Order this document
by AN1791/D

Motorola Semiconductor Application Note

AN1791

Using the QADC64 Module on the M68F375 Microcontroller

By **Austin Obobaifo and Thomas Walter**
PowerTrain Systems Division
Austin, Texas

Introduction

This application note provides useful material for using the MPFB1632 (modular platform board) and MPBF375BGA modular platform and evaluation boards. It highlights effective use of the queued analog-to-digital converter (QADC64) module.

Automotive applications often require analog-to-digital (ATD) conversion. Such applications include:

- Water and air temperature measurement
- Mass air pressure
- Throttle engine position measurement
- Mass air flow
- O_2 sensor
- Oil and fuel pressure sensor

In automotive evaluation boards, the QADC64 inputs require additional filtering to protect against electrostatic discharges and filter noise. Research results concerning useful filtering circuits for varying applications are included here for use by microcontroller developers.

Further information on the M68F375 can be found in the M68F375 user's manual at http://www.mcu.motsps.com/lit/.

It is assumed that the developer will use the SingleStep™ debug software tool, designed by Software Development Systems, Inc. (SDS), to communicate with the microcontroller through the Background Debug™ mode (BDM) port. SDS handles licensing of the software.

The main features of this application note are:

- M68F375 overview
- MPFB1632/MPBF375BGA modular platform/evaluation boards overview
- Setting up and initializing the modular platform/evaluation board
- QADC64 module overview
- Initializing the QADC64 module
- Example of practical applications
- QADC64 Input signal conditioning
- Measuring digital output voltages
- Useful application hardware and software

M68F375 Features

The M68F375 is a member of the MC68300/68HC16 Family of modular microcontrollers. Featured in its modules are:

- MC68000 Family central processor unit, CPU32
- Single-chip integration module, SCIM2E
- 10-bit queued A/D converter module with analog multiplexer, QADC64/AMUX

SingleStep is a trademark of Software Development Systems, Inc.
Background Debug is a trademark of Motorola, Inc.

- Queued serial multi-channel communication module, QSMCM
- Third generation time processor unit, TPU3
- 6-Kbyte dual port TPU3 emulation RAM module, DPTRAM
- 256-K FLASH EEPROM module, CMFI
- 8-K RAM module, SRAM
- Four 512-byte overlay static RAM modules, SRAM
- 8-K ROM module, ROM
- CAN module, TOUCAN
- Configurable timer module, CTM9

QADC64

The QADC consists of an analog front-end and a digital control subsystem, which includes an intermodule bus (IMB3) interface block. The analog section includes input pins, channel selection logic, an analog multiplexer, and one sample and hold analog circuit. The digital section contains the conversion sequencing logic. Also included are the periodic/interval timer, control and status registers, the conversion command word (CCW) table RAM, and the result word table RAM.

The QADC64 offers these features:

- Internal sample and hold
- Up to 16 analog input channels using internal multiplexing
- Directly supports up to four external multiplexers
- Up to 41 total input channels with internal and external multiplexing
- Programmable input sample time for various source impedances
- Two conversion command queues with a total of 64 entries
- Sub-queues possible using pause mechanism
- Queue complete and pause software interrupts available on both queues

- Queue pointers indicate current location for each queue
- Automated queue modes indicated by:
  - External edge trigger, queues 1 and 2, and gated mode, queue 1 only
  - Periodic/interval timer, within QADC64 module, queues 1 and 2
  - Software command, queues 1 and 2
- Single scan or continuous scan of queues
- 54 result registers
- Output data readable in three formats:
  - Right-justified unsigned
  - Left-justified signed
  - Left-justified unsigned
- Unused analog channels can be used as digital ports

M68MPFB1632 Overview

The MPFB is one component of the M68MEVB1632 modular evaluation board (MEVB). The MEVB consists of the M68MPFB1632 modular platform board, a microcontroller unit personality board (MPB), and a developer choice of debugging software.

The MEVB is an economical tool for designing, debugging, and evaluating MCU operation of the MC68300 MCU Families. By providing the essential MCU timing and I/O (input/output) circuitry, the MEVB simplifies user evaluation of prototype hardware/software products. The MEVB requires user-supplied power and host computer.

The MPFB only operates with MPBs that support system integration module (SIM) or single-chip integration module (SCIM) MCUs.

Features

MPFB features include:

- Platform which provides interface and power connections for the MPB
- On-board support for multiple memory devices, types (RAM, EPROM, and FLASH EEPROM), and sizes (32 Kbytes to 512 Kbytes)
- Two RS-232C terminal input/output (I/O) ports for user evaluation of the serial communication interface (SCI)
- Logic analyzer pod connectors for all MCU pins
- Port replacement unit (PRU) to rebuild I/O ports lost to address/data/control
- On-board V_{PP} (+12 Vdc) generation for MCU and FLASH EEPROM programming
- On-board wire-wrap area
- Development software:
  - SingleStep debug software tool
  - ICD16/ICD32 user interface for the MEVB debugger
  - PROG16/PROG32 programmer for the M68300 Family MCUs
  - IASM-integrated assembler/editor

MPFB Layout

Figure 1 illustrates the modulator platform board layout and **Figure 2** shows operating jumper settings for the MPBF375BGA daughter personality board.

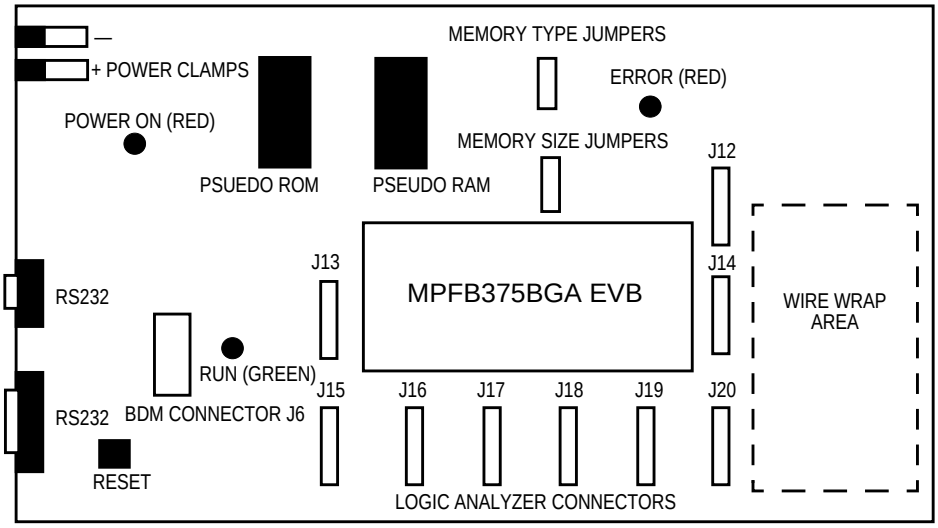


Figure 1. Modular Platform Board Layout

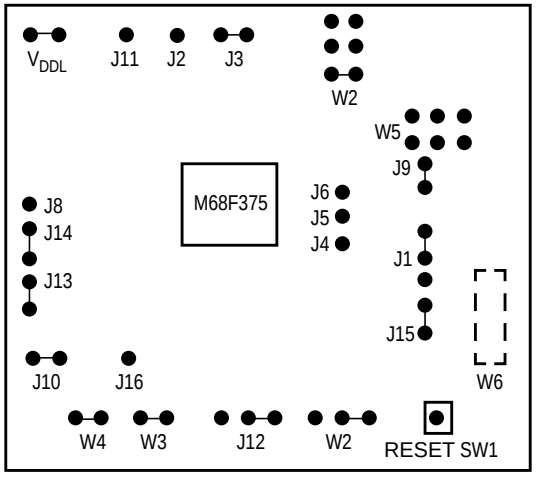


Figure 2. MPFB375BGA EVB Jumper Settings

Installing the Software Tools and Interface

NOTE: *These instructions describe the installation and use of SDS debug version 7.3.1 using the Macraigor wiggler (interface). Refer to the instructions of the particular tool that is being used.*

Installing SDS Debug on a PC

For the M68F375, SingleStep version 7.3.1 or later version of the software is required. A license from SDS is required to run this software. This will either be provided on a disk or by email. Create the *license.dat* file from the email and keep it handy for the installation.

To install:

1. Insert the SDS SingleStep debug CD-ROM into the CD drive.
2. Choose **RUN** from the **File** or **Start** menu.
3. Type *D:\setup* and press <**RETURN**> (where *D:* is the CD drive).
4. Follow the installation steps and insert the disk containing the *license.dat* file when prompted. Select the CPU32 set of tools.

Next, create and install the configuration files for the M68MPFB1632 evaluation board. These commands will set up some aliases and chip selects for the external FLASH and SRAM on the board.

1. Create a directory under a root called *cerberus*.
2. Using a text editor, create a file containing the following text and name the file *sstep.ini*.

```
# This "sstep.ini" file is provided for supporting the
# Motorola Cerberus EVB

set vectskip = ( 0-255 )
alias _config "source c:\cerberus\cerberus.dbg"
```

The last section of this file instructs SDS SingleStep to use the file *cerberus.dbg* to set up the MPBF375BGA evaluation board with the right chip selects.

3. Save the text file *sstep.ini* in the *cerberus* directory created earlier.
4. Next, use the following text to create the *cerberus.dbg* file and save it in the *cerberus* directory.

NOTE: *This initialization code sets up several options that may not be applicable to the user's specific application. They are listed for completeness and reference. This code is only applicable to the MPBF375BGA EVB. Any line starting with # (the pound sign) is a comment and does not have to be included in the file. Refer to the cerberus manual at http://www.mcu.motsps.com/lit for specific module register information.*

```
# This "cerberus.dbg" file is provided for supporting the
# Motorola cerberus MPB on MFPB1632 platform board
#
# Version 0.5 - Randy Dees - November 9,1998
#      added Overlay SRAM initialization
# Version 0.6 - Austin Obobaifo - January 7,1999
#      added optional CSBOOT initialization
# Version 0.7 - Randy Dees - January 7,1999
#      added optional internal flash initialization
#      changed CSBOOT setup parameters
#      added additional comments
# Version 0.8 - Randy Dees - January 11,1999
#      updated CSBOOT setup parameters
# Version 0.9 - Mike Ferguson/Randy Dees - January 18,1999
#      set STOP and LOCK in CMFIMCR before writing CMFIBA

# turn off setting of interrupts in SDS
set vectskip = ( 0-255 )

# set up SYPCR
# SWE = 0 - turn off software watchdog
# SWP=0 - watchdog not prescaled
# SWT = 00 - 2^9 /fsys
# HME = 0 - halt monitor off
# BME = 0 - Bus monitor on
# BMT = 10 - 16 system clocks
write -b SD:0xfffa21 = 0x06

# set up SCIMMCR
# FRZSW=1 - stop PIT & SWDG in BDM mode
# SUPV=1 register accesses in Supervisor mode only
# MM=1 address space to FFF000-FFFFFFFF
# IARB=1111
write -w SD:0xfffa00 = 0x40cf

# set clock to 16 MHz
write -w SD:0xfffa04=0x7008
# for 24 MHZ uncomment the following line
# write -w SD:0xfffa04=0xD008
# for 32 MHZ uncomment the following line
# write -w SD:0xfffa04=0xF008

# set up DPTRAM to address 0xff0000
write -w 0xfff884=0xff00

# set SRAM base to 0x200000
write -w SD:0xfffb04=0x0020
write -w SD:0xfffb06=0x0000
# Enable internal SRAM at location
write -w SD:0xfffb00 = 0x0000
```

```
# set overlay SRAM1 base to 0x202000
write -w SD:0xfff844=0x0020
write -w SD:0xfff846=0x2000
# Enable internal SRAM at location
write -w SD:0xfff840 = 0x0000

# set overlay SRAM2 base to 0x202200
write -w SD:0xfff84c=0x0020
write -w SD:0xfff84e=0x2200
# Enable internal SRAM at location
write -w SD:0xfff848 = 0x0000

# set overlay SRAM3 base to 0x202400
write -w SD:0xfff854=0x0020
write -w SD:0xfff856=0x2400
# Enable internal SRAM at location
write -w SD:0xfff850 = 0x0000

# set overlay SRAM4 base to 0x202600
write -w SD:0xfff85c=0x0020
write -w SD:0xfff85e=0x2600
# Enable internal SRAM at location
write -w SD:0xfff858 = 0x0000

#####################################################################
# Optionally Set up Internal FLASH
# CMFIMCR
# add comment symbol to disable
write -w SD:0xFFF800=0x0000

# CMFICTL1 (disable all options)
# add comment symbol to disable
write -w SD:0xfff80c=0x0000

#CMFICTL2 disable all options)
# add comment symbol to disable
write -w SD:0xfff80e=0x0000

# set base address to 0
#CMFIBA
# add comment symbol to disable
write -w SD:0xFFF800=0x8800
write -l SD:0xFFF808=0x0
# turn flash back on
write -w SD:0xFFF800=0x0000

#####################################################################
# CSBOOT (External pseudo ROM on MFPB1632)

# CSBARBT - Chip select Boot Base Address register
# Block size of 128K bytes
```

```
# Set base address to 0x80000
# Optionally change base to 0 for emulation of
#    internal flash by external RAM
# add comment symbol to disable
write -w SD:0xfffa48=0X0804

# CSORBT CS Boot Option Register
# Asynchronous, 16 bit, data strobe, 3 clock access
# Supervisor/User space
# set number of wait states inserted to
# optimize bus speed for your specific application
# 0b0111 1111 1011 0001
# add comment symbol to disable
write -w SD:0xfffa4a=0x7C30

# CSPAR0 - Enable Chip select for CSBOOT
# add comment symbol to disable
# set port to 16 bit width
write -w SD:0xfffa44=0X0003
```

NOTE: *These three switches can be used with the write and read commands:*

- *-b, Byte read/write*
- *-w, 16-bit half word*
- *-l, 32-bit longword*

5. Create a new windows short cut for the SDS SingleStep program using the following steps:

- Click on the **Start** menu, **Settings** and **Taskbar.**
- Click on the **Start Menu Programs** tab.
- Click on the **Advanced** button.
- Open the **Programs** folder.
- Open the **SingleStep 7.3.1** folder.
- Copy the **SingleStep On-Chip** link.
- Rename this link *cerberus.*
- Right click the new link and select **Properties.**
- Make sure the target path is: *c:**sds731**cmd**bdm68k.exe.*
- Make sure the program starts in *C:**cerberus* or the default installation folder.

Debugging the M68F375 using SingleStep should be an easy launch from the windows program's menu after completing the above steps. After installation of the software, the user is ready to attach the BDM interface. Supplied with the SDS SingleStep on-chip debugger software will be a parallel-to-serial interface specifically for the M68F375 series of microcontrollers. This interface is necessary to allow the PC to communicate via its parallel port to the BDM port on the MPFB.

Connecting the MPFB to a PC

Refer to **Figure 1** and **Figure 3** for these steps:

1. To the parallel port on the PC, connect a cable with a female, 25-pin D-SUB connector on the other end.
2. To the female end, attach the Macraigor BDM interface, ensuring that the D-SUB connector is secured firmly.
3. Attach the ribbon cable at the opposite end of the interface to the evaluation board using the 10-pin, r-row header (Berg BDM connector) in the top right corner of the board. (See **Figure 1** and **Figure 3**). Make sure the red line on the ribbon is aligned with pin 1.

NOTE: *The connector at the end of the ribbon cable is keyed so that it can only fit correctly, but it must be pushed down all the way for adequate connection.*

4. With the PC connected to the interface and the interface attached to the evaluation board, power the modular platform board using a 5-V source.

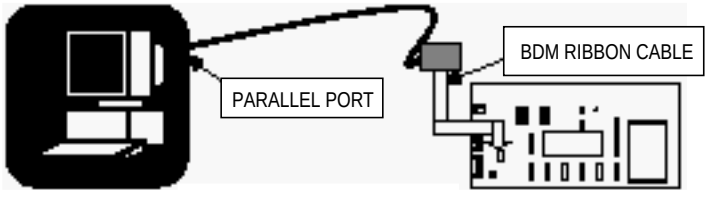


Figure 3. BDM Interface Connection

Initializing the Hardware

Starting SDS SingleStep Debug

Follow these steps to start the SDS SingleStep debug:

1. Start the SDS SingleStep on-chip software. A tabbed dialogue will be shown.
2. Under the **File** tab, select the **Debug without a file** radio button.
3. Choose the **Processor** tab and select CPU32 from the drop down list. There is no coprocessor.
4. Click **OK** and the PC will connect to the Cerberus. Look for **Started Successfully** message in the debug session dialogue box. Choose **Close**. If the Cerberus needs to be reconnected, start the debug session again by selecting **Debug…** from the file menu.
5. From the window drop down menu, select the **Command** window. There should be a window with a **SingleStep>** prompt. This interface will be used to write and read registers on the microcontroller. Clicking on the **RAM** icon will also provide a means of observing and manipulating memory locations.

Initializing the QADC64 Module

Follow these steps to initialize the QADC64 module:

1. Create the following file for initializing the QADC64 module using a text editor.

```
#  Qadc1.dbg  script for SDS 7.3.1 debugger.
#
#  Austin Obobaifo / Thomas Walter
#  Mon 02 Nov 1998
#
#  Script for turning on the QADC, and letting it run in "free run mode"
#
#  Results for rrj values are located at 0xfff400
#  to see the values, do a read -w 0xfff400
#
#  Note first line is QADCA - Chan8 through 15 PAQ0-PQA7 (AN 52 to AN59)
#  This script also scans Vrl VRh and (Vrh-Vrl)2
#

# set QADC64 mcr for normal operation (should be default)

write -w 0xfff400 = 0x0

# set QACR0 to have a high prescale of 4 & low prescale of 4.  /* PSH = 3   PSL = 3*/

write -w 0xfff40A = 0x0033
#Internally muxed. (Non Muxed externally)

# set QACR2 to disable Queue2, and only run queue 1.  Note, this should be a
#  default value      /* disable que2 */

write -w 0xfff40E = 0x0040

# set up CCW command conversion word

# ccw0_a=0x00F4;             /* conversion command word.  Analog 0 = AN52 */
write -w 0xfff600 = 0x00F4


# ccw1_a=0x00F5;             /* conversion command word.  Analog 1 = AN53 */
write -w 0xfff602 = 0x00F5

# ccw2_a=0x00F6;             /* conversion command word.  Analog 2 = AN54 */
write -w 0xfff604 = 0x00F6

# ccw3_a=0x00F7;             /* conversion command word.  Analog 3 = AN55 */
write -w 0xfff606 = 0x00F7
```

```
# ccw4_a=0x00F8;              /* conversion command word.  Analog 4 = AN56 */
write -w 0xfff608 = 0x00F8

# ccw5_a=0x00F9;              /* conversion command word.  Analog 5 = AN57 */
write -w 0xfff60a = 0x00F9

# ccw6_a=0x00FA;              /* conversion command word.  Analog 6 = AN58 */
write -w 0xfff60C = 0x00FA

# ccw7_a=0x00FB;              /* conversion command word.  Analog 7 = AN59 */
write -w 0xfff60E = 0x00FB

# ccw8_a=0x00FC;              /* conversion command word.  Analog   = Vrl debugging */
write -w 0xfff610 = 0x00FC


# ccw9_a=0x00FD;              /* conversion command word.  Analog   = Vrh debugging */
write -w 0xfff612 = 0x00FD

# ccw7_a=0x00FE;              /* conversion command word.  Analog   = (Vrh-Vrl)/2 */
write -w 0xfff614 = 0x00FE

# ccw8_a=0x00FF;              /* conversion command word.  End of Queue command */
write -w 0xfff616 = 0x00FF
#longer but more accurate sample time

# set QACR1 for continuous scan mode   /*Control Register 1*/
write -w 0xfff40C  = 0x1400
```

2. Save this file as *QADC64.dbg* in the default *SDS Startup* directory.
3. To load and execute the file, enter the path to the file when SDS starts using **Debug with file option** or type the following in the **Command** window. This file will prepare the QADC module to receive input signals and perform required conversions.

SingleStep> C:\"default startup directory where init.dbg resides"\QADC64.dbg <RETURN>

NOTE: *Details of the conversion modes can be found in the Queued Analog-to-Digital converter module section of the cerberus user manual at http://www.motsps.com/lit/.*

Alternative Multiplexed Channel Assignment File

Create this alternative multiplexed channel assignment file.

```
#  Qadc2.dbg  script for SDS 7.3.1 debugger.
#  Austin Obobaifo
#  Frid DEC 11 98
#  Script for turning on the QADC, and letting it run in "free run mode"
#
#  Results for rrj values are located at 0xfff400
#  to see the values, do a read -w 0xfff400
#
#  note first line is QADCA - Chan8 through 15 PAQ0-PQA7 (AN 52 to AN59)
#  This script also scans Vrl VRh and (Vrh-Vrl)2
#

# set mcr for normal operation (should be default)

write -w 0xfff400 = 0x0

# set QACR0 to have a high prescale of 4 & low prescale of 4.  /* PSH = 3   PSL = 3*/

write -w 0xfff40A = 0x8033
#Externally muxed.

# set QACR2 to disable Queue2, and only run queue 1.  Note, this should be a
#  default value      /* disable que2 */

write -w 0xfff40E = 0x0040

# set up CCW command conversion word
#Externally multiplexed Channel assignments

#00C0 sets the IST bits in the CCW register for [QCLK X 16]
#This permits longer sample time while performing conversions
#on one channel:0000 (PQB0).
#A developer can send a pulse at a time and observe result conversion.

# ccw0_a=0x00C0;              /* conversion command word.  Analog 0 = AN52 */
write -w 0xfff600 = 0x00C0

# ccw1_a=0x00C0;              /* conversion command word.  Analog 1 = AN53 */
write -w 0xfff602 = 0x00C0

# ccw2_a=0x00C0;              /* conversion command word.  Analog 2 = AN54 */
write -w 0xfff604 = 0x00C0

# ccw3_a=0x00C0;              /* conversion command word.  Analog 3 = AN55 */
write -w 0xfff606 = 0x00C0
```

```
# ccw4_a=0x00C0;              /* conversion command word.  Analog 4 = AN56 */
write -w 0xfff608 = 0x00C0

# ccw5_a=0x00C0;              /* conversion command word.  Analog 5 = AN57 */
write -w 0xfff60a = 0x00C0

# ccw6_a=0x00C0;              /* conversion command word.  Analog 6 = AN58 */
write -w 0xfff60C = 0x00C0

# ccw7_a=0x00C0;              /* conversion command word.  Analog 7 = AN59 */
write -w 0xfff60E = 0x00C0

# ccw8_a=0x00C0;              /* conversion command word.  Analog   = Vrl debugging */
write -w 0xfff610 = 0x00C0

# ccw9_a=0x00C0;              /* conversion command word.  Analog   = Vrh debugging */
write -w 0xfff612 = 0x00C0

# ccw7_a=0x00C0;              /* conversion command word.  Analog   = (Vrh-Vrl)/2 */
write -w 0xfff614 = 0x00C0


# ccw8_a=0x00FF;              /* conversion command word.  End of Queue command */
write -w 0xfff616 = 0x00FF
#longer but more accurate sample time

# set QACR1 for continuous scan   /*Control Register 1*/
write -w 0xfff40C  = 0x1400
```

Alternative C Initialization file

Create the following C file to initialize the QADC64. Be sure to include the header file also.

```
/* Qadc.c  Program to allow QADC_A channel A0-7 to keep converting signals and placing
them into the memory location.
*
*  18 NOV 98 -- Austin Obobaifo
*/

#include "C:\your path\Qadc64.h"

void qadc_init()

{
  unsigned int i;


/* The QADC64 Module configuration register.    pg. 13-32 Black Oak Manual  */

qadc64mcr_a.stop = 0;             /* Enable the QADC64  */
```

```
qadc64mcr_a.frz  = 0;             /* Ignore the freeze bit */
qadc64mcr_a.supv  = 0;            /* allow user access */


/*  The QADC64 Control register 0       pg. 13-34        */

qacr0_a.mux = 0;                  /* internally muxed, 16 channels */
qacr0_a.trg = 0;                  /* trigger assignment */
qacr0_a.psh = 0;
qacr0_a.psa = 0x3;                /* high prescale of 4 */
qacr0_a.psl = 0x3;                /* low prescale of 4 */

/* The QADC64 Control register 1        pg. 13-35        */
qacr1_a.cie1=0;                   /* queue completion interrupts disabled */
qacr1_a.pie1=0;                   /* queue 1 pause interrupts disabled */
qacr1_a.sse1=0;                   /* single scan enable */
qacr1_a.mq1=0x14;                 /* Queue 1 Operating Mode -- continuous scan */
qacr2_a.mq2=0x40;                 /* disable Queue 2 -- default value */


/*  CCW */

CCW0_a=0xF4;                      /* conversion command word.  Analog 0 = AN52 */
CCW1_a=0xF5;                      /* conversion command word.  Analog 1 = AN53 */
CCW2_a=0xF6;                      /* conversion command word.  Analog 2 = AN54 */
CCW3_a=0xF7;                      /* conversion command word.  Analog 3 = AN55 */
CCW4_a=0xF8;                      /* conversion command word.  Analog 4 = AN56 */
CCW5_a=0xF9;                      /* conversion command word.  Analog 5 = AN57 */
CCW6_a=0xFA;                      /* conversion command word.  Analog 6 = AN58 */
CCW7_a=0xFB;                      /* conversion command word.  Analog 7 = AN59 */
CCW8_a=0xFC;                      /* conversion command word.  Vrh */
CCW9_a=0xFD;                      /* conversion command word.  Vrl */
CCW10_a=0xFE;                     /* conversion command word.  (Vrh-Vrl)/2 */
CCW11_a=0xFF;                     /* conversion command word.  End of Queue command */


/* results are located at 0x304A80 --> 304A8E    00 --> 007 */
}
qadc_run()
{
qacr1_a.mq1 = 0x14;
}

main()
{
qadc_init();
qadc_run();
}
```

QADC64 Header File

Use the following code for the QAD64 header file.

```
/****************************************************************************
*       qadc64.h  The Header file for the QADC64 in the Cerberus
*       Revision History:
*       qadc_rev001.h                       Austin Obobaifo16NOV98
*                                           original make. Consistent with
*                                           User Manual dated 2Sep98
*       Registers are declared twice, one in all lower case,
*       the other in all upper case.

*               "regname" is a bit-mapped struct variable which
*                  requires a subfield.  The default sub-
*                  field is "bit0","bit1",... where "bit15"
*                  is the most-significant bit.

*               "REGNAME" is the entire register expressed as a
*                  long.  This form is a macro expansion
*                  derived from the lower case equivalent,
*                  Therefore it will not be seen in the
*                  linker's output file.

*               Exceptions: All special purpose registers(SPRs).

****************************************************************************/

#ifndef _QADC_H
#define _QADC_H

/****************************************************************************

* The QADC64 Module configuration register.  pg. 13-32

****************************************************************************/
typedef struct {
       unsigned stop:                      1;
       unsigned frz:                       1;
       unsigned reserved8_13:              6;
       unsigned supv:                      1;
       unsigned reserved9_15:              7;
       } Qadc64mcr;
#define qadc64mcr_a (*(Qadc64mcr *)0xFFF400)
#define QADC64MCR_A (*(volatile unsigned short *)0xFFF400)

/****************************************************************************

* The QADC64 Interrupt register.            pg. 13-32

****************************************************************************/
```

```
typedef struct {
       unsigned irl1:                      5;
       unsigned irl2:                      5;
       unsigned reserved10_15:             6;
       } Qadc64int;
#define qadc64int_a (*(Qadc64int *)0xFFF404)
#define QADC64INT_A (*(volatile unsigned short *)0x304804)

/*****************************************************************************

* The QADC64 Port A/B Data register.  pg. 13-33

*****************************************************************************/

typedef struct {
       unsigned pqa7:                      1;
       unsigned pqa6:                      1;
       unsigned pqa5:                      1;
       unsigned pqa4:                      1;

unsigned pqa3:               1;
       unsigned pqa2:        1;
       unsigned pqa1:        1;
       unsigned pqb0:        1;
       unsigned pqb7:        1;
       unsigned pqb6:        1;
       unsigned pqb5:        1;

unsigned pqb4:               1;
       unsigned pqb3:        1;
       unsigned pqb2:        1;
       unsigned pqb1:        1;
       unsigned pqb0:        1;
       } Portq;
#define portqa_b (*(Portq *)0xFFF406)
#define PORTQA_B (*(volatile unsigned short *)0xFFF406)

/*****************************************************************************

* The QADC64 Data direction register  pg. 13-34

*****************************************************************************/
typedef struct {
       unsigned ddqa7:       1;
       unsigned ddqa6:       1;
       unsigned ddqa5:       1;
       unsigned ddqa4:       1;
       unsigned ddqa3:       1;
       unsigned ddqa2:       1;
       unsigned ddqa1:       1;
```

```
       unsigned ddqa0:       1;
       unsigned reserved0_7: 8;
       } Dddrqa;
#define ddrqa_a (*(Dddrqa *)0xFFF408)
#define DDRQA_A (*(volatile unsigned short *)0xFFF408)

/*****************************************************************************

* The QADC64 Control register 0  pg. 13-34

*****************************************************************************/

typedef struct {

unsigned mux:                              1;
       unsigned reserved13_14:             2;
       unsigned trg:                       1;
       unsigned reserved4_6:               3;
       unsigned psh:                       5;
       unsigned psa:                       1;
       unsigned psl:                       3;
       } Qacr0;
#define qacr0_a (*(Qacr0 *)0xFFF40A)
#define QACR0_A (*(volatile unsigned short *)0xFFF40A)
/*****************************************************************************

* The QADC64 Control register 1  pg. 13-35

*****************************************************************************/
typedef struct {
       unsigned cie1:                      1;
       unsigned pie1:                      1;
       unsigned sse1:                      1;
       unsigned mq1:                       5;
       unsigned reserved0_7:               8;
       } Qacr1;
#define qacr1_a (*(Qacr1 *)0xFFF40C)
#define QACR1_A (*(volatile unsigned short *)0xFFF40C)

/*****************************************************************************

* The QADC64 Control register 2  pg. 13-38

*****************************************************************************/
typedef struct {
       unsigned cie2:                      1;
       unsigned pie2:                      1;
       unsigned sse2:                      1;
       unsigned mq2:                       5;
       unsigned resume:                    1;
       unsigned bq2:                       7;
```

```
       } Qacr2;
#define qacr2_a (*(Qacr2 *)0xFFF40e)
#define QACR2_A (*(volatile unsigned short *)0xFFF40e)

/*****************************************************************************

* The QADC64 Status register 0  pg. 13-40

*****************************************************************************/

typedef struct {
       unsigned cf1:                        1;
       unsigned pf1:                        1;
       unsigned cf2:                        1;
       unsigned pf2:                        1;
       unsigned tor1:                       1;
       unsigned tor2:                       1;
       unsigned qs:                         4;
       unsigned cwp:                        6;
       } Qasr0;

#define qasr0_a (*(Qasr0 *)0xFFF410)
#define QASR0_A (*(volatile unsigned short *)0xFFF410)

/*****************************************************************************

* The QADC64 Status register 1  pg. 13-41

*****************************************************************************/

typedef struct {
       unsigned reserved14_15:              2;
       unsigned cwpq1:                      6;
       unsigned reserved8_9:                2;
       unsigned cwpq2:                      6;
       } Qasr1;
#define qasr1_a (*(Qasr1 *)0xFFF412)
#define QASR1_A (*(volatile unsigned short *)0xFFF412)

/*****************************************************************************

* The QADC64 Conversion command word table   pg. 13-45

*      Note: There are 64 words of these for each QADC

*****************************************************************************/

//Conversion Command Word Table

typedef struct {
       unsigned reserved10_15:              6;
```

```
        unsigned p:                         1;
        unsigned byp:                       1;
        unsigned ist:                       2;
        unsigned chan:                      6;
        } Ccw;
#define ccw0_a (*(Ccw *)0xFFF600)
#define ccw1_a (*(Ccw *)0xFFF602)
#define ccw2_a (*(Ccw *)0xFFF604)
#define ccw3_a (*(Ccw *)0xFFF606)
#define ccw4_a (*(Ccw *)0xFFF608)
#define ccw5_a (*(Ccw *)0xFFF60a)
#define ccw6_a (*(Ccw *)0xFFF60c)
#define ccw7_a (*(Ccw *)0xFFF60e)
#define ccw8_a (*(Ccw *)0xFFF610)
#define ccw9_a (*(Ccw *)0xFFF612)
#define ccw10_a (*(Ccw *)0xFFF614)
#define ccw11_a (*(Ccw *)0xFFF616)
#define ccw12_a (*(Ccw *)0xFFF618)
#define ccw13_a (*(Ccw *)0xFFF61a)
#define ccw14_a (*(Ccw *)0xFFF61c)
#define ccw15_a (*(Ccw *)0xFF61e)
#define ccw16_a (*(Ccw *)0xFFF620)
#define ccw17_a (*(Ccw *)0xFFF622)
#define ccw18_a (*(Ccw *)0xFFF624)
#define ccw19_a (*(Ccw *)0xFFF626)
#define ccw20_a (*(Ccw *)0xFFF628)
#define ccw21_a (*(Ccw *)0xFFF62a)
#define ccw22_a (*(Ccw *)0xFFF62c)
#define ccw23_a (*(Ccw *)0xFFF62e)
#define ccw24_a (*(Ccw *)0xFFF630)
#define ccw25_a (*(Ccw *)0xFFF632)
#define ccw26_a (*(Ccw *)0xFFF634)
#define ccw27_a (*(Ccw *)0xFFF636)
#define ccw28_a (*(Ccw *)0x304a38)

#define ccw29_a (*(Ccw *)0xFFF63a)
#define ccw30_a (*(Ccw *)0xFFF63c)
#define ccw31_a (*(Ccw *)0xFFF63e)
#define ccw32_a (*(Ccw *)0xFFF640)
#define ccw33_a (*(Ccw *)0xFFF642)
#define ccw34_a (*(Ccw *)0xFFF644)
#define ccw35_a (*(Ccw *)0xFFF646)
#define ccw36_a (*(Ccw *)0xFFF648)
#define ccw37_a (*(Ccw *)0xFFF64a)
#define ccw38_a (*(Ccw *)0xFFF64c)
#define ccw39_a (*(Ccw *)0xFFF64e)
#define ccw40_a (*(Ccw *)0xFFF650)
#define ccw41_a (*(Ccw *)0xFFF652)
#define ccw42_a (*(Ccw *)0xFFF654)
#define ccw43_a (*(Ccw *)0xFFF656)
#define ccw44_a (*(Ccw *)0xFFF658)
```

```
#define ccw45_a (*(Ccw *)0xFFF65a)
#define ccw46_a (*(Ccw *)0xFFF65c)
#define ccw47_a (*(Ccw *)0xFFF65e)
#define ccw48_a (*(Ccw *)0xFFF660)
#define ccw49_a (*(Ccw *)0xFFF662)
#define ccw50_a (*(Ccw *)0xFFF664)
#define ccw51_a (*(Ccw *)0xFFF666)
#define ccw52_a (*(Ccw *)0xFFF668)
#define ccw53_a (*(Ccw *)0xFFF66a)
#define ccw54_a (*(Ccw *)0xFFF66c)
#define ccw55_a (*(Ccw *)0xFFF66e)
#define ccw56_a (*(Ccw *)0xFFF670)
#define ccw57_a (*(Ccw *)0xFFF672)
#define ccw58_a (*(Ccw *)0xFFF674)
#define ccw59_a (*(Ccw *)0xFFF676)
#define ccw60_a (*(Ccw *)0xFFF678)
#define ccw61_a (*(Ccw *)0xFFF67a)
#define ccw62_a (*(Ccw *)0xFFF67c)
#define ccw63_a (*(Ccw *)0xFFF67e)
#define ccw0_b (*(Ccw *)0xFFF900)

#define ccw1_b (*(Ccw *)0xFFF902)
#define ccw2_b (*(Ccw *)0xFFF904)
#define ccw3_b (*(Ccw *)0xFFF906)
#define ccw4_b (*(Ccw *)0xFFF908)
#define ccw5_b (*(Ccw *)0xFFF90a)
#define ccw6_b (*(Ccw *)0xFFF90c)
#define ccw7_b (*(Ccw *)0xFFF90e)
#define ccw8_b (*(Ccw *)0xFFF910)
#define ccw9_b (*(Ccw *)0xFFF912)

#define ccw10_b (*(Ccw *)0xFFF914)
#define ccw11_b (*(Ccw *)0xFFF916)
#define ccw12_b (*(Ccw *)0xFFF918)
#define ccw13_b (*(Ccw *)0xFFF91a)
#define ccw14_b (*(Ccw *)0xFFF91c)
#define ccw15_b (*(Ccw *)0xFFF91e)
#define ccw16_b (*(Ccw *)0xFFF920)
#define ccw17_b (*(Ccw *)0xFFF922)
#define ccw18_b (*(Ccw *)0xFFF924)
#define ccw19_b (*(Ccw *)0xFFF926)
#define ccw20_b (*(Ccw *)0xFFF928)
#define ccw21_b (*(Ccw *)0xFFF92a)
#define ccw22_b (*(Ccw *)0xFFF92c)
#define ccw23_b (*(Ccw *)0xFFF92e)
#define ccw24_b (*(Ccw *)0xFFF930)
#define ccw25_b (*(Ccw *)0xFF932)
#define ccw26_b (*(Ccw *)0xFFF934)
#define ccw27_b (*(Ccw *)0xFFF936)
#define ccw28_b (*(Ccw *)0xFFF938)
#define ccw29_b (*(Ccw *)0xFFF93a)
```

```
#define ccw30_b (*(Ccw *)0xFFF93c)
#define ccw31_b (*(Ccw *)0xFFF93e)
#define ccw32_b (*(Ccw *)0xFFF940)
#define ccw33_b (*(Ccw *)0xFFF942)
#define ccw34_b (*(Ccw *)0xFFF944)
#define ccw35_b (*(Ccw *)0xFFF946)
#define ccw36_b (*(Ccw *)0xFFF948)
#define ccw37_b (*(Ccw *)0xFFF94a)
#define ccw38_b (*(Ccw *)0xFFF94c)
#define ccw39_b (*(Ccw *)0xFFF94e)
#define ccw40_b (*(Ccw *)0xFFF950)
#define ccw41_b (*(Ccw *)0xFFF952)

#define ccw42_b (*(Ccw *)0xFFF954)
#define ccw43_b (*(Ccw *)0xFFF956)
#define ccw44_b (*(Ccw *)0xFFF958)
#define ccw45_b (*(Ccw *)0xFFF95a)
#define ccw46_b (*(Ccw *)0xFFF95c)
#define ccw47_b (*(Ccw *)0xFFF95e)
#define ccw48_b (*(Ccw *)0xFFF960)
#define ccw49_b (*(Ccw *)0xFFF962)
#define ccw50_b (*(Ccw *)0xFFF964)
#define ccw51_b (*(Ccw *)0xFFF966)
#define ccw52_b (*(Ccw *)0xFFF968)

#define ccw53_b (*(Ccw *)0xFFF96a)
#define ccw54_b (*(Ccw *)0xFFF96c)
#define ccw55_b (*(Ccw *)0xFFF96e)
#define ccw56_b (*(Ccw *)0xFFF970)
#define ccw57_b (*(Ccw *)0xFFF972)
#define ccw58_b (*(Ccw *)0xFFF974)
#define ccw59_b (*(Ccw *)0xFFF976)
#define ccw60_b (*(Ccw *)0xFFF978)
#define ccw61_b (*(Ccw *)0xFFF97a)
#define ccw62_b (*(Ccw *)0xFFF97c)
#define ccw63_b (*(Ccw *)0xFFF97e)

/*****************************************************************************
******************************************************************************

* The QADC64 Result word tables.               pg. 13-48

*      Note: There 3 sets of 64 words of these for each QADC.

******************************************************************************
```

```
*****************************************************************************/
/*****************************************************************************

* The QADC64 Right justified unsigned result register. pg. 13-48

*       Note: There are 64 words of these for each QADC.

*****************************************************************************/


typedef struct {
       unsigned reserved10_15:              6;
       unsigned result:      10;
       } Rjurr;

#define rjurr0_a (*(Rjurr *)0xFFF680)
#define rjurr1_a (*(Rjurr *)0xFFF682)
#define rjurr2_a (*(Rjurr *)0xFFF684)
#define rjurr3_a (*(Rjurr *)0xFFF686)
#define rjurr4_a (*(Rjurr *)0xFFF688)
#define rjurr5_a (*(Rjurr *)0xFFF68a)

#define rjurr6_a (*(Rjurr *)0xFFF68c)
#define rjurr7_a (*(Rjurr *)0xFFF68e)
#define rjurr8_a (*(Rjurr *)0xFFF690)
#define rjurr9_a (*(Rjurr *)0xFFF692)
#define rjurr10_a (*(Rjurr *)0xFFF894)
#define rjurr11_a (*(Rjurr *)0xFFF696)
#define rjurr12_a (*(Rjurr *)0xFFF698)
#define rjurr13_a (*(Rjurr *)0xFFF69a)
#define rjurr14_a (*(Rjurr *)0xFFF69c)
#define rjurr15_a (*(Rjurr *)0xFFF6ae)
#define rjurr16_a (*(Rjurr *)0xFFF6a0)
#define rjurr17_a (*(Rjurr *)0xFFF6a2)
#define rjurr18_a (*(Rjurr *)0xFFF6a4)
#define rjurr19_a (*(Rjurr *)0xFFF6a6)
#define rjurr20_a (*(Rjurr *)0xFFF6a8)
#define rjurr21_a (*(Rjurr *)0xFFF6aa)
#define rjurr22_a (*(Rjurr *)0xFFF6ac)
#define rjurr23_a (*(Rjurr *)0xFFF6ae)
#define rjurr24_a (*(Rjurr *)0xFFF6b0)
#define rjurr25_a (*(Rjurr *)0xFFF6b2)
#define rjurr26_a (*(Rjurr *)0xFFF6b4)
#define rjurr27_a (*(Rjurr *)0xFFF6b6)
#define rjurr28_a (*(Rjurr *)0xFFF6b8)
#define rjurr29_a (*(Rjurr *)0xFFF6ba)
#define rjurr30_a (*(Rjurr *)0xFFF6bc)
#define rjurr31_a (*(Rjurr *)0xFFF6be)
```

```
#define rjurr32_a (*(Rjurr *)0xFFF6c0)
#define rjurr33_a (*(Rjurr *)0xFFF6c2)
#define rjurr34_a (*(Rjurr *)0xFFF6c4)
#define rjurr35_a (*(Rjurr *)0xFFF6c6)
#define rjurr36_a (*(Rjurr *)0xFFF6c8)
#define rjurr37_a (*(Rjurr *)0xFFF6ca)
#define rjurr38_a (*(Rjurr *)0xFFF6cc)
#define rjurr39_a (*(Rjurr *)0xFFF6ce)
#define rjurr40_a (*(Rjurr *)0xFFF6d0)
#define rjurr41_a (*(Rjurr *)0xFFF6d2)
#define rjurr42_a (*(Rjurr *)0xFFF6d4)
#define rjurr43_a (*(Rjurr *)0xFFF6d6)
#define rjurr44_a (*(Rjurr *)0xFFF6d8)
#define rjurr45_a (*(Rjurr *)0xFFF6da)
#define rjurr46_a (*(Rjurr *)0xFFF6dc)
#define rjurr47_a (*(Rjurr *)0xFFF6de)
#define rjurr48_a (*(Rjurr *)0xFFF6e0)

#define rjurr49_a (*(Rjurr *)0xFFF6e2)
#define rjurr50_a (*(Rjurr *)0xFFF6e4)
#define rjurr51_a (*(Rjurr *)0xFFF6e6)
#define rjurr52_a (*(Rjurr *)0xFFF6e8)
#define rjurr53_a (*(Rjurr *)0xFFF6ea)
#define rjurr54_a (*(Rjurr *)0xFFF6ec)
#define rjurr55_a (*(Rjurr *)0xFFF6ee)
#define rjurr56_a (*(Rjurr *)0xFFF6f0)
#define rjurr57_a (*(Rjurr *)0xFFF6f2)
#define rjurr58_a (*(Rjurr *)0xFFF6f4)
#define rjurr59_a (*(Rjurr *)0xFFF6f6)
#define rjurr60_a (*(Rjurr *)0xFFF6f8)
#define rjurr61_a (*(Rjurr *)0xFFF6fa)
#define rjurr62_a (*(Rjurr *)0xFFF6fc)
#define rjurr63_a (*(Rjurr *)0xFFF6fe)

#define rjurr0_b (*(Rjurr *)0xFFFa80)

#define rjurr1_b (*(Rjurr *)0xFFFa82)
#define rjurr2_b (*(Rjurr *)0xFFFa84)
#define rjurr3_b (*(Rjurr *)0xFFFa86)
#define rjurr4_b (*(Rjurr *)0xFFFa88)
#define rjurr5_b (*(Rjurr *)0xFFFa8a)
#define rjurr6_b (*(Rjurr *)0xFFFa8c)
#define rjurr7_b (*(Rjurr *)0xFFFa8e)
#define rjurr8_b (*(Rjurr *)0xFFFa90)
#define rjurr9_b (*(Rjurr *)0xFFFa92)
#define rjurr10_b (*(Rjurr *)0xFFFa94)
#define rjurr11_b (*(Rjurr *)0xFFFa96)
#define rjurr12_b (*(Rjurr *)0xFFFa98)
#define rjurr13_b (*(Rjurr *)0xFFFa9a)
#define rjurr14_b (*(Rjurr *)0xFFFa9c)
#define rjurr15_b (*(Rjurr *)0xFFFa9e)
```

```
#define rjurr16_b (*(Rjurr *)0xFFFaa0)
#define rjurr17_b (*(Rjurr *)0xFFFaa2)
#define rjurr18_b (*(Rjurr *)0xFFFaa4)
#define rjurr19_b (*(Rjurr *)0xFFFaa6)
#define rjurr20_b (*(Rjurr *)0xFFFaa8)
#define rjurr21_b (*(Rjurr *)0xFFFaaa)

#define rjurr22_b (*(Rjurr *)0xFFFaac)
#define rjurr23_b (*(Rjurr *)0xFFFaae)

#define rjurr24_b (*(Rjurr *)0xFFFab0)
#define rjurr25_b (*(Rjurr *)0xFFFab2)
#define rjurr26_b (*(Rjurr *)0xFFFab4)
#define rjurr27_b (*(Rjurr *)0xFFFab6)
#define rjurr28_b (*(Rjurr *)0xFFFab8)
#define rjurr29_b (*(Rjurr *)0xFFFaba)
#define rjurr30_b (*(Rjurr *)0xFFFabc)
#define rjurr31_b (*(Rjurr *)0xFFFabe)
#define rjurr32_b (*(Rjurr *)0xFFFac0)
#define rjurr33_b (*(Rjurr *)0xFFFac2)
#define rjurr34_b (*(Rjurr *)0xFFFac4)
#define rjurr35_b (*(Rjurr *)0xFFFac6)
#define rjurr36_b (*(Rjurr *)0xFFFac8)
#define rjurr37_b (*(Rjurr *)0xFFFaca)
#define rjurr38_b (*(Rjurr *)0xFFFacc)
#define rjurr39_b (*(Rjurr *)0xFFFace)
#define rjurr40_b (*(Rjurr *)0xFFFad0)
#define rjurr41_b (*(Rjurr *)0xFFFad2)
#define rjurr42_b (*(Rjurr *)0xFFFad4)
#define rjurr43_b (*(Rjurr *)0xFFFad6)
#define rjurr44_b (*(Rjurr *)0xFFFad8)
#define rjurr45_b (*(Rjurr *)0xFFFada)
#define rjurr46_b (*(Rjurr *)0xFFFadc)
#define rjurr47_b (*(Rjurr *)0xFFFade)
#define rjurr48_b (*(Rjurr *)0xFFFae0)
#define rjurr49_b (*(Rjurr *)0xFFFae2)
#define rjurr50_b (*(Rjurr *)0xFFFae4)
#define rjurr51_b (*(Rjurr *)0xFFFae6)
#define rjurr52_b (*(Rjurr *)0xFFFae8)
#define rjurr53_b (*(Rjurr *)0xFFFaea)
#define rjurr54_b (*(Rjurr *)0xFFFaec)
#define rjurr55_b (*(Rjurr *)0xFFFaee)
#define rjurr56_b (*(Rjurr *)0xFFFaf0)
#define rjurr57_b (*(Rjurr *)0xFFFaf2)
#define rjurr58_b (*(Rjurr *)0xFFFaf4)
#define rjurr59_b (*(Rjurr *)0xFFFaf6)
#define rjurr60_b (*(Rjurr *)0xFFFaf8)
#define rjurr61_b (*(Rjurr *)0xFFFafa)
#define rjurr62_b (*(Rjurr *)0xFFFafc)
#define rjurr63_b (*(Rjurr *)0xFFFaef)
```

```
/*****************************************************************************

* The QADC64 Left justified signed result register. pg. 13-48

*       Note: There are 64 words of these for each QADC.

*****************************************************************************/


typedef struct {
       unsigned s:                          1;
       unsigned result:      9;
       unsigned reserved0_5:                6;
       } Ljsrr;

#define ljsrr0_a (*(Ljsrr *)0xFFF700)
#define ljsrr1_a (*(Ljsrr *)0xFFF702)

#define ljsrr2_a (*(Ljsrr *)0xFFF704)
#define ljsrr3_a (*(Ljsrr *)0xFFF706)

#define ljsrr4_a (*(Ljsrr *)0xFFF708)
#define ljsrr5_a (*(Ljsrr *)0xFFF70a)
#define ljsrr6_a (*(Ljsrr *)0xFFF70c)
#define ljsrr7_a (*(Ljsrr *)0xFFF70e)
#define ljsrr8_a (*(Ljsrr *)0xFFF710)
#define ljsrr9_a (*(Ljsrr *)0xFFF712)
#define ljsrr10_a (*(Ljsrr *)0xFFF714)
#define ljsrr11_a (*(Ljsrr *)0xFFF716)
#define ljsrr12_a (*(Ljsrr *)0xFFF718)
#define ljsrr13_a (*(Ljsrr *)0xFFF71a)
#define ljsrr14_a (*(Ljsrr *)0xFFF71c)
#define ljsrr15_a (*(Ljsrr *)0xFFF71e)
#define ljsrr16_a (*(Ljsrr *)0xFFF720)
#define ljsrr17_a (*(Ljsrr *)0xFFF722)
#define ljsrr18_a (*(Ljsrr *)0xFFF724)
#define ljsrr19_a (*(Ljsrr *)0xFFF726)
#define ljsrr20_a (*(Ljsrr *)0xFFF728)
#define ljsrr21_a (*(Ljsrr *)0xFFF72a)

#define ljsrr22_a (*(Ljsrr *)0xFFF72c)
#define ljsrr23_a (*(Ljsrr *)0xFFF72e)
#define ljsrr24_a (*(Ljsrr *)0xFFF730)
#define ljsrr25_a (*(Ljsrr *)0xFFF732)

#define ljsrr26_a (*(Ljsrr *)0xFFF734)
#define ljsrr27_a (*(Ljsrr *)0xFFF736)
#define ljsrr28_a (*(Ljsrr *)0xFFF738)
#define ljsrr29_a (*(Ljsrr *)0xFFF73a)
#define ljsrr30_a (*(Ljsrr *)0xFFF73c)
```

```
#define ljsrr31_a (*(Ljsrr *)0xFFF73e)
#define ljsrr32_a (*(Ljsrr *)0xFFF740)
#define ljsrr33_a (*(Ljsrr *)0xFFF742)
#define ljsrr34_a (*(Ljsrr *)0xFFF744)
#define ljsrr35_a (*(Ljsrr *)0xFFF746)
#define ljsrr36_a (*(Ljsrr *)0xFFF748)
#define ljsrr37_a (*(Ljsrr *)0xFFF74a)
#define ljsrr38_a (*(Ljsrr *)0xFFF74c)
#define ljsrr39_a (*(Ljsrr *)0xFFF74e)
#define ljsrr40_a (*(Ljsrr *)0xFFF750)
#define ljsrr41_a (*(Ljsrr *)0xFFF752)
#define ljsrr42_a (*(Ljsrr *)0xFFF754)
#define ljsrr43_a (*(Ljsrr *)0xFFF756)
#define ljsrr44_a (*(Ljsrr *)0xFFF758)
#define ljsrr45_a (*(Ljsrr *)0xFFF75a)
#define ljsrr46_a (*(Ljsrr *)0xFFF75c)
#define ljsrr47_a (*(Ljsrr *)0xFFF75e)
#define ljsrr48_a (*(Ljsrr *)0xFFF760)
#define ljsrr49_a (*(Ljsrr *)0xFFF762)
#define ljsrr50_a (*(Ljsrr *)0xFFF764)
#define ljsrr51_a (*(Ljsrr *)0xFFF766)
#define ljsrr52_a (*(Ljsrr *)0xFFF768)
#define ljsrr53_a (*(Ljsrr *)0xFFF76a)
#define ljsrr54_a (*(Ljsrr *)0xFFF76c)
#define ljsrr55_a (*(Ljsrr *)0xFFF76e)
#define ljsrr56_a (*(Ljsrr *)0xFFF770)
#define ljsrr57_a (*(Ljsrr *)0xFFF772)
#define ljsrr58_a (*(Ljsrr *)0xFFF774)
#define ljsrr59_a (*(Ljsrr *)0xFFF776)
#define ljsrr60_a (*(Ljsrr *)0xFFF778)
#define ljsrr61_a (*(Ljsrr *)0xFFF77a)
#define ljsrr62_a (*(Ljsrr *)0xFFF77c)
#define ljsrr63_a (*(Ljsrr *)0xFFF77e)
#define ljsrr0_b (*(Ljsrr *)0xFFFb00)
#define ljsrr1_b (*(Ljsrr *)0xFFFb02)
#define ljsrr2_b (*(Ljsrr *)0xFFFb04)
#define ljsrr3_b (*(Ljsrr *)0xFFFb06)
#define ljsrr4_b (*(Ljsrr *)0xFFFb08)
#define ljsrr5_b (*(Ljsrr *)0xFFFb0a)

#define ljsrr6_b (*(Ljsrr *)0xFFFb0c)
#define ljsrr7_b (*(Ljsrr *)0xFFFb0e)
#define ljsrr8_b (*(Ljsrr *)0xFFFb10)
#define ljsrr9_b (*(Ljsrr *)0xFFFb12)
#define ljsrr10_b (*(Ljsrr *)0xFFFb14)
#define ljsrr11_b (*(Ljsrr *)0xFFFb16)
#define ljsrr12_b (*(Ljsrr *)0xFFFb18)
#define ljsrr13_b (*(Ljsrr *)0xFFFb1a)
#define ljsrr14_b (*(Ljsrr *)0xFFFb1c)
#define ljsrr15_b (*(Ljsrr *)0xFFFb1e)
#define ljsrr16_b (*(Ljsrr *)0xFFFb20)
```

```
#define ljsrr17_b (*(Ljsrr *)0xFFFb22)
#define ljsrr18_b (*(Ljsrr *)0xFFFb24)
#define ljsrr19_b (*(Ljsrr *)0xFFFb26)
#define ljsrr20_b (*(Ljsrr *)0xFFFb28)
#define ljsrr21_b (*(Ljsrr *)0xFFFb2a)
#define ljsrr22_b (*(Ljsrr *)0xFFFb2c)
#define ljsrr23_b (*(Ljsrr *)0xFFFb2e)
#define ljsrr24_b (*(Ljsrr *)0xFFFb30)

#define ljsrr25_b (*(Ljsrr *)0xFFFb32)
#define ljsrr26_b (*(Ljsrr *)0xFFFb34)
#define ljsrr27_b (*(Ljsrr *)0xFFFb36)
#define ljsrr28_b (*(Ljsrr *)0xFFFb38)
#define ljsrr29_b (*(Ljsrr *)0xFFFb3a)
#define ljsrr30_b (*(Ljsrr *)0xFFFb3c)
#define ljsrr31_b (*(Ljsrr *)0xFFFb3e)
#define ljsrr32_b (*(Ljsrr *)0xFFFb40)
#define ljsrr33_b (*(Ljsrr *)0xFFFb42)
#define ljsrr34_b (*(Ljsrr *)0xFFFb44)
#define ljsrr35_b (*(Ljsrr *)0xFFFb46)
#define ljsrr36_b (*(Ljsrr *)0xFFFb48)
#define ljsrr37_b (*(Ljsrr *)0xFFFb4a)
#define ljsrr38_b (*(Ljsrr *)0xFFFb4c)
#define ljsrr39_b (*(Ljsrr *)0xFFFb4e)

#define ljsrr40_b (*(Ljsrr *)0xFFFb50)
#define ljsrr41_b (*(Ljsrr *)0xFFFb52)
#define ljsrr42_b (*(Ljsrr *)0xFFFb54)
#define ljsrr43_b (*(Ljsrr *)0xFFFb56)
#define ljsrr44_b (*(Ljsrr *)0xFFFb58)
#define ljsrr45_b (*(Ljsrr *)0xFFFb5a)

#define ljsrr46_b (*(Ljsrr *)0xFFFb5c)
#define ljsrr47_b (*(Ljsrr *)0xFFFb5e)
#define ljsrr48_b (*(Ljsrr *)0xFFFb60)
#define ljsrr49_b (*(Ljsrr *)0xFFFb62)
#define ljsrr50_b (*(Ljsrr *)0xFFFb64)
#define ljsrr51_b (*(Ljsrr *)0xFFFb66)
#define ljsrr52_b (*(Ljsrr *)0xFFFb68)
#define ljsrr53_b (*(Ljsrr *)0xFFFb6a)
#define ljsrr54_b (*(Ljsrr *)0xFFFb6c)
#define ljsrr55_b (*(Ljsrr *)0xFFFb6e)
#define ljsrr56_b (*(Ljsrr *)0xFFFb70)
#define ljsrr57_b (*(Ljsrr *)0xFFFb72)
#define ljsrr58_b (*(Ljsrr *)0xFFFb74)
#define ljsrr59_b (*(Ljsrr *)0xFFFb76)
#define ljsrr60_b (*(Ljsrr *)0xFFFb78)
#define ljsrr61_b (*(Ljsrr *)0xFFFb7a)
#define ljsrr62_b (*(Ljsrr *)0xFFFb7c)
#define ljsrr63_b (*(Ljsrr *)0xFFFb7e)
```

```
/*****************************************************************************

* The QADC64 Left justified unsigned result register. pg. 13-49

*       Note: There are 64 words of these for each QADC.

*****************************************************************************/
typedef struct {
       unsigned result:           10;
       unsigned reserved0_5:       6;
       } Ljurr;
#define ljurr0_a (*(Ljurr *)0xFFF780)
#define ljurr1_a (*(Ljurr *)0xFFF782)
#define ljurr2_a (*(Ljurr *)0xFFF784)
#define ljurr3_a (*(Ljurr *)0xFFF786)
#define ljurr4_a (*(Ljurr *)0xFFF788)
#define ljurr5_a (*(Ljurr *)0xFFF78a)
#define ljurr6_a (*(Ljurr *)0xFFF78c)
#define ljurr7_a (*(Ljurr *)0xFFF78e)
#define ljurr8_a (*(Ljurr *)0xFFF790)
#define ljurr9_a (*(Ljurr *)0xFFF792)
#define ljurr10_a (*(Ljurr *)0xFFF794)
#define ljurr11_a (*(Ljurr *)0xFFF796)
#define ljurr12_a (*(Ljurr *)0xFFF798)
#define ljurr13_a (*(Ljurr *)0xFFF79a)

#define ljurr14_a (*(Ljurr *)0xFFF79c)
#define ljurr15_a (*(Ljurr *)0xFFF7ae)
#define ljurr16_a (*(Ljurr *)0xFFF7a0)
#define ljurr17_a (*(Ljurr *)0xFFF7a2)
#define ljurr18_a (*(Ljurr *)0xFFF7a4)
#define ljurr19_a (*(Ljurr *)0xFFF7a6)
#define ljurr20_a (*(Ljurr *)0xFFF7a8)
#define ljurr21_a (*(Ljurr *)0xFFF7aa)
#define ljurr22_a (*(Ljurr *)0xFFF7ac)
#define ljurr23_a (*(Ljurr *)0xFFF7ae)
#define ljurr24_a (*(Ljurr *)0xFFF7b0)
#define ljurr25_a (*(Ljurr *)0xFFF7b2)
#define ljurr26_a (*(Ljurr *)0xFFF7b4)
#define ljurr27_a (*(Ljurr *)0xFFF7b6)
#define ljurr28_a (*(Ljurr *)0xFFF7b8)
#define ljurr29_a (*(Ljurr *)0xFFF7ba)
#define ljurr30_a (*(Ljurr *)0xFFF7bc)
#define ljurr31_a (*(Ljurr *)0xFFF7be)
#define ljurr32_a (*(Ljurr *)0xFFF7c0)
#define ljurr33_a (*(Ljurr *)0xFFF7c2)

#define ljurr34_a (*(Ljurr *)0xFFF7c4)
#define ljurr35_a (*(Ljurr *)0xFFF7c6)
#define ljurr36_a (*(Ljurr *)0xFFF7c8)
#define ljurr37_a (*(Ljurr *)0xFFF7ca)
```

```
#define ljurr38_a (*(Ljurr *)0xFFF7cc)
#define ljurr39_a (*(Ljurr *)0xFFF7ce)
#define ljurr40_a (*(Ljurr *)0xFFF7d0)
#define ljurr41_a (*(Ljurr *)0xFFF7d2)
#define ljurr42_a (*(Ljurr *)0xFFF7d4)
#define ljurr43_a (*(Ljurr *)0xFFF7d6)
#define ljurr44_a (*(Ljurr *)0xFFF7d8)
#define ljurr45_a (*(Ljurr *)0xFFF7da)

#define ljurr46_a (*(Ljurr *)0xFFF7dc)
#define ljurr47_a (*(Ljurr *)0xFFF7de)
#define ljurr48_a (*(Ljurr *)0xFFF7e0)
#define ljurr49_a (*(Ljurr *)0xFFF7e2)
#define ljurr50_a (*(Ljurr *)0xFFF7e4)
#define ljurr51_a (*(Ljurr *)0xFFF7e6)
#define ljurr52_a (*(Ljurr *)0xFFF7e8)
#define ljurr53_a (*(Ljurr *)0xFFF7ea)
#define ljurr54_a (*(Ljurr *)0xFFF7ec)
#define ljurr55_a (*(Ljurr *)0xFFF7ee)
#define ljurr56_a (*(Ljurr *)0xFFF7f0)
#define jjurr57_a (*(Ljurr *)0xFFF7f2)
#define ljurr58_a (*(Ljurr *)0xFFF7f4)
#define ljurr59_a (*(Ljurr *)0xFFF7f6)
#define ljurr60_a (*(Ljurr *)0xFFF7f8)
#define ljurr61_a (*(Ljurr *)0xFFF7fa)
#define ljurr62_a (*(Ljurr *)0xFFF7fc)
#define ljurr63_a (*(Ljurr *)0xFFF7fe)
#define ljurr0_b (*(Ljurr *)0xFFFb80)
#define ljurr1_b (*(Ljurr *)0xFFFb82)
#define ljurr2_b (*(Ljurr *)0xFFFb84)
#define ljurr3_b (*(Ljurr *)0xFFFb86)
#define ljurr4_b (*(Ljurr *)0xFFFb88)
#define ljurr5_b (*(Ljurr *)0xFFFb8a)
#define ljurr6_b (*(Ljurr *)0xFFFb8c)
#define ljurr7_b (*(Ljurr *)0xFFFb8e)
#define ljurr8_b (*(Ljurr *)0xFFFb90)
#define ljurr9_b (*(Ljurr *)0xFFFb92)
#define ljurr10_b (*(Ljurr *)0xFFFb94)
#define ljurr11_b (*(Ljurr *)0xFFFb96)
#define ljurr12_b (*(Ljurr *)0xFFFb98)
#define ljurr13_b (*(Ljurr *)0xFFFb9a)
#define ljurr14_b (*(Ljurr *)0xFFFb9c)
#define ljurr15_b (*(Ljurr *)0xFFFb9e)
#define ljurr16_b (*(Ljurr *)0xFFFba0)
#define ljurr17_b (*(Ljurr *)0xFFFba2)
#define ljurr18_b (*(Ljurr *)0xFFFba4)
#define ljurr19_b (*(Ljurr *)0xFFFba6)
#define ljurr20_b (*(Ljurr *)0xFFFba8)
#define ljurr21_b (*(Ljurr *)0xFFFbaa)
#define ljurr22_b (*(Ljurr *)0xFFFbac)
#define ljurr23_b (*(Ljurr *)0xFFFbae)
```

```
#define ljurr24_b (*(Ljurr *)0xFFFbb0)
#define ljurr25_b (*(Ljurr *)0xFFFbb2)
#define ljurr26_b (*(Ljurr *)0xFFFbb4)
#define ljurr27_b (*(Ljurr *)0xFFFbb6)
#define ljurr28_b (*(Ljurr *)0xFFFbb8)
#define ljurr29_b (*(Ljurr *)0xFFFbba)
#define ljurr30_b (*(Ljurr *)0xFFFbbc)
#define ljurr31_b (*(Ljurr *)0xFFFbbe)
#define ljurr32_b (*(Ljurr *)0xFFFbc0)
#define ljurr33_b (*(Ljurr *)0xFFFbc2)

#define ljurr34_b (*(Ljurr *)0xFFFbc4)
#define ljurr35_b (*(Ljurr *)0xFFFbc6)
#define ljurr36_b (*(Ljurr *)0xFFFbc8)
#define ljurr37_b (*(Ljurr *)0xFFFbca)
#define ljurr38_b (*(Ljurr *)0xFFFbcc)
#define ljurr39_b (*(Ljurr *)0xFFFbce)
#define ljurr40_b (*(Ljurr *)0xFFFbd0)
#define ljurr41_b (*(Ljurr *)0xFFFbd2)
#define ljurr42_b (*(Ljurr *)0xFFFbd4)
#define ljurr43_b (*(Ljurr *)0xFFFbd6)
#define ljurr44_b (*(Ljurr *)0xFFFbd8)
#define ljurr45_b (*(Ljurr *)0xFFFbda)
#define ljurr46_b (*(Ljurr *)0xFFFbdc)
#define ljurr47_b (*(Ljurr *)0xFFFbde)
#define ljurr48_b (*(Ljurr *)0xFFFbe0)
#define ljurr49_b (*(Ljurr *)0xFFFbe2)
#define ljurr50_b (*(Ljurr *)0xFFFbe4)
#define ljurr51_b (*(Ljurr *)0xFFFbe6)
#define ljurr52_b (*(Ljurr *)0xFFFbe8)
#define ljurr53_b (*(Ljurr *)0xFFFbea)
#define ljurr54_b (*(Ljurr *)0xFFFbec)
#define ljurr55_b (*(Ljurr *)0xFFFbee)
#define ljurr56_b (*(Ljurr *)0xFFFbf0)
#define ljurr57_b (*(Ljurr *)0xFFFbf2)
#define ljurr58_b (*(Ljurr *)0xFFFbf4)
#define ljurr59_b (*(Ljurr *)0xFFFbf6)
#define ljurr60_b (*(Ljurr *)0xFFFbf8)
#define ljurr61_b (*(Ljurr *)0xFFFbfa)
#define ljurr62_b (*(Ljurr *)0xFFFbfc)
#define ljurr63_b (*(Ljurr *)0xFFFbfe)

#endif
```

Measuring and Analyzing QADC64 Conversion Signals

The QADC64 module is designed with 16 analog input pins. The 16 channel/port pins can support up to 41 channels when external multiplexing is used. All of the channel pins also can be used as general-purpose digital port pins.

In non-multiplexed mode (internal multiplexing), the 16 channel pins are connected to an internal multiplexer which sends the analog signals into the A/D converter.

In externally multiplexed mode, the QADC64 allows channel selection up to four external multiplexer chips to allow selection of one of eight input pins. This maximizes the amount of analog inputs handled by the QADC (Refer to Figure 2-5 in the cerberus 68F375 overview specification at http://www.mcu.motsps.com/lit/.)

Connecting External Analog Inputs and Measuring Digital Output Voltages

1. Locate input pin array W6 numbered 1-16 on MPBF375BGA daughter board as shown in **Figure 4**.

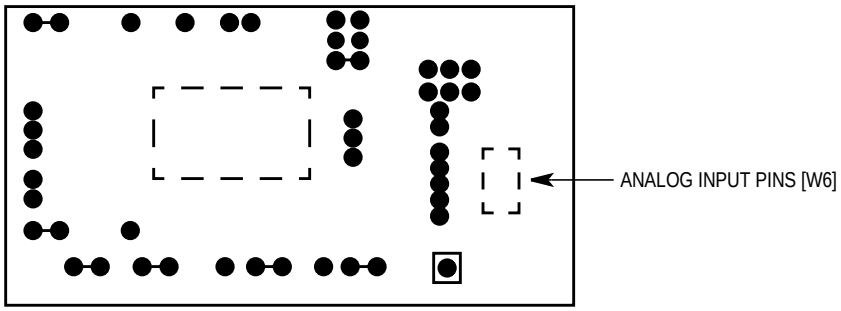


Figure 4. MPBF375BGA Daughter Board

2. Connect analog input to desired pin number on W6 as outlined in **Table 1** and **Table 2**.

Table 1 refers to pin assignments when using the QADC in non-multiplexed mode. In this mode, the QADC is still internally muxed. **Table 2** shows the availability of more inputs with external channel multiplexing.

3. Measure corresponding digital voltages on corresponding PQAx and PQBx output pins using a multimeter.

Table 1. QADC Pin Assignments in Non-Multiplexed Mode

Non-Multiplexed Input Pins, Internally Muxed					Channel Number in CHAN	
Output Port Name	**Input Analog Pin Name**	**Input Pin # on W6**	**Other Functions**	**Pin Type**	**Binary**	**Decimal**
PQB0	AN0	1	—	Input	000000	0
PQB1	AN1	2	—	Input	000001	1
PQB2	AN2	3	—	Input	000010	2
PQB3	AN3	4	—	Input	000011	3
—	—	—	Invalid	—	0000100 to 011111	4 to 31
—	—	—	Reserved	—	10XXXX	30 to 47
PQB4	AN48	5	—	Input	110000	48
PQB5	AN49	6	—	Input	110001	49
PQB6	AN50	7	—	Input	110010	50
PQB7	AN51	8	—	Input	110011	51
PQA0	AN52	9	—	Input/output	110100	52
PQA1	AN53	10	—	Input/output	110101	53
PQA2	AN54	11	—	Input/output	110110	54
PQA3	AN55	12	—	Input/output	110111	55
PQA4	AN56	13	—	Input/output	111000	56
PQA5	AN57	14	—	Input/output	111001	57
PQA6	AN58	15	—	Input/output	111010	58
PQA7	AN59	16	—	Input/output	111011	59
—	V_{RL}	—	—	Input	111100	60
—	V_{RH}	—	—	Input	111101	61
—	—	—	$(V_{RH}–V_{RL})$ /2	—	111110	62

Table 2. Input Availability with External Channel Multiplexing

Multiplexed Input Pins, Externally Muxed					Channel Number in CHAN	
Output Port Name	**Input Analog Pin Name**	**Input Pin # on W6**	**Other Functions**	**Pin Type**	**Binary**	**Decimal**
PQB0	Anw	1	—	Input	000000	0 to 14 even
PQB1	Anx	2	—	Input	000001	1 to 15 odd
PQB2	Any	3	—	Input	000010	16 to 30 even
PQB3	Anz	4	—	Input	000011	17 to 31 odd
—	—	—	Reserved	—	0000100 to 011111	32 to 47
PQB4	AN48	5	—	Input	110000	48
PQB5	AN49	6	—	Input	110001	49
PQB6	AN50	7	—	Input	110010	50
PQB7	AN51	8	—	Input	110011	51
PQA0	—	9	MA0	Input/output	110100	52
PQA1	—	10	MA1	Input/output	110101	53
PQA2	—	11	MA2	Input/output	110110	54
PQA3	AN55	12	—	Input/output	110111	55
PQA4	AN56	13	—	Input/output	111000	56
PQA5	AN57	14	—	Input/output	111001	57
PQA6	AN58	15	—	Input/output	111010	58
PQA7	AN59	16	—	Input/output	111011	59
—	V_{RL}	—	—	Input	111100	60
—	V_{RH}	—	—	Input	111101	61
—	—	—	$(V_{RH}–V_{RL})$ /2	—	111110	62
—	—	—	End of queue code	—	111111	63

NOTE: *The QADC64 internal analog multiplexer allows the 16 channel/port pins to support up to 41 channels when external multiplexing is used (including internal channels). See the QADC64 functional specification at http://www.mcu.motsps.com/lit/ for details.*

Analyzing Module Conversions

Knowing what to expect in the result word table (starting at location 0XFFF680) can be helpful for debugging.

The next formula calculates the value of a digital result following a QADC64 10-bit conversion:

$$\text{Result} = 2^{10} \times V_A \div V_{RH} - V_{RL}$$

Example: An analog input of 2.5 volts will be observed as:

$$2^{10} \times 2.5 \div 5\ V - 0\ V = 512 = 200 \text{ hex at location } 0XFFF680$$

Under normal operating conditions, the total unadjusted error (TUE) result count should be less than or equal to 2.

Acceptable result values for a 2.5-V analog input will be $1FE, $1FF, and $200.

For a 5.0-V input, acceptable results will be $3FE, $3FF, and $400.

Figure 5 shows a typical linearity plot at:

2 MHz ÷ 5.0 V (typical voltage) ÷ 25°C (room temperature)

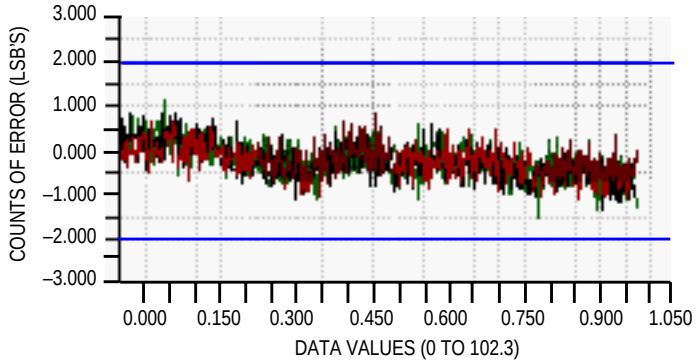


Figure 5. Typical Linearity Plot

Current Injection and Conversion

Design specifications require allowable input current of ±3 mA to the QADC64 during conversions. Experimental results show that input currents greater than ±5 mA induce TUE changes of more than 2 counts. To minimize linearity errors, input signal conditioning circuits are recommended. The next section provides experimental details on useful conditioning circuits.

Practical Automotive Applications

Analog input to the QADC64 from typical engine sensors and controllers introduces input impedance.

An experiment measured resistive characteristics of two common sensors: A coolant temperature sensor and a throttle position sensor. The results are outlined here.

Parts:

- TPS225 throttle position sensor, Wells automotive part
- SU201 coolant temperature sensor, Wells automotive part

Figure 6 shows a plot of resistance against temperature changes for the coolant temperature sensor. **Figure 7** shows a plot of resistance versus throttle position for the throttle position sensor.

These values are useful in determining the Thevenin resistance to the analog QADC64 input while designing additional input filters.

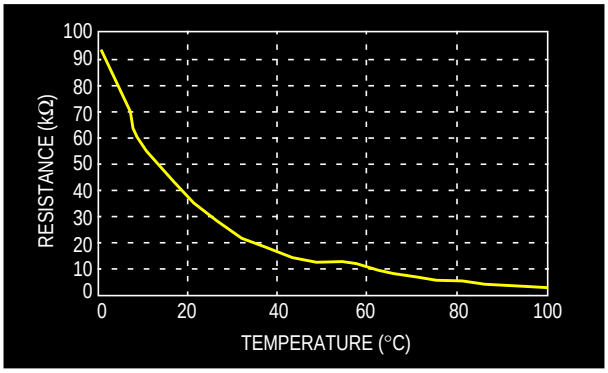


Figure 6. Wells SU201 Impedance Results

Table 3. Engine Coolant Resistance Value Variations with Temperature

Temperature in °C	Resistance in kΩ
100	2.60
95	3.26
90	3.70
85	4.33
79	5.20
75	5.90
70	7.07
65	8.32
60	10.33
55	12.41
50	11.72
48	12.40
45	13.49
42	14.74
40	16.52
38	17.38
35	19.70
31	22.70
28	25.95
24	30.40
17	41.50
15	46.90
11	53.50
8	63.40
7	70.70
1	92.50

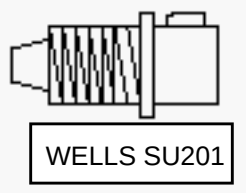
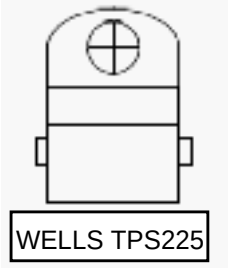
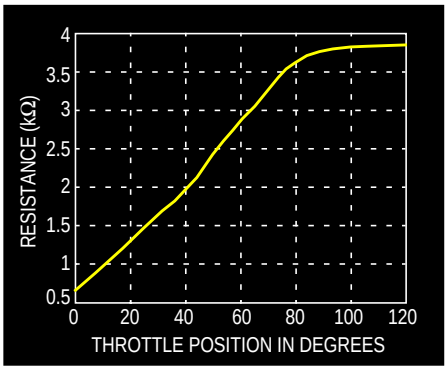


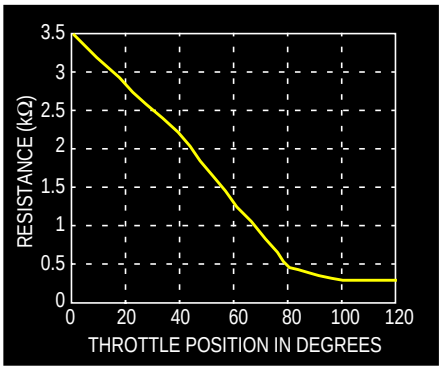
Table 4. Throttle Position Resistance Value Variations

Throttle Position in Degrees	Resistance R1 in kΩ	Resistance R2 in kΩ
0	0.756	3.452
20	1.394	2.848
40	2.048	2.196
60	2.917	1.339
80	3.704	0.517
100	3.850	0.338
120	3.863	0.337





GRAPH OF R1 AGAINST POSITION IN DEGREES



GRAPH OF R2 AGAINST POSITION IN DEGREES

Figure 7. Wells TPS225 Impedance Results

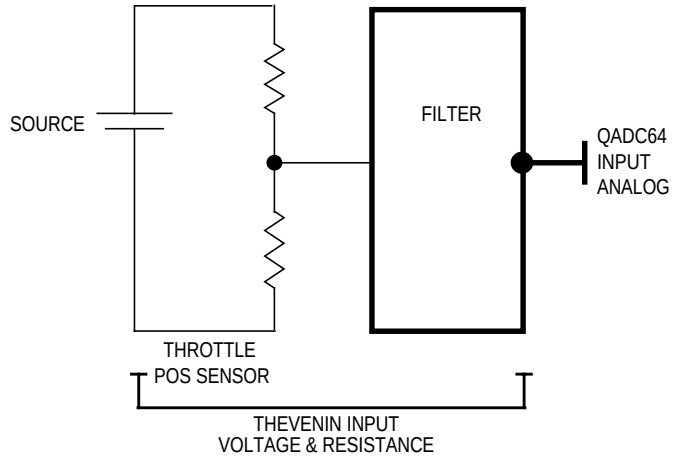


Figure 8. Resistance versus Throttle Position for the Throttle Position Sensor

Input Signal Conditioning Circuit Analysis

Input signal conditioning circuits serve three main purposes for QADC64 applications:

1. Effective protection against electrostatic discharge spikes
2. Filtering noise to the QADC64 inputs
3. Limiting current injection during conversion, minimizing linearity errors

The degree of protection that the QADC64 lines require against spikes and excessive voltage depends on the application. The basic method is to use Zener diodes for voltage clamping and series resistors for current limiting. Capacitors are also used to form an RC (resistance capacitor) filter that minimizes RF voltages. The RC filter time constant should be set for the lowest frequency possible that will still allow the application to work as planned.

The next example circuit will minimize input current and voltage to the QADC, which may otherwise offset the module conversion performance.

Example: Assuming an automotive V battery input signal has a DC value of 16 V on, which is superimposed, a 100-kHz sinusoid of 1-V peak amplitude. The circuit analysis using **Figure 9** shows experimental results of input signal values to the QADC module.

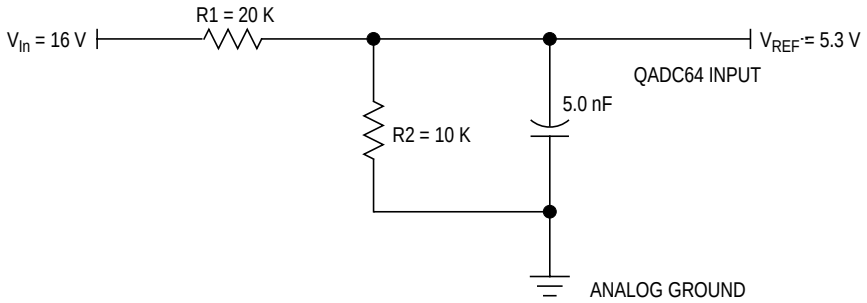


Figure 9. Conditioning a 16-Volt Input

Low pass filter analysis:

$$1 \div \omega o = 5.0\ nF \bullet (200\ K \div 30\ K) = 33.3\ exp^{-6}$$

$$\omega o = 30.0\ exp^{3}$$

$$f = 30.0\ exp^{3} \div 2\Pi = 4.8\ kHz$$

Circuit will filter out all frequencies above this.

Input voltage to the QADC:

$$R1||R2 = (10\ K \div 10\ K) + 20\ K = 0.33\ k\Omega$$

$$V_{Ref} = 0.33 \bullet 16\ V = 5.3\ V$$

Current through the capacitor =

$$C \bullet dv \div dt = 5\ nf \bullet d \div dt\ [\sin(6.284 * 100\ kHz)] = -4\ nA$$

Result Analysis

For effective operation over full temperature and frequency range, the disruptive input injection current to the QADC64 should be a minimum of –3 mA and a maximum of +3 mA. The analog supply with reference to V_{SSA} minimum should be –0.3 V and a maximum of 5.5 V. Exceeding

these limits may cause conversion errors while transitions within the limits do not affect device reliability or cause permanent damage.

The resulting reference voltage of 5.3 V falls within an acceptable analog input voltage limit.

Capacitance Selection Criteria

For practical input signal conditioning circuits, to ensure a maximum sampling error of the input ≥ 1 LSB (least significant bit), the external filter capacitor, C_f, must be ≥ 1024 • C_{samp} where $C_{samp} \approx 5$ pF.

In the recommended example circuit, $C_f = 5$ nF, which is = 1024 • C_{samp}.

Ordinary, aluminum, electrolytic capacitors are most often used for power-supply filtering and bypassing. It is critical to choose a filter capacitor whose effective series impedance is low at all rated temperatures and frequencies. Otherwise, the RMS (root mean square) filter current multiplied by the resistive component of the series impedance can cause excessive self-heating. And, if the heat cannot flow out of the capacitor, the temperature rises, causing early failure.

Standard ceramic capacitors are useful for filter circuits. They have a typical ESR (electro static ratio) of 0.1 Ω or lower. So when a digital IC tries to draw a 50 mA surge of current for a couple of nanoseconds, the low ESR helps to prevent spikes on the power supply bus.

The circuit in **Figure 10** yields a low pass cut off frequency of 1.59 kHz.

NOTE: *Maximum leakage occurs at maximum operating temperature. Current decreases by approximately one-half for each 8 to 12 °C in the ambient temperature range of 50 to 125 °C.*

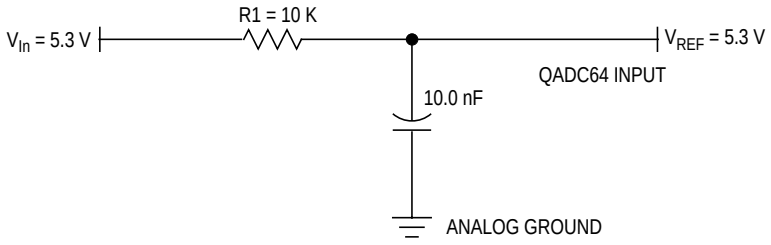


Figure 10. Recommended RC Filter Circuit Example

Protection Against Voltage Overloads

The simple RC network will filter the QADC64 analog input and allow for normal overload protection, for example, input shorted to the battery. However, this does not provide enough protection for overloads such as voltage spikes (momentary pulses up to 400 V) or for load dumps (momentary pulses of –40 V) seen in an automotive environment. Although the QADC64 design includes internal input diodes, additional protection may yield better conversion and performance results.

This overload protection can be realized by connecting two Schottky diodes and a Zener diode in reverse bias at the QADC64 input (shown in **Figure 11**.)

Schottky diodes have a smaller forward voltage (V_{fs}) than ordinary pn diodes, making them suitable for good low leakage characteristics in the reverse biased state. Zener diodes are useful for voltage clamping and are designed to break down in a predictable and well-behaved way. Thus, combining them, as shown here, blocks current while allowing negligible leakage into the QADC64.

For optimal low leakage, use a transistor's collector-base junction instead of a discrete diode. The popular 2N930 and 2N3707 have leakage less than 1 pA, even at as much as 7 V and 10 pA at 50 V. Other sources of ultra-low-leakage diodes are the 2N4117A and the PN4117A diodes. These devices are JFETS with very small junctions, so leakage well below 0.1 pA is standard with 1.0 pA max guaranteed.

NOTE: *It is recommended that users fully evaluate any prototype board at its full temperature range to gain insight into additional problems.*

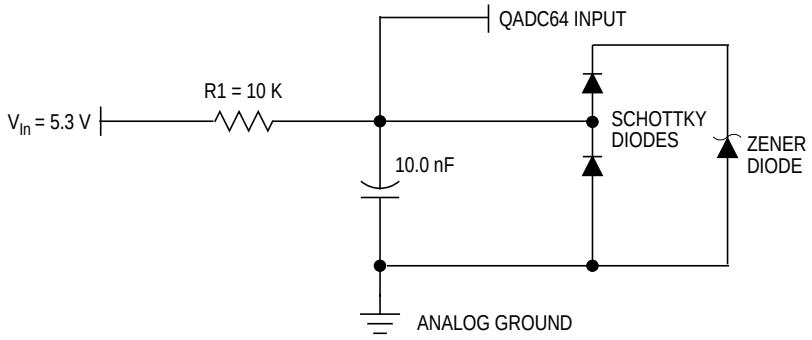


Figure 11. Overload Protection Using a Schottky and Zener Diode

Overload protection also can be accomplished using available packaged devices. The Motorola 5.6-volt SC-59 quad monolithic common anode device is a good example of such useful ESD (electro static discharge) overload protection devices. It makes effective use of diodes to limit input current leakage. It is also especially ideal for situations where board space is at a premium. (Motorola's preferred device is the MMQA5V6T1.)

MMQA5V6T1 Specification Features

- **SC-59 package allows four separate unidirectional configurations**
- **Peak power –24 Watts @ 1.0 ms, unidirectional**
- **Maximum clamping voltage @ peak pulse current**
- **Low current leakage < 2.0 μA**
- **ESD rating of class N, exceeding 15 kV, per the human body model**

CASE 318F-01
STYLE 1
SC-59 PLASTIC

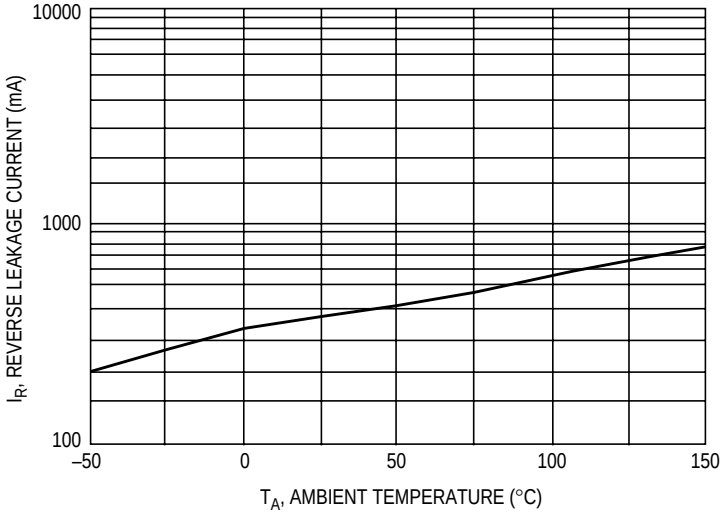


Figure 12. MMQA5V6T1 Typical Leakage Current versus Temperature Plot

A Software Engineering Approach to QADC64 Signal Analysis

The following software was developed for use as an exercise tool for the QADC64 module. It sends digital samples of data points, which represent varying input voltage levels to the parallel port.

These experimental data values can be sent simultaneously to a digital-to-analog converter using the test circuit provided in **Figure 13**. The resulting filtered analog output can be fed to the QADC64 analog input pin (AN*X*). The developer then can check converted digital values in the QADC64 result word address location using the SDS debug tool. Refer to **Connecting External Analog Inputs and Measuring Digital Output Voltages** in this application note.

Results in the result word table provide a quick assessment of the QADC64 linearity yields.

```
//This code can be easily modified and recompiled to provide custom data.
//It was developed by:

//Austin Ehi. Obobaifo
//Engineering Rotation Program

//Second Rotation @ PowerTrain Systems Division Motorola Inc.
//Nov 13 1998.

#include <stdio.h>                          //Required Libraries
#include <stdlib.h>
#include <conio.h>
#include <string.h>
#include <iostream.h>
#include <string.h>
#include <dos.h>
#include <math.h>
#include <windows.h>
#include <time.h>
#include <sys/types.h>
#include <sys\timeb.h>
#include <string.h>
#define port 0x378
void Delay(long option, int length);

void main()

{

//Declaring Variables required
```

```
int i=0;                                    //increment variable

double xpi=6.2832;                          //2*pi

double ampl=5.0;                            //Required waveform Amplitude

FILE *newpulse;                             //Desired output file pointer

double yaxis[1024];                         //array for 10 data bit resolution values


do            //Calculate values and continuously output to the
{             //Parallel port until keyboard hit


for(i=0; i<1023;i++)                        //All 1024 Data points
{
       yaxis[i]=(ampl*(cos((xpi*60)*i)));   //Developer can specify function here

       printf("%d \n",yaxis[i]);            //display check for correct values!

       _outp(port + 0,int(yaxis[i]));       //send each sample byte to data port
0x378

                                            //_outp(port + 0,int(yaxis[256]));
                            //Alternatively, send a specific voltage at a time

       Delay(1,10000);                      //Interval Sampling delay time
                                            //Using System clock

}

}


while(!kbhit());                            //End Do loop

       newpulse = fopen("data.txt","w");


if (!newpulse)
       perror("Unable to open file for writing");  //Error checking

else
       newpulse = fopen("data.txt","w");    //Open output file
       for (i=0; i<1023; i++)
       {
                                            //Write data sample/s to output file
       fprintf(newpulse,"%f \n",yaxis[i])   //for verification of correct values

       }

       fclose(newpulse);                    //Close output file return;
}
```

```
void Delay(long option, int length)         //Useful system delay subroutine

{
    static _int64 update_ticks_p1ms;
    _int64 start, end, timeOut, freq, update_ticks = length * update_ticks_p1ms;



    switch(option)                          // Process Delay option
    {
        case 0:                             // initialize delay
            if (!QueryPerformanceFrequency((LARGE_INTEGER*)&freq));
            update_ticks_p1ms = freq/10000;
            break;
        default:                            // perform delay
            QueryPerformanceCounter((LARGE_INTEGER*)&start);
            timeOut = start + update_ticks;
            QueryPerformanceCounter((LARGE_INTEGER*)&end);
            while (end < timeOut)
            QueryPerformanceCounter((LARGE_INTEGER*)&end);
            break;
    }
}                                           //END
```

The block diagram in **Figure 13** outlines the hardware and software design flow.

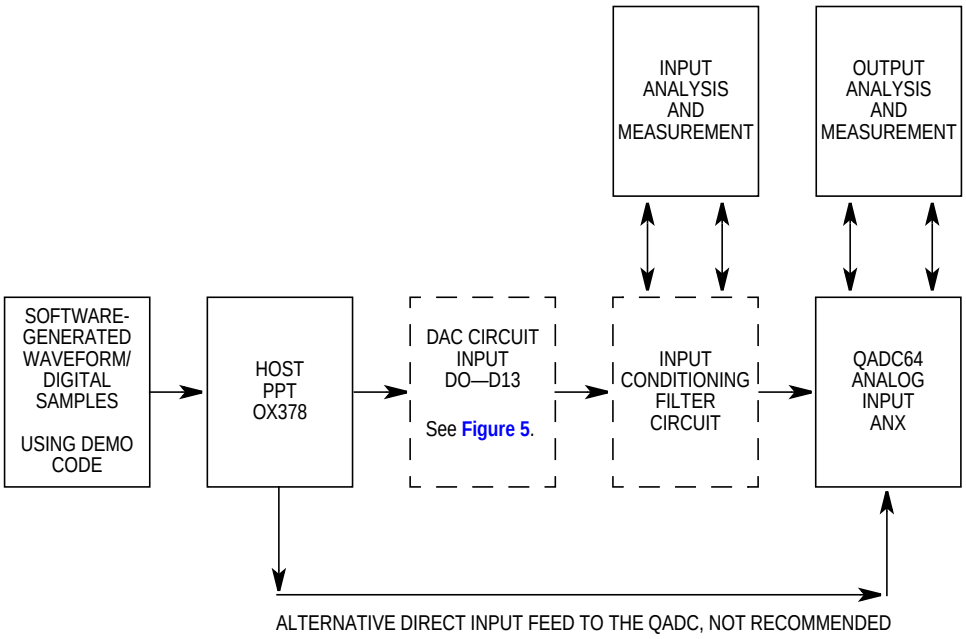


Figure 13. Hardware and Software Design Flow Block Diagram

QADC64 Test Hardware Schematic

Figure 14 shows the schematic of a fully tested and implemented circuit. Developers, using the QADC64 example software in this application note, can use this circuit. The 14-bit monolithic CMOS digital-to-analog converter (AD7538KN) accepts digital values from the host parallel port. The first filter circuit using the LM324 opamp serves as a reference voltage conditioning circuit which can be customized to minimize noise to the DAC as well as protect against ESD spikes.

The second filter circuit serves to filter noise and provide more accurate voltage values for the QADC64 analog input ANX. Similar QADC64 conversion results were noted using this circuit with the example software.

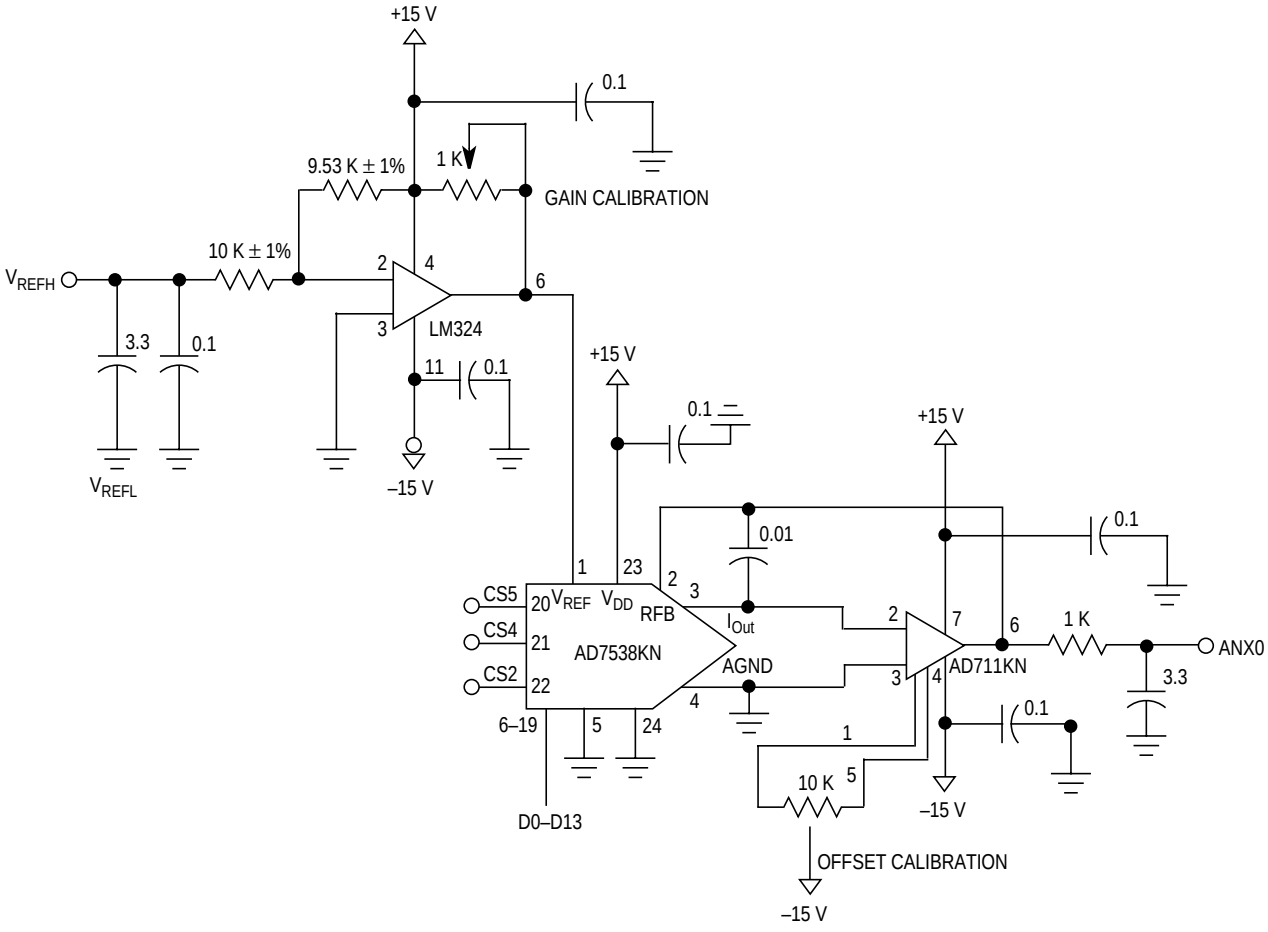


Figure 14. Experimental QADC64 Test Circuit

To calibrate offset:

1. Load $0000 into DAC using software.
2. Use offset pot to null ANX0 voltage with respect to ground.

To calibrate gain:

1. Load $3FFF into DAC using software.
2. Use gain pot to null ANX0 voltage with respect to V_{REFH}.

Parts Listing

Parts needed include:

- Philips LM 324 operational amplifier
- Analog devices AD7538KN 14-bit digital-to-analog converter
- Philips AD711KN operational amplifier
- Standard resistors
- Standard capacitors

References

Breiner, Biran. *Matlab for Engineers*. Harlow, England, Addison-Wesley, © 1995.

MC68336/376 User's Manual, Motorola document order number MC68336/376UM/AD.

Microelectronics Circuits, 4th edition. Oxford University Press, New York, New York: Sedra & Smith.

Motorola Embedded Controllers Website http://www.mcu.motsps.com/.

Pease, Robert A. *Troubleshooting Analog Circuits*. Boston, Massachusetts, Butterworth-Heinemann, © 1991.

Transient Voltage Suppressor for ESD Protection Technical Data, Motorola document order number MMQA5V6T1/D.

Valentine, Richard. *Motor Control Electronics Handbook*. New York, New York, McGraw Hill, © 1998.

How to reach us:

USA/EUROPE/Locations Not Listed: Motorola Literature Distribution, P.O. Box 5405, Denver, Colorado 80217.
1-800-441-2447 or 1-303-675-2140. Customer Focus Center, 1-800-521-6274

JAPAN: Motorola Japan Ltd.: SPD, Strategic Planning Office, 141, 4-32-1 Nishi-Gotanda, Shinagawa-Ku, Tokyo, Japan, 03-5487-8488

ASIA/PACIFIC: Motorola Semiconductors H.K. Ltd., Silicon Harbour Centre, 2 Dai King Street, Tai Po Industrial Estate,
Tai Po, New Territories, Hong Kong, 852-26629298

Mfax™, Motorola Fax Back System: RMFAX0@email.sps.mot.com; http://sps.motorola.com/mfax/;
TOUCHTONE, 1-602-244-6609; US and Canada ONLY, 1-800-774-1848

HOME PAGE: http://motorola.com/sps/

MOTOROLA