear RTI flag */
  ADRCXM1 = ADRCX;                      /* update A/D result for next cycle */
  ERRM3X = ERRM2X;                      /* update error pipeline */
  ERRM2X = ERRM1X;
  ERRM1X = ERRX;
  PORTA = 0x00;                         /* scope strobe */
}
```

B.2 DOPID Assembly Listing

```
 1                ************************************************************
 2                *    DOPID assembly function*
 3                *    These routines calculate the new PWM duty cycle*
 4                *    using the MC68HC11N4 math coprocessor.  The*
 5                *    code can be run on an M68HC11EVS with an*
 6                *    M68HC11K4 emulator and MC68HC11N4 processor.*
 7                *    The EVS monitor should be 1.1 or later. The EVS*
 8                *    and vectors were set to SPECIAL TEST MODE to aid debug.*
 9                *    This code is called by a C routine but could be converted*
10                *    to an all assembly environment by defining the variables*
11                *    in assembly instead of as externals.*
12                ************************************************************
13
14  0000                  MODULE  DOPID
15  0000                  PUBLIC  DOPID
16
17  0000                  P68H11
18  0000                  RSEG    CODE
19
20  006C            PWDTY1  set     $006c
21  0030            ADCTL   set     $0030
22  0031            ADR1    set     $0031
23  0040            CREGH   set     $0040
24  0041            CREGMH  set     $0041
25  0042            CREGML  set     $0042
26  0043            CREGL   set     $0043
27  0044            ALUC    set     $0044
28  0045            AREGH   set     $0045
29  0046            AREGL   set     $0046
30  0047            BREGH   set     $0047
31  0048            BREGL   set     $0048
32  0049            ALUF    set     $0049
33
34
35
36
37  0000                  EXTERN  ADRCX:ZPAGE           $0084  EXTERNAL VARIABLES
38  0000                  EXTERN  ADRCXM1:ZPAGE         $0086  SIGNED INTS
39  0000                  EXTERN  CMNDVX:ZPAGE          $0082
40  0000                  EXTERN  ERRX:ZPAGE            $0088
41  0000                  EXTERN  ERRM1X:ZPAGE          $008A
42  0000                  EXTERN  ERRM2X:ZPAGE          $008C
43  0000                  EXTERN  ERRM3X:ZPAGE          $008E
44  0000                  EXTERN  KPNUM:ZPAGE           $0090
45  0000                  EXTERN  KPDEN:ZPAGE           $0092
46  0000                  EXTERN  KINUM:ZPAGE           $0094
47  0000                  EXTERN  KIDEN:ZPAGE           $0096
48  0000                  EXTERN  KDNUM:ZPAGE           $0098
49  0000                  EXTERN  KDDEN:ZPAGE           $009A
50  0000                  EXTERN  PERDTNUM:ZPAGE        $009C
51  0000                  EXTERN  PERDTDEN:ZPAGE        $009E
52  0000                  EXTERN  INT56:ZPAGE           $00A0
53  0000                  EXTERN  FC56:ZPAGE            $00A2
54  0000                  EXTERN  TEMP1:ZPAGE           $00A4
55  0000                  EXTERN  TEMP2:ZPAGE           $00A6
56  0000                  EXTERN  TEMP3:ZPAGE           $00A8
57  0000                  EXTERN  TEMP4:ZPAGE           $00AA
58  0000                  EXTERN  KPTRM:ZPAGE           $00B4 LONGS
59  0000                  EXTERN  KITRM:ZPAGE           $00BC
60  0000                  EXTERN  KDTRM:ZPAGE           $00B8
61  0000                  EXTERN  LTEMP1:ZPAGE          $00C0
62  0000                  EXTERN  LTEMP2:ZPAGE          $00C4
63  0000                  EXTERN  LTEMP3:ZPAGE          $00C8
64  0000                  EXTERN  LTEMP4:ZPAGE          $00CC
65  0000                  EXTERN  LTEMP5:ZPAGE          $00D0
66  0000                  EXTERN  LTEMP6:ZPAGE          $00D4
```

```
 67  0000                    EXTERN  LTEMP7:ZPAGE          $00D8
 68  0000                    EXTERN  LTEMP8:ZPAGE          $00DC
 69  0000                    EXTERN  LTEMP9:ZPAGE          $00E0
 70  0000                    EXTERN  LTEMPA:ZPAGE          $00E4
 71  0000                    EXTERN  FCINT56:ZPAGE         $00E8
 72  0000                    EXTERN  INTFC56:ZPAGE         $00EC
 73  0000                    EXTERN  OLDDTY:ZPAGE          $00B0
 74  0000                    EXTERN  NEWDTY:ZPAGE          $00AC
 75  0000            DOPID:
 76                  ******** OUTPUT LAST PERIOD RESULT AND DO KP TERM ********
 77
 78  0000 9601               LDAA    NEWDTY+1              OUTPUT PREVIOUS CALC.
 79  0002 976C               STAA    PWDTY1
 80  0004 4F                 CLRA
 81  0005 D631               LDAB    ADR1                  GET CHANNEL 1 A/D RESULT.
 82  0007 DD00               STD     ADRCX                 DO KP TERM
 83  0009 C610               LDAB    #$10                  FIRST START NEW A/D
 84  000B D730               STAB    ADCTL                 CONVERSION
 85  000D DC00               LDD     CMNDVX
 86  000F 9300               SUBD    ADRCX                 FORM ERROR TERM
 87  0011 DD00               STD     ERRX
 88  0013 2B06               BMI     NFLAG1                SET UP SIGN FLAG IN TEMP3
 89  0015 8600               LDAA    #$00                  POS
 90  0017 9700               STAA    TEMP3
 91  0019 2004               BRA     NFLGG2
 92  001B 86FF       NFLAG1  LDAA    #$FF                  NEG
 93  001D 9700               STAA    TEMP3
 94  001F 8680       NFLGG2  LDAA    #$80                  SET ALU FOR SMUL
 95  0021 9744               STAA    ALUC
 96  0023 DC00               LDD     ERRX
 97  0025 DD45               STD     AREGH
 98  0027 8601               LDAA    #$01                  CLEAR ACF
 99  0029 9749               STAA    ALUF
100  002B DC00               LDD     KPNUM
101  002D DD47               STD     BREGH                 TRIGGER SMUL
102  002F 134901FC   WPMUL1  BRCLR   ALUF,#$01,WPMUL1      WAIT FOR ACF
103  0033 86C0               LDAA    #$D0                  SET ALU FOR SIDIV
104  0035 DD44               STD     ALUC
105  0037 8601               LDAA    #$01                  CLEAR ACF
106  0039 9749               STAA    ALUF
107  003B DC00               LDD     KPDEN
108  003D DD45               STD     AREGH                 TRIGGER SIDIV
109  003F 134901FC   WPDIV1  BRCLR   ALUF,#$01,WPDIV1      WAIT FOR ACF
110  0043 8601               LDAA    #$01                  CLEAR ACF
111  0045 9749               STAA    ALUF
112  0047 86E8               LDAA    #$E8                  TRIGGER SFDIV
113  0049 DD44               STD     ALUC
114  004B 134901FC   WPFDV1  BRCLR   ALUF,#$01,WPFDV1      WAIT FOR ACF
115  004F DC40               LDD     CREGH                 GET INT PART OF RESULT
116  0051 DD00               STD     KPTRM
117  0053 DD00               STD     LTEMP1
118  0055 DC42               LDD     CREGML                GET FRACTION
119  0057 DD02               STD     KPTRM+2
120  0059 DD02               STD     LTEMP1+2
121  005B BD0361             JSR     ADLNG                 NOW ADD TO OLDDTY
122  005E BD007F             JSR     DOKIT                 DO I TERM, ADD TO OLDDTY
123  0061 BD014F             JSR     DOKDT                 DO D TERM, ADD TO OLDDTY
124  0064 DC00               LDD     NEWDTY                CHECK LIMITS
125  0066 2B0F               BMI     JAMZP
126  0068 1A8300FF           CPD     #$00FF
127  006C 2B10               BMI     KXDONE
128  006E CC00FF             LDD     #$00FF                JAM FF
129  0071 DD00               STD     NEWDTY                SATURATED HIGH
130  0073 DD00               STD     OLDDTY
131  0075 2007               BRA     KXDONE
132  0077 CC0000     JAMZP   LDD     #$0000                JAM 00
133  007A DD00               STD     NEWDTY                SATURATED LOW
134  007C DD00               STD     OLDDTY
```

```
135  007E 39        KXDONE  RTS
136
137                 * ROUTINE TO DO INTEGRAL TERM *
138
139  007F DC00      DOKIT   LDD     ADRCX                   GET CURRENT CONVERSION
140  0081 D300              ADDD    ADRCXM1                 FORM (ADRCX + ADRCXM1)/2
141  0083 04                LSRD
142  0084 DD00              STD     LTEMP2
143  0086 2507              BCS     JMHAFI
144  0088 CC0000            LDD     #$0000
145  008B DD02            STD     LTEMP2+2              FRACTIONAL PART OF FINAL ERROR
146  008D 2005             BRA     INTKIE                TERM WILL ALWAYS BE 0 or 0.5
147  008F CC8000    JMHAFI  LDD     #$8000
148  0092 DD02              STD     LTEMP2+2
149  0094 DC00      INTKIE  LDD     CMNDVX
150  0096 9300              SUBD    LTEMP2
151  0098 1302800B          BRCLR   LTEMP2+2,#$80,NOFCN
152  009C DD00              STD     LTEMP2
153  009E 1A830000          CPD     #$0000
154  00A2 2F0B              BLE     NGFLG3
155  00A4 830001            SUBD    #$0001
156  00A7 DD00     NOFCN   STD     LTEMP2                CMNDVX - ((ADRCX + ADRCXM1)/2)
157  00A9 2B04              BMI     NGFLG3                    SET UP SIGN FLAG IN TEMP3
158  00AB 8600              LDAA    #$00
159  00AD 2002              BRA     NGFLG2
160  00AF 86FF      NGFLG3  LDAA    #$FF
161  00B1 9700      NGFLG2  STAA    TEMP3
162  00B3 86C0              LDAA    #$D0                   SET ALU FOR SIDIV TO FORM
163  00B5 DD44              STD     ALUC                    KINUM/KIDEN
164  00B7 8601              LDAA    #$01                    CLEAR ACF
165  00B9 9749              STAA    ALUF
166  00BB CC0000            LDD     #$0000                  SET UP KI NUMERATOR
167  00BE DD40              STD     CREGH
168  00C0 DC00              LDD     KINUM
169  00C2 DD42              STD     CREGML
170  00C4 DC00              LDD     KIDEN
171  00C6 DD45              STD     AREGH                   TRIGGER SIDIV
172  00C8 134901FC  WIDIV1  BRCLR   ALUF,#$01,WIDIV1       WAIT FOR ACF
173  00CC 8601              LDAA    #$01                    CLEAR ACF
174  00CE 9749              STAA    ALUF
175  00D0 86E8              LDAA    #$E8                    TRIGGER SFDIV
176  00D2 DD44              STD     ALUC
177  00D4 134901FC  WIFDV1  BRCLR   ALUF,#$01,WIFDV1       WAIT FOR ACF
178  00D8 DC40              LDD     CREGH
179  00DA DD00              STD     LTEMP3
180  00DC DC42              LDD     CREGML
181  00DE DD02              STD     LTEMP3+2
182  00E0 86C0              LDAA    #$D0                   SET ALU FOR SIDIV TO FORM
183  00E2 DD44              STD     ALUC                    PERDTNUM/PERDTDEN
184  00E4 8601              LDAA    #$01                    CLEAR ACF
185  00E6 9749              STAA    ALUF
186  00E8 CC0000            LDD     #$0000                  SET UP PERDTNUM
187  00EB DD40              STD     CREGH
188  00ED DC00              LDD     PERDTNUM
189  00EF DD42              STD     CREGML
190  00F1 DC00              LDD     PERDTDEN
191  00F3 DD45              STD     AREGH                   TRIGGER SIDIV
192  00F5 134901FC  WIDIV2  BRCLR   ALUF,#$01,WIDIV2       WAIT FOR ACF
193  00F9 8601              LDAA    #$01                    CLEAR ACF
194  00FB 9749              STAA    ALUF
195  00FD 86E8              LDAA    #$E8                    TRIGGER SFDIV
196  00FF DD44              STD     ALUC
197  0101 134901FC  WIFDV2  BRCLR   ALUF,#$01,WIFDV2       WAIT FOR ACF
198  0105 DC40              LDD     CREGH
199  0107 DD00              STD     LTEMP4
200  0109 DC42              LDD     CREGML
201  010B DD02              STD     LTEMP4+2
202  010D DC00            LDD     LTEMP3                NOW FORM LTEMP2*LTEMP3*LTEMP4
```

```
203  010F DD00              STD     LTEMP5
204  0111 DC02              LDD     LTEMP3+2
205  0113 DD02              STD     LTEMP5+2
206  0115 DC00              LDD     LTEMP4
207  0117 DD00              STD     LTEMP6
208  0119 DC02              LDD     LTEMP4+2
209  011B DD02              STD     LTEMP6+2
210  011D 9600              LDAA    TEMP3                   SAVE SIGN FLAG
211  011F 9700              STAA    TEMP4                   AND USE TEMP3 AS A FLAG
212  0121 8600              LDAA    #$00                   FOR LTEMP6 BEING POSITIVE
213  0123 9700              STAA    TEMP3
214  0125 BD023A            JSR     MULLNG                DO LTEMP3*LTEMP4(PERDT*KI)
215  0128 DC00              LDD     LTEMP7                  NOW PUT RESULT IN LTEMP5
216  012A DD00              STD     LTEMP5
217  012C DC02              LDD     LTEMP7+2
218  012E DD02              STD     LTEMP5+2
219  0130 9600              LDAA    TEMP4                   RETRIEVE SIGN FLAG
220  0132 9700              STAA    TEMP3
221  0134 DC00              LDD     LTEMP2
222  0136 DD00              STD     LTEMP6                  ERROR FOR KI TERM
223  0138 DC02              LDD     LTEMP2+2
224  013A DD02              STD     LTEMP6+2
225  013C BD023A            JSR     MULLNG                  DO RESULT*LTEMP2
226  013F DC00              LDD     LTEMP7
227  0141 DD00              STD     LTEMP1
228  0143 DD00              STD     KITRM
229  0145 DC02              LDD     LTEMP7+2
230  0147 DD02              STD     LTEMP1+2
231  0149 DD02              STD     KITRM+2                 ADD KI TERM INTO NEWDTY
232  014B BD0361            JSR     ADLNG                   KITERM DONE
233  014E 39                RTS                             RETURN
234
235                 *********    ROUTINE TO DO KD TERM     **********
236
237  014F DC00      DOKDT   LDD     ERRX                    FORM (ERRX - ERRM3X)
238  0151 9300              SUBD    ERRM3X                      + 3*(ERRM1X - ERRM2X)
239  0153 DD00              STD     TEMP1
240  0155 DC00              LDD     ERRM1X
241  0157 9300              SUBD    ERRM2X
242  0159 DD45              STD     AREGH                   FORM 3*(ERRM1X - ERRM2X)
243  015B 8680              LDAA    #$80
244  015D 9744              STAA    ALUC
245  015F 8601              LDAA    #$01
246  0161 9749              STAA    ALUF
247  0163 CC0003            LDD     #$0003
248  0166 DD47              STD     BREGH
249  0168 134901FC  WDMUL0  BRCLR   ALUF,#$01,WDMUL0        WAIT FOR ACF
250  016C DC42              LDD     CREGML
251  016E D300              ADDD    TEMP1
252  0170 DD00              STD     LTEMPA
253  0172 2B02              BMI     NGFLGS0                SET UP SIGN FLAG IN TEMP3
254  0174 2006              BRA     POSGN
255  0176 86FF      NGFLGS0 LDAA    #$FF
256  0178 9700              STAA    TEMP3
257  017A 2004              BRA     KDFLGD
258  017C 8600      POSGN   LDAA    #$00
259  017E 9700              STAA    TEMP3
260  0180 CC0000    KDFLGD  LDD     #$0000
261  0183 DD02              STD     LTEMPA+2                DONE
262  0185 8600              LDAA    #$00                    FORM 6*PERDTNUM
263  0187 9744              STAA    ALUC
264  0189 8601              LDAA    #$01
265  018B 9749              STAA    ALUF
266  018D CC0006            LDD     #$0006
267  0190 DD45              STD     AREGH
268  0192 DC00              LDD     PERDTNUM
269  0194 DD47              STD     BREGH
270  0196 134901FC  WDMUL1  BRCLR   ALUF,#$01,WDMUL1        WAIT FOR ACF
```

```
271  019A DC42              LDD     CREGML
272  019C DD00              STD     TEMP2
273  019E 86C0      NOFCND  LDAA    #$D0                    SET ALU FOR SIDIV TO FORM
274  01A0 DD44              STD     ALUC                    KDNUM/KDDEN
275  01A2 8601              LDAA    #$01                    CLEAR ACF
276  01A4 9749              STAA    ALUF
277  01A6 CC0000            LDD     #$0000                  SET UP KD NUMERATOR
278  01A9 DD40              STD     CREGH
279  01AB DC00              LDD     KDNUM
280  01AD DD42              STD     CREGML
281  01AF DC00              LDD     KDDEN
282  01B1 DD45              STD     AREGH                   TRIGGER SIDIV
283  01B3 134901FC  WDDIV1  BRCLR   ALUF,#$01,WDDIV1        WAIT FOR ACF
284  01B7 8601              LDAA    #$01                    CLEAR ACF
285  01B9 9749              STAA    ALUF
286  01BB 86E8              LDAA    #$E8                    TRIGGER SFDIV
287  01BD DD44              STD     ALUC
288  01BF 134901FC  WDFDV1  BRCLR   ALUF,#$01,WDFDV1        WAIT FOR ACF
289  01C3 DC40              LDD     CREGH
290  01C5 DD00              STD     LTEMP8
291  01C7 DC42              LDD     CREGML
292  01C9 DD02              STD     LTEMP8+2
293  01CB 86C0              LDAA    #$D0                    SET ALU FOR SIDIV TO FORM
294  01CD DD44              STD     ALUC                    PERDTDEN/(PERDTNUM*6)
295  01CF 8601              LDAA    #$01                    CLEAR ACF
296  01D1 9749              STAA    ALUF
297  01D3 CC0000            LDD     #$0000                  SET UP PERDTNUM
298  01D6 DD40              STD     CREGH
299  01D8 DC00              LDD     PERDTDEN
300  01DA DD42              STD     CREGML
301  01DC DC00              LDD     TEMP2
302  01DE DD45              STD     AREGH                   TRIGGER SIDIV
303  01E0 134901FC  WDDIV2  BRCLR   ALUF,#$01,WDDIV2        WAIT FOR ACF
304  01E4 8601              LDAA    #$01                    CLEAR ACF
305  01E6 9749              STAA    ALUF
306  01E8 86E8              LDAA    #$E8                    TRIGGER SFDIV
307  01EA DD44              STD     ALUC
308  01EC 134901FC  WDFDV2  BRCLR   ALUF,#$01,WDFDV2        WAIT FOR ACF
309  01F0 DC40              LDD     CREGH
310  01F2 DD00              STD     LTEMP9
311  01F4 DC42              LDD     CREGML
312  01F6 DD02              STD     LTEMP9+2
313  01F8 DC00            LDD     LTEMP8                  NOW FORM LTEMPA*LTEMP8*LTEMP9
314  01FA DD00              STD     LTEMP5
315  01FC DC02              LDD     LTEMP8+2
316  01FE DD02              STD     LTEMP5+2
317  0200 DC00              LDD     LTEMP9
318  0202 DD00              STD     LTEMP6
319  0204 DC02              LDD     LTEMP9+2
320  0206 DD02              STD     LTEMP6+2
321  0208 9600              LDAA    TEMP3                   SAVE SIGN FLAG
322  020A 9700              STAA    TEMP4                   AND USE TEMP3 AS A FLAG
323  020C 8600              LDAA    #$00                    FOR LTEMP6 BEING POSITIVE
324  020E 9700              STAA    TEMP3
325  0210 BD023A            JSR     MULLNG                  DO LTEMP8*LTEMP9
326  0213 DC00              LDD     LTEMP7                  NOW PUT RESULT IN LTEMP5
327  0215 DD00              STD     LTEMP5
328  0217 DC02              LDD     LTEMP7+2
329  0219 DD02              STD     LTEMP5+2
330  021B 9600              LDAA    TEMP4                   RETRIEVE SIGNED ERROR
331  021D 9700              STAA    TEMP3
332  021F DC00              LDD     LTEMPA
333  0221 DD00              STD     LTEMP6                  ERROR FOR KD TERM
334  0223 DC02              LDD     LTEMPA+2
335  0225 DD02              STD     LTEMP6+2
336  0227 BD023A            JSR     MULLNG                  DO RESULT*LTEMPA
337  022A DC00              LDD     LTEMP7
338  022C DD00              STD     LTEMP1
```

```
339  022E DD00              STD     KDTRM
340  0230 DC02              LDD     LTEMP7+2
341  0232 DD02              STD     LTEMP1+2
342  0234 DD02              STD     KDTRM+2
343  0236 BD0361            JSR     ADLNG                   ADD KD TERM INTO NEWDTY
344  0239 39                RTS                             KDTERM DONE
345
346                 *  SUBROUTINE TO MULTIPLY LONGS(INTEGER & FRACTION)  *
347                 *  LTEMP5*LTEMP6=LTEMP7 ONLY LTEMP6 CAN HAVE         *
348                 *  A NEGATIVE TERM TO HANDLE.                        *
349
350  023A 8680      MULLNG  LDAA    #$80                    SET ALU FOR SMUL
351  023C 9744              STAA    ALUC                    AND MULTIPLY INTS
352  023E DC00              LDD     LTEMP5
353  0240 DD45              STD     AREGH
354  0242 8601              LDAA    #$01                    CLEAR ACF
355  0244 9749              STAA    ALUF
356  0246 DC00              LDD     LTEMP6
357  0248 DD47              STD     BREGH                   TRIGGER SMUL
358  024A 134901FC  WMULL1  BRCLR   ALUF,#$01,WMULL1        WAIT FOR ACF
359  024E DC42              LDD     CREGML
360  0250 DD00              STD     INT56
361  0252 8601              LDAA    #$01                    CLEAR ACF AND DO NEXT MULT
362  0254 9749              STAA    ALUF
363  0256 8680              LDAA    #$80                    TEST TEMP3 SIGN
364  0258 9500              BITA    TEMP3                   SEE IF ERR IS NEG
365  025A 2B12              BMI     NEGFRAC                 TERM IS NEGATIVE
366  025C DC02              LDD     LTEMP6+2                GET FRAC NOT NEG
367  025E DD47              STD     BREGH                   TRIGGER SMUL
368  0260 134901FC  WMULL2  BRCLR   ALUF,#$01,WMULL2        WAIT FOR ACF
369  0264 DC40              LDD     CREGH                   SCALE AND STORE
370  0266 DD00              STD     INTFC56
371  0268 DC42              LDD     CREGML
372  026A DD02              STD     INTFC56+2
373  026C 2022              BRA     NXFRAC
374  026E CC0000    NEGFRAC LDD     #$0000                  NEGATE FRAC
375  0271 9302              SUBD    LTEMP6+2
376  0273 DD47              STD     BREGH                   TRIGGER SMUL
377  0275 134901FC  WMULL3  BRCLR   ALUF,#$01,WMULL3        WAIT FOR ACF
378  0279 CC0000            LDD     #$0000                  NEGATE RESULT
379  027C 9340              SUBD    CREGH                   SCALE AND STORE
380  027E 2B02              BMI     INTFIX1
381  0280 2003              BRA     INTFIX2
382  0282 C30001    INTFIX1 ADDD    #$0001
383  0285 DD00      INTFIX2 STD     INTFC56
384  0287 CC0000            LDD     #$0000
385  028A 9342              SUBD    CREGML
386  028C DD02              STD     INTFC56+2
387  028E 8600              LDAA    #$00
388  0290 DC02      NXFRAC  LDD     LTEMP5+2                GET FRAC AND MULTIPLY
389  0292 DD45              STD     AREGH                   WITH POSSIBLE NEG INT
390  0294 8601              LDAA    #$01                    CLEAR ACF
391  0296 9749              STAA    ALUF
392  0298 8600              LDAA    #$00
393  029A DC00              LDD     LTEMP6
394  029C DD47              STD     BREGH                   TRIGGER SMUL
395  029E 134901FC  WMULL4  BRCLR   ALUF,#$01,WMULL4        WAIT FOR ACF
396  02A2 8680              LDAA    #$80
397  02A4 9500              BITA    TEMP3
398  02A6 2A0F              BPL     DFXINT
399  02A8 8680              LDAA    #$80                    SEE IF SIGN OVERFLOW
400  02AA 9545              BITA    AREGH                   ON FRACTION
401  02AC 2B02              BMI     FXINT
402  02AE 2007              BRA     DFXINT
403  02B0 CC0000    FXINT   LDD     #$0000
404  02B3 9340              SUBD    CREGH
405  02B5 DD40              STD     CREGH
406  02B7 DC40      DFXINT  LDD     CREGH                   THIS SCALES FRAC
```

```
 407  02B9 2B02             BMI     NFFIX
 408  02BB 2003             BRA     PFFIX
 409  02BD C30001    NFFIX  ADDD    #$0001
 410  02C0 DD00      PFFIX  STD     FCINT56
 411  02C2 DC42             LDD     CREGML
 412  02C4 DD02             STD     FCINT56+2
 413  02C6 8601             LDAA    #$01                    NOW DO FRAC*FRAC
 414  02C8 9749             STAA    ALUF                    CLEAR ACF
 415  02CA 8600             LDAA    #$00
 416  02CC 9744            STAA    ALUC                   SET UNSIGNED MULT FOR FRACS
 417  02CE 8680             LDAA    #$80                    TEST ERR SIGN
 418  02D0 9500             BITA    TEMP3                   SEE IF ERR IS NEG
 419  02D2 2B0E             BMI     NFCFRAC                 TERM IS NEGATIVE
 420  02D4 DC02             LDD     LTEMP6+2                GET FRAC NOT NEG
 421  02D6 DD47             STD     BREGH                   TRIGGER SMUL
 422  02D8 134901FC  WMULL5 BRCLR   ALUF,#$01,WMULL5        WAIT FOR ACF
 423  02DC DC40             LDD     CREGH                   SCALE AND STORE
 424  02DE DD00             STD     FC56
 425  02E0 2012             BRA     SUMMUL
 426  02E2 CC0000    NFCFRAC LDD    #$0000                  NEGATE FRAC
 427  02E5 9302             SUBD    LTEMP6+2
 428  02E7 DD47             STD     BREGH                   TRIGGER SMUL
 429  02E9 134901FC  WMULL6 BRCLR   ALUF,#$01,WMULL6        WAIT FOR ACF
 430  02ED CC0000           LDD     #$0000                  NEGATE RESULT
 431  02F0 9340             SUBD    CREGH                   SCALE AND STORE
 432  02F2 DD00             STD     FC56
 433  02F4 DC00      SUMMUL LDD     INT56                   NOW SUMM ALL PRODUCTS
 434  02F6 D300             ADDD    FCINT56                 INTS ARE ALL SIGNED
 435  02F8 D300             ADDD    INTFC56                 CAN JUST ADD UP
 436  02FA DD00             STD     LTEMP7
 437  02FC 8680             LDAA    #$80                    TEST ERRX SIGN
 438  02FE 9500             BITA    TEMP3                   SEE IF FRACS ARE NEG
 439  0300 2B22             BMI     SUMNFC                  FRACS ARE NEGATIVE
 440  0302 DC02             LDD     FCINT56+2               POSITIVE
 441  0304 D302             ADDD    INTFC56+2
 442  0306 2502             BCS     FCCAR1
 443  0308 2009             BRA     SUMFC1
 444  030A 8F        FCCAR1 XGDX                            SAVE SUM
 445  030B CC0001           LDD     #$0001                  ADD CARRY INTO INT
 446  030E D300             ADDD    LTEMP7
 447  0310 DD00             STD     LTEMP7
 448  0312 8F               XGDX                           RETRIEVE SUM
 449  0313 D300      SUMFC1 ADDD    FC56
 450  0315 DD02             STD     LTEMP7+2
 451  0317 2502             BCS     FCCAR2
 452  0319 2045             BRA     SMFCDP
 453  031B CC0001    FCCAR2 LDD     #$0001                  ADD CARRY INTO INT
 454  031E D300             ADDD    LTEMP7
 455  0320 DD00             STD     LTEMP7
 456  0322 203C             BRA     SMFCDP
 457  0324 CC0000    SUMNFC LDD     #$0000                  COMPLEMENT NEG FRACS
 458  0327 9302             SUBD    FCINT56+2
 459  0329 DD02             STD     FCINT56+2
 460  032B CC0000           LDD     #$0000
 461  032E 9302             SUBD    INTFC56+2
 462  0330 DD02             STD     INTFC56+2
 463  0332 CC0000           LDD     #$0000
 464  0335 9300             SUBD    FC56
 465  0337 DD00             STD     FC56
 466  0339 DC02             LDD     FCINT56+2               NEGATIVE
 467  033B D302             ADDD    INTFC56+2
 468  033D 2502             BCS     FCCAR3                  REMEMBER SIGN BIT!!
 469  033F 2009             BRA     SUMFC2
 470  0341 8F        FCCAR3 XGDX                            SAVE SUM
 471  0342 CCFFFF           LDD     #$FFFF                  ADD BORROW INTO INT
 472  0345 D300             ADDD    LTEMP7
 473  0347 DD00             STD     LTEMP7
 474  0349 8F               XGDX                           RETRIEVE SUM
```

```
475  034A D300      SUMFC2  ADDD    FC56
476  034C DD02              STD     LTEMP7+2
477  034E 2502              BCS     FCCAR4
478  0350 2007              BRA     SMFCDN
479  0352 CCFFFF    FCCAR4  LDD     #$FFFF                  ADD BORROW INTO INT
480  0355 D300              ADDD    LTEMP7
481  0357 DD00              STD     LTEMP7
482  0359 CC0000    SMFCDN  LDD     #$0000                  CONVERT BACK TO NEG
483  035C 9302              SUBD    LTEMP7+2
484  035E DD02              STD     LTEMP7+2
485  0360 39        SMFCDP  RTS
486
487
488                 * SUBROUTINE TO ADD INTEGER AND FRACTION IN LTEMP1 TO OLDDTY *
489
490  0361 8680      ADLNG   LDAA    #$80                    TEST ERRX SIGN
491  0363 9500              BITA    TEMP3
492  0365 2B19              BMI     KXNEG                   TERM IS NEGATIVE
493  0367 DC00              LDD     LTEMP1                  GET INT PART
494  0369 D300              ADDD    OLDDTY                  ADD AND STORE INT
495  036B DD00              STD     NEWDTY
496  036D DC02              LDD     LTEMP1+2                GET FRAC PART
497  036F D302              ADDD    OLDDTY+2                ADD AND STORE FRAC
498  0371 DD02              STD     NEWDTY+2
499  0373 2502              BCS     INCINT
500  0375 201E              BRA     ADDONE
501  0377 CC0001    INCINT  LDD     #$0001                  ADD CARRY FROM FRAC
502  037A D300              ADDD    NEWDTY
503  037C DD00              STD     NEWDTY
504  037E 2015              BRA     ADDONE
505  0380 DC00      KXNEG   LDD     LTEMP1                  GET INT PART
506  0382 D300              ADDD    OLDDTY                  ADD AND STORE INT
507  0384 DD00              STD     NEWDTY
508  0386 DC02              LDD     LTEMP1+2                GET FRAC PART
509  0388 D302              ADDD    OLDDTY+2                ADD AND STORE FRAC
510  038A DD02              STD     NEWDTY+2                ACTUALLY A SUBTRACTION
511  038C 2507              BCS     ADDONE
512  038E CCFFFF    DECINT  LDD     #$FFFF                  SUBTRACT BORROW FROM FRAC
513  0391 D300              ADDD    NEWDTY
514  0393 DD00              STD     NEWDTY
515  0395 DC00      ADDONE  LDD     NEWDTY                  UPDATE OLDDTY FOR NEXT TERM
516  0397 DD00              STD     OLDDTY                    OR FINISH
517  0399 DC02              LDD     NEWDTY+2
518  039B DD02              STD     OLDDTY+2
519  039D 39                RTS                             RETURN TO CALLING ROUTINE
520  039E                   END

  Errors:  None          #########
  Bytes:   926           # DOPID #
  CRC:     EC21          #########
```

TO OBTAIN ADDITIONAL PRODUCT INFORMATION:

USA/EUROPE: Motorola Literature Distribution;
P.O. Box 20912; Phoenix, Arizona 85036. 1-800-441-2447
JAPAN: Nippon Motorola Ltd.; Tatsumi-SPD-JLDC, Toshikatsu Otsuki,
6F Seibu-Butsuryu-Center, 3-14-2 Tatsumi Koto-Ku, Tokyo 135, Japan. 03-3521-8315
HONG KONG: Motorola Semiconductors H.K. Ltd.; 8B Tai Ping Industrial Park,
51 Ting Kok Road, Tai Po, N.T., Hong Kong. 852-26629298
MFAX: RMFAX0@email.sps.mot.com - TOUCHTONE (602) 244-6609
INTERNET: http://www.mot.com

MOTOROLA