INTRODUCTION

One very common and relatively quick method for finding the square root of a number is the Newton-Raphson method. Although this method is quick in terms of mathematics, it also requires extensive use of division to produce results, usually iterating many times. In the PIC18CXX2 microcontroller family, though not difficult, division does require several basic operations. However, with the help of the single cycle hardware multiplier, one of the many nice features in the PIC18CXX2 and the use of a technique different from the Newton-Raphson method, division is avoided. The following algorithm demonstrates how the single cycle multiplier is useful in calculating a square root and at the same time, save processor time.

THE ALGORITHM

Using the binary nature of the microcontroller, the square root of a fixed precision number can be found quickly. Each digit in a binary number represents a power of two. By successively rotating through each bit, or power of two and testing the result against the desired value, i.e. squaring the guess and checking if it is greater than the original argument, the approximate root gets closer and closer to the actual value. In conjunction with this, the value is achieved quickly. This is because each bit is tested rather than every possible 8-bit combination. The general technique is outlined in Figure 1. For a 16-bit integer, only nine program loops are required to completely test and produce a result. Example 1 is a demonstration of this procedure:

FIGURE 1: SQUARE ROOT FLOW CHART
EXAMPLE 1: 8-BIT EXAMPLE

\[ A = \sqrt{0xCF48} \]

or

\[ A^2 = 0xCF48 \]

<table>
<thead>
<tr>
<th>Step</th>
<th>A</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1000 0000 (0x80)</td>
<td>this squared is less than 0xCF48, start next cycle with a new bit</td>
</tr>
<tr>
<td></td>
<td><strong>new bit</strong></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1100 0000 (0xC0)</td>
<td>this squared is less than 0xCF48, start next cycle with a new bit</td>
</tr>
<tr>
<td></td>
<td><strong>new bit</strong></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1110 0000 (0xE0)</td>
<td>this squared is less than 0xCF48, start next cycle with a new bit</td>
</tr>
<tr>
<td></td>
<td><strong>new bit</strong></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1111 0000 (0xF0)</td>
<td>this is greater than 0xCF48, shift bit right</td>
</tr>
<tr>
<td></td>
<td><strong>shifted bit</strong></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1110 1000 (0xE8)</td>
<td>this is greater than 0xCF48, shift bit right</td>
</tr>
<tr>
<td></td>
<td><strong>shifted bit</strong></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1110 0100 (0xE4)</td>
<td>this squared is less than 0xCF48, start next cycle with a new bit</td>
</tr>
<tr>
<td></td>
<td><strong>new bit</strong></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1110 0110 (0xE6)</td>
<td>this squared is less than 0xCF48, start next cycle with a new bit</td>
</tr>
<tr>
<td></td>
<td><strong>new bit</strong></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1110 0111 (0xE7)</td>
<td>this is greater than 0xCF48, shift right</td>
</tr>
<tr>
<td></td>
<td><strong>bit shifted out</strong></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1110 0110 (0xE6)</td>
<td>right-most bit is thrown away for the integer approximation and the process is finished; otherwise, this could keep going for more accurate fractional approximation</td>
</tr>
</tbody>
</table>
ANALYSIS

Following the flow of this algorithm, there are only nine loops for an 8-bit number. And summing all the mathematics involved, there is only one multiplication and one conditional test for each step; a conditional test is most likely a subtraction with some bit testing. Plus, there are some logical operations to perform the bit manipulations, again one per loop. This means there are three basic operations per loop, totaling to 27 operations for the complete routine. Of course, the actual number of operations goes up some when applied to a specific microcontroller, but subjectively speaking, this is still not bad when compared to the large number of steps required to perform any number of divisions as required by the Newton-Raphson method.

The program in Appendix A is a functioning demonstration of the algorithm described above for 16-bit and 32-bit numbers. Table 1 gives the simulation results for these code examples. Also, the code is written specifically for the PIC18CXX2 series microcontrollers, but it can be modified to run on PIC17C microcontrollers that have a hardware multiplier.

CONCLUSION

This algorithm is just one possible way to compute the square root of a number. Its advantage is in the use of multiplication, a function easily performed on the PIC18CXX2 microcontroller, rather than division, an operation requiring a number of basic operations. In addition, the method and coding are extremely simple, requiring very little program and data memory. The end result is a fast and compact method to calculate the integer square root of a number.

MEMORY REQUIREMENTS

<table>
<thead>
<tr>
<th>Section Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>Section</td>
</tr>
<tr>
<td>--------------</td>
</tr>
<tr>
<td>R_Vctr</td>
</tr>
<tr>
<td>.cinit</td>
</tr>
<tr>
<td>S_ROOT</td>
</tr>
<tr>
<td>SRoot</td>
</tr>
<tr>
<td>SimpMth</td>
</tr>
</tbody>
</table>

Program Memory Usage

<table>
<thead>
<tr>
<th>Start</th>
<th>End</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x000000</td>
<td>0x000003</td>
</tr>
<tr>
<td>0x00002a</td>
<td>0x000141</td>
</tr>
</tbody>
</table>

284 out of 32768 program addresses used, program memory utilization is 0%
APPENDIX A: MAIN.ASM

; *******************************************************************
; Title "Square Root Calling Routine Demo"
; *******************************************************************

; *******************************************************************
; *** Author: Ross Fosler ***
; *** Applications Engineer ***
; *** Microchip Technology Inc. ***
; *** Program:main.asm ***
; *** This routine calls the square root function ***
; *** to find the root of two arbitrary numbers. ***
; *** Last Rev:August 10, 2000 ***
; *** Ver 1.00 ***
; ***
; *******************************************************************

listp=18C252
#include P18C252.INC

EXTERN ARGA0, ARGA1, ARGA2, ARGA3
EXTERN RES0, RES1
EXTERN Sqrt

W equ 0 ; Standard constants
F equ 1
a equ 0

R_Vctr CODE 0x0000
    goto Main
; *******************************************************************

Software License Agreement
The software supplied herewith by Microchip Technology Incorporated (the "Company") for its PICmicro® Microcontroller is intended and supplied to you, the Company’s customer, for use solely and exclusively on Microchip PICmicro Microcontroller products.

The software is owned by the Company and/or its supplier, and is protected under applicable copyright laws. All rights are reserved. Any use in violation of the foregoing restrictions may subject the user to criminal sanctions under applicable laws, as well as to civil liability for the breach of the terms and conditions of this license.

THIS SOFTWARE IS PROVIDED IN AN "AS IS" CONDITION. NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
; Calling Routine
SRoot CODE

Main

movlw 0xCF
movwf ARGA0, a
movlw 0x48
movwf ARGA1, a

call Sqrt ; Sqrt(0xCF48)

movlw 0xE0
movwf ARGA3, a
movlw 0x12
movwf ARGA2, a
movlw 0xA1
movwf ARGA1, a
movlw 0x40
movwf ARGA0, a

call Sqrt ; Sqrt(0xE012A140)

bra Main ; RES1:RES0 should now contain 0xEF81

; **************************************************

END
APPENDIX B: SQRT.ASM

; *******************************************************************
; Title:"16/32 bit Integer Square Root"
; *******************************************************************

; *******************************************************************
; *** ***
; *** Author: Ross Fosler ***
; *** Applications Engineer ***
; *** Microchip Technology Inc. ***
; *** ***
; *** Program:sqrt.asm ***
; *** This module contains code to perform fast integer ***
; *** square root functions on either 16 or 32 bit ***
; *** values. ***
; *** ***
; *** Last Rev:August 10, 2000 ***
; *** Ver 1.00 ***
; *** ***
; *******************************************************************

#include P18C252.INC

# include P18C252.INC

; *******************************************************************
; *******************************************************************

MSB equ 7 ; general literal constants
LSB equ 0
W equ 0
F equ 1
a equ 0

SimpMth UDATA_ACS

ARGA0 res 1 ; various argument registers
ARGA1 res 1
ARGA2 res 1
ARGA3 res 1

GLOBAL ARGA0, ARGA1, ARGA2, ARGA3

ARG1H res 1
ARG1L res 1
ARG2H res 1
ARG2L res 1

GLOBAL ARG1H, ARG1L, ARG2H, ARG2L

SARG1 res 1 ; signed arguments
SARG2 res 1

GLOBAL SARG1, SARG2

RES1 res 1 ; result registers
RES0 res 1

GLOBAL RES0, RES1

SQRES0 res 1
SQRES1 res 1
SQRES2 res 1
SQRES3  res  1

GLOBAL  SQRES0, SQRES1, SQRES2, SQRES3

BITLOC0  res  1 ; temporary registers
BITLOC1  res  1
TEMP0  res  1
TEMP1  res  1

; *******************************************************************
; *******************************************************************
; The function of this square root routine is to determine the root
; to the nearest integer. At the same time the root is found at the
; best possible speed; therefore, the root is found a little differently
; for the two basic sizes of numbers, 16-bit and 32-bit. The following
; differentiates the two and jumps to the appropriate function.

; Sqrt(ARGA3:ARGA2:ARGA1:ARGA0) = RES1:RES0

S_ROOT  CODE

Sqrt  tstfsz  ARGA3, a ; determine if the number is 16-bit
bra  Sqrt32 ; or 32-bit and call the best function

tstfsz  ARGA2, a
bra  Sqrt32

clf  RES1, a
bra  Sqrt16

GLOBAL  Sqrt

; ******************** Square Root ****************************
;  Sqrt16(ARGA1:ARGA0) = RES0

Sqrt16  clrf  TEMP0, a ; clear the temp solution
movlw  0x80 ; setup the first bit
movwf  BITLOC0, a
movwf  RES0, a

Square8  movf  RES0, W, a ; square the guess
mulwf  RES0, a

movf  PRODL, W, a ; ARGA - PROD test
subwf  ARGA0, W, a
movf  PRODH, W, a
subwf  ARGA1, W, a

btfsc  STATUS, C, a
bra  NextBit ; if positive then next bit
; if negative then rotate right

movff  TEMP0, RES0 ; move last good value back into RES0
rrncf  BITLOC0, F, a ; then rotate the bit and put it
movf  BITLOC0, W, a ; back into RES0
iorwf  RES0, F, a

btfsc  BITLOC0, 7, a ; if last value was tested then get
bra  Done ; out

bra  Square8 ; also go back for another test

NextBit  movf  RES0, TEMP0 ; copy the last good approximation
rrncf  BITLOC0, F, a ; rotate the bit location register
movf  BITLOC0, W, a
iorwf  RES0, F, a
btfsc BITLOC0, 7, a ; if last value was tested then get
bra Done ; out
bra Square8

Done movff TEMP0,RES0 ; put the final result in RES0
return

GLOBAL Sqrt16

; *******************************************************************
; ******************** Square Root **********************************
; Sqrt32(ARGA3:ARGA2:ARGA1:ARGA0) = RES1:RES0

Sqrt32 clrf TEMP0, a ; clear the temp solution
clrft TEMP1, a
clrft BITLOC0, a ; setup the first bit
clrft RES0, a
movlw 0x80
movwf BITLOC1, a ; BitLoc = 0x8000
movwf RES1, a ; RES = 0x8000

Squar16 movff RES0, ARG1L ; square the guess
movff RES1, ARG1H
call Sql16

movf SQRES0, W, a ; ARGA - PROD test
subwf ARGA0, W, a
movf SQRES1, W, a
subwfb ARGA1, W, a
movf SQRES2, W, a
subwfb ARGA2, W, a
movf SQRES3, W, a
subwfb ARGA3, W, a

btfsc STATUS, C, a ; if positive then next bit
bra NxtBt16 ; if negative then rotate right
addlw 0x00 ; clear carry
movff TEMP0, RES0 ; move last good value back into RES0
movff TEMP1, RES1

rrcf BITLOC1, F, a ; then rotate the bit and put it
rrcf BITLOC0, F, a
movf BITLOC1, W, a ; back into RES1:RES0
iorwf RES1, F, a
movf BITLOC0, W, a
iorwf RES0, F, a

btfsc STATUS, C, a ; if last value was tested then get
bra Done32 ; out
bra Squar16 ; else go back for another test

NxtBt16 addlw 0x00 ; clear carry
movff RES0, TEMP0 ; copy the last good approximation
movff RES1, TEMP1

rrcf BITLOC1, F, a ; rotate the bit location register
rrcf BITLOC0, F, a
movf BITLOC1, W, a ; and put back into RES1:RES0
iorwf RES1, F, a
movf BITLOC0, W, a
iorwf RES0, F, a
btfsc STATUS, C, a ; if last value was tested then get
bra Done32 ; out
bra Squar16

Done32 movff TEMPO,RES0 ; put the final result in RES1:RES0
movff TEMP1,RES1
return

GLOBAL Sqrt32

; **************************************************************************
; *********** 16 X 16 Unsigned Square **************************************
; SQRRES3:SQRRES0 = ARG1H:ARG1L ^2
;

Sq16 movf ARG1L, W, a
mulwf ARG1L ; ARG1L * ARG2L -> PRODH:PRODL
movff PRODH, SQRRES1 ;
movff PRODL, SQRRES0 ;

movf ARG1H, W, a
mulwf ARG1H ; ARG1H * ARG2H -> PRODH:PRODL
movff PRODH, SQRRES3 ;
movff PRODL, SQRRES2 ;

movf ARG1L, W, a
mulwf ARG1H ; ARG1L * ARG2H -> PRODH:PRODL
movf ARG1L, W, a ; ARG1L * ARG2L -> PRODH:PRODL
addwf SQRRES1, F, a ; Add cross
movf PRODH, W, a ; products
addwfc SQRRES2, F, a ;
clf WREG, a ;
addwfc SQRRES3, F, a ;

movf ARG1H, W, a ;
mulwf ARG1L ; ARG1H * ARG2L -> PRODH:PRODL
movf PRODL, W, a ;
addwf SQRRES1, F, a ; Add cross
movf PRODH, W, a ; products
addwfc SQRRES2, F, a ;
clf WREG, W ;
addwfc SQRRES3, F, a ;

return

GLOBAL Sq16

; ***************************************************************************
end
Information contained in this publication regarding device applications and the like is intended through suggestion only and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. No representation or warranty is given and no liability is assumed by Microchip. No licenses are conveyed, implicitly or otherwise, except as maybe explicitly expressed herein, under any intellectual property rights. The Microchip logo and name are registered trademarks of Microchip Technology Inc. in the U.S.A. and other countries. All rights reserved. All other trademarks mentioned herein are the property of their respective companies.

Microchip received QS-9000 quality system certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona in July 1999. The Company’s quality system processes and procedures are QS-9000 compliant for its PICmicro® 8-bit MCUs, KEELOQ® code hopping devices, Serial EEPROMs and microperipheral development systems is ISO 9001 certified.