4A24 A-D CARD MANUAL

Version 1.6
HW Rev 9 - SW Rev. B5

This page intentionally not blank

Table of Contents

Table of Contents

4A24

GENERAL

The 4A24 is a high resolution, high speed, low cost DSP based A-D card for the PC/104 bus. 200 Khz and 500 KHz models are available. On card EEPROM calibration storage and fully digital zero and span calibration eliminates potentiometers and simplifies circuitry for low cost and high performance.

A high bandwidth instrumentation amplifier is used for the input. 16 single ended or 8 differential inputs are provided via the input multiplexor. A 0 to 5V unipolar range and +- 2.5V and +- 5V bipolar ranges are jumper selectable

Input channel scan sequence, averaging, option, raw/calibrated data choice, and autozero option are determined by a built-in 1024 entry function table. Four function table extended multiplexor select addresses are available from the Analog I/O connector for submultiplexing or other functions.

The averaging option in the function table allows A-D data on selected channels to be unaveraged, or averaged from 2 to 128 times for noise reduction. On card averaging reduces bus utilization compared to host averaging of A-D data.

Converted data is placed in a 128K word sample FIFO so that host latency requirements are relaxed.

Conversions can be started by the timer, an external start convert input, or the host processor. Synchronous timed burst conversions (1 to 2^32-1 conversions) can be started by the host or the external start convert input.

The conversion start timer is a 24 bit, 50 MHz programmable divider for high resolution and wide conversion rate range.

The 4A24 can operate in polled or interrupt driven mode. In polled mode, the data available status of the FIFO is read from a status port and if data as available, the FIFO port can be read. In the simplest polled mode, the host writes to the FIFO port to start a conversion, polls the status register for the DAV flag and then reads the A-D data from the FIFO port.

In interrupt driven mode, an interrupt is generated when the 128K word FIFO reaches a predetermined count. At this point the host can do a block I/O read from the FIFO to efficiently transfer the data to the host. The 4A24 can be programmed to use any of the available PC/104 bus interrupts.

24 general purpose I/O bits are provided. Each I/O bit can be individually programmed to be an input or output. Optional pullups are provided for all I/O bits.

All host interface logic on the 4A24 is contained in a FPGA to allow field upgradability and customization for special applications including on card signal processing ad high speed I/O.

HARDWARE CONFIGURATION

GENERAL

The 4A24 has 2 hardware settable options. These options are the input range and single-ended/differential input mode selection. Hardware options are determined by moving sets of jumpers to different positions. The jumper positions assume that the 4A24 card is right-side up, that is the 4A24 silkscreen text is right-side up.

INPUT RANGE

The 4A24 has three input ranges, -2.5 --> +2.5V, 0 --> +5V and -5 --> +5V. These ranges are selected by jumpers W4 and W5

W4	W5	RANGE	
DOWN	UP	-2.5V --> +2.5V	
DOWN	DOWN	0 --> +5V	
UP	UP	-5V --> +5V	(DEFAULT)

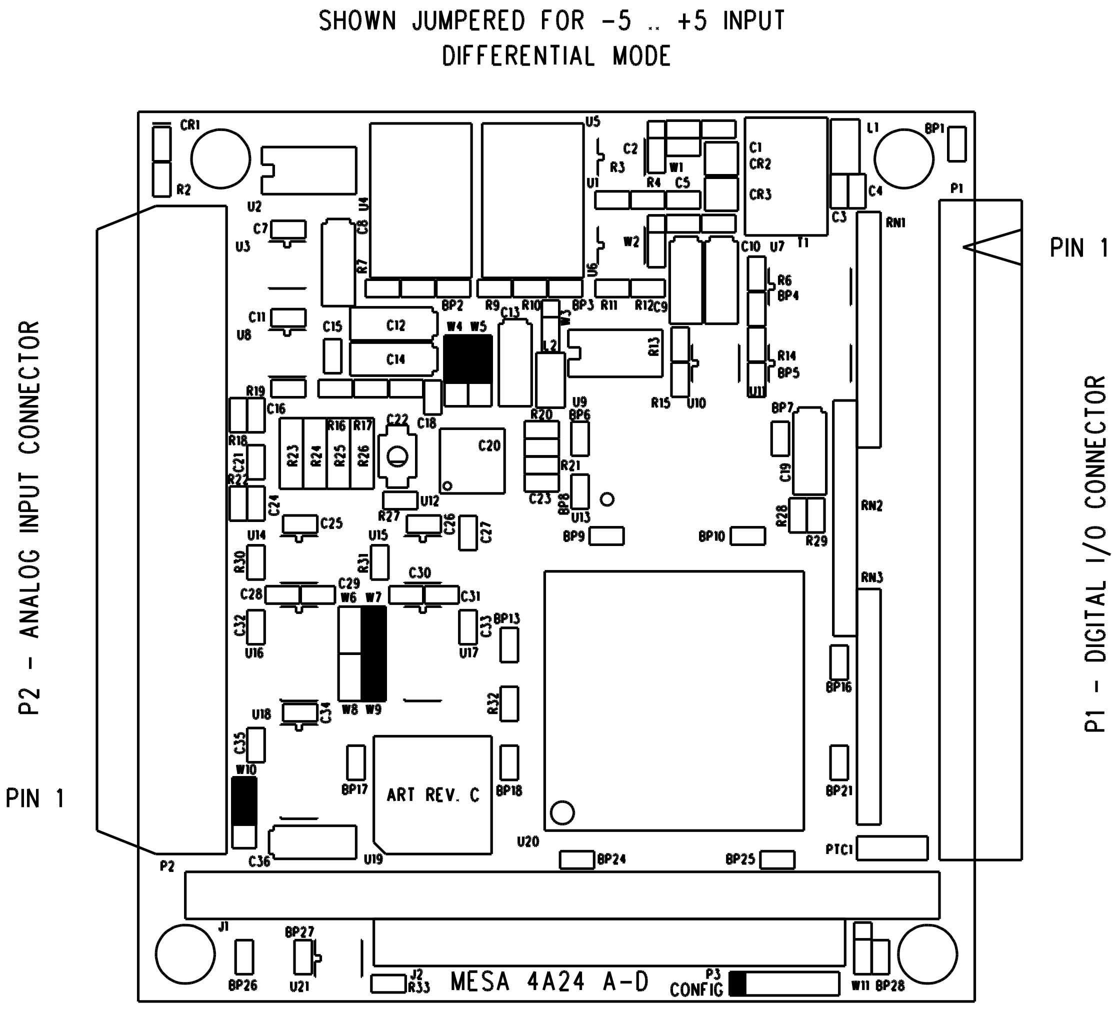
At card reset when the EEPROM cardmode variable is set to auto (the default), the jumper settings are sensed and the A-D setup and calibrated for the current input range. This is only done at card reset, so if you change the input range jumpers, you must reset the card before it will function correctly.

SINGLE-ENDED/DIFFERENTIAL MODE

The 4A24's input circuitry can operate in single ended or differential mode. In single ended mode, 16 single ended inputs are available. In differential mode 8 differential inputs are available. The input mode is selected by jumpers W6,W7,W8,and W9. When only W6 and W8 are installed (both jumpers on left side), single ended mode is selected. When only W7 and W9 are installed (both jumpers on right side), differential mode is selected.

W6	W8	W7	W9	MODE	
IN	IN	OUT	OUT	SINGLE ENDED	
OUT	OUT	IN	IN	DIFFERENTIAL	(DEFAULT)

CONNECTOR AND DEFAULT JUMPER LOCATIONS


SHOWN JUMPERED FOR -5 .. +5 INPUT
DIFFERENTIAL MODE
P2 - ANALOG INPUT CONNECTOR
PIN 1
P1 - DIGITAL I/O CONNECTOR
PIN 1
MESA 4A24 A-D
ART REV. C
P3 CONFIG
W1
W2
W3
W4 W5
W6 W7
W8 W9
W10
W11

HOST INTERFACE

INTERFACE SPECIFICATIONS

GENERAL

The 4A24s host interface consists of two main parts, a FIFO interface, used for reading A-D data, and the internal processor interface, used for setup and control. The host interface appears as a register map with seven 16 bit registers starting at the BASE address +2. (Note that that all addresses are *byte* addresss) All host interaction with the 4A24 is done via these registers. The default 4A24 base address is 0x220h. 4A24's with firmware revision of B5 or greater have the option of a user settable base address.

FIFO

Acquired analog data is placed on a hardware FIFO that can be read from the PC/104 bus. This FIFO has a capacity of 131072 (2^17) samples. The FIFO is read at 4A24 base address.

DATA FIFO @ BASE+0

D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D

Writes to the FIFO can be used to start individual A-D conversions or a burst of conversion if these options are enabled. The specific data written to the FIFO is ignored.

COMMAND/STATUS REGISTER

The register at base address +2 is the command/status register. The command/status register is used in conjunction with the data register for reading and writing A-D setup parameters. The bit definition for command/status register writes are as follows:

COMMAND/STATUS REGISTER WRITES @BASE + 2

W	X	X	X	X	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0

W (MSB) is the write bit. If this bit is ‘1', a write operation will be performed. If it is a ‘0', a read operation will be performed. The P0 through P10 bits specify which parameter is to be read or written

HOST INTERFACE

INTERFACE SPECIFICATIONS

On reads, Bit 15 of the command/status register is the busy bit, indicating that a command is in progress. Bit 14 is the Data Available bit, indicating that data is available from the host FIFO. Bit 13 is the IRQ status bit that reflects that state of the IRQ FF. The other bits (S) are general purpose status bits that can be used by the firmware to show internal system status.

COMMAN/STATUS REGISTER READS @BASE + 2

B	DA	IRQ	S	S	S	S	S	S	S	S	S	S	S	S	S

On power up, the lower 8 status bits in the command/status register show the DSP firmware revision number.

DATA REGISTER

A-D setup and control parameters are transferred to and from the 4A24 via the data register. The data register is located at 4A24 base address +4.

DATA REGISTER @ BASE+4

D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D

WRITING A PARAMETER

The process of writing a parameter is as follows: First write the parameter data to the data register. Next the command register is written with the desired parameter number and write flag (1 bit in MSB). When the command is written, the busy bit will be set and will stay set until the command is completed. Commands and data must not be written when the busy bit is high.

READING A PARAMETER

The process of reading a parameter is as follows: First the command register is written with the desired parameter number. Once the command is written the busy bit will be set and will stay set until the requested data is available in the data register. Once the busy bit returns to 0 the data can be read from the data register. Commands must not be written when the busy bit is high. Data should not be read from the data register when the busy bit is high.

HOST INTERFACE

GENERAL

BUSY TIME

The internal DSP handles host interface requests on a synchronous polled basis, that is it does all the A-D operations, then runs a host interface loop until A-D data becomes available at the next sample time. One side effect of this mode of operation is that the host interface will stay busy for the time it takes to process the A-D data (~2 usec), and that the host interface bandwidth will decrease with increasing A-D sample rates.

Note that this busy time does not affect the data FIFO. Transfers from the data FIFO can run as fast as 5M words per second.

If a host interface cycle starts just after A-D data needs to be processed, the interface busy bit will stay high for the maximum (~2 usec) period. There is always one host interface cycle done per sample period, so that if you set the sample rate faster than the DSP can process the A-D data, the host interface will still function.

HOST INTERFACE

GENERAL

FIRMWARE DOWNLOAD

The 16 bit DSP in the 4A24 can have its firmware downloaded from the host if desired. This will overwrite the standard 4A24 firmware that is part of the FPGA configuration. Three registers are involved in program downloading: the program address register, the program data register, and the processor reset register.

PROGRAM ADDRESS REGISTER @ BASE+8

X	X	X	X	X	A	A	A	A	A	A	A	A	A	A	A

PROGRAM DATA REGISTER @ BASE+10

D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D

DSP RESET REGISTER @ BASE+14

X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	R

DOWNLOAD PROCESS

The process for downloading new DSP firmware is as follows: First the DSP is reset by writing a 1 to the DSP reset register. Then all the words of DSP program are written by first writing the target address for the word to the program address register, then writing the program data for that address to the program data register. This is repeated for all the words of the program firmware. When all the program words have been written, DSP reset is removed by writing a 0 to the DSP reset register, starting execution of the new code.

CUSTOM FIRMWARE

Please contact MESA if you need custom features in the DSP firmware. We can also help if you wish to develop your own firmware or modify the standard firmware. Custom firmware can be built into the FPGA configuration if desired.

CONNECTORS

ANALOG INPUT CONNECTOR

P2 is the analog input connector. P2 is a latching 40 pin header for 40 conductor .050" pitch flat cable P2 pinout is as follows:

1	SHIELD GND	2	SHIELD GND
3	+IN0 (IN0)	4	-IN0 (IN8)
5	+IN1 (IN1)	6	-IN1 (IN9)
7	+IN2 (IN2)	8	-IN2 (IN10)
9	+IN3 (IN3)	10	-IN3 (IN11)
11	SHIELD GND	12	SHIELD GND
13	+IN4 (IN4)	14	-IN4 (IN12)
15	+IN5 (IN5)	16	-IN5 (IN13)
17	+IN6 (IN6)	18	-IN6 (IN14)
19	+IN7 (IN7)	20	-IN7 (IN15)
21	SHIELD GND	22	SHIELD GND
23	INPUT COMMON	24	INPUT COMMON
25	VREF OUT	26	VREF OUT
27	ANALOG GND	28	ANALOG GND
29	SHIELD GND	30	SHIELD GND
31	EXTSTRTCONV	32	POWER GND
33	MUXA3	34	MUXA4
35	MUXA5	36	MUXA6
37	MUXA7	38	NC
39	+5V POWER	40	POWER GND

Input names in parenthesis are for single ended mode. INPUT COMMON is AD common input for single ended mode.

CONNECTORS

DIGITAL I/O CONNECTOR

P1 is the digital I/O input connector. Two 12 BIT digital I/O ports are available on P1. P1 is a 50 pin shrouded header for 50 conductor .050" pitch flat cable. P1 pinout is as follows:

1	PORTA0	3	PORTA1
5	PORTA2	7	PORTA3
9	PORTA4	11	PORTA5
13	PORTA6	15	PORTA7
17	PORTA8	19	PORTA9
21	PORTA10	23	PORTA11
25	PORTB0	27	PORTB1
29	PORTB2	31	PORTB3
33	PORTB4	35	PORTB5
37	PORTB6	39	PORTB7
41	PORTB8	43	PORTB9
45	PORTB10	47	PORTB11
49	+5V POWER		

All even numbered pins are grounded.

OPERATION

PARAMETERS

GENERAL

The 4A24 A-D card has a large number of parameters that control its operation, Some of these parameters access internal hardware and some are variables that the DSP uses. There are more parameters and internal hardware than shown in this table, a complete list of parameters can be found in the DSP source code listing in the appendix of this manual

PARAMETER DESCRIPTIONS

OFFSET AND GAIN PARAMETERS:

PARAMETER	NUMBER	DESC
ADOFFSET5B	22	A-D offset for 5V bipolar mode, read from EEPROM at startup.
ADGAIN5B	23	A-D gain for 5V bipolar mode, read from EEPROM at startup.
ADOFFSET2D5B	24	A-D offset for 2.5V bipolar mode, read from EEPROM at startup.
ADGAIN2D5B	25	A-D gain for 2.5V bipolar mode, read from EEPROM at startup.
ADOFFSET5U	26	A-D offset for 5V unipolar mode, read from EEPROM at startup.
ADGAIN5U	27	A-D gain for 5V unipolar mode, read from EEPROM at startup
ADOFFSET2D5U	28	A-D offset for 2.5V unipolar mode, read from EEPROM at startup.
ADGAIN2D5U	29	A-D gain for 2.5V unipolar mode, read from EEPROM at startup.
ADOFFSET	30	Working value of A-D offset, one of the above offset values, chosen at startup based on the CARDMODE EEEPROM value. May be overwriiten by host or DSP in autozero mode.
ADOFFSETHIGH	31	High 16 bits of ADOFFSET (sign extended)

OPERATION

PARAMETER DESCRIPTIONS

OFFSET AND GAIN PARAMETERS:

PARAMETER	NUMBER	DESC
ADGAIN	32	Working A-D gain calibration value, chosen from the EEPROM gain values above at startup depending on the EEPROM CARDMODE value.
CARDMODE	45	Determines A-D range, read from EEPROM at startup or automatically determined by startup logic if EEPROM CARDMODE parameter is 0

FIFO AND INTERRUPT PARAMETERS

BOUNDEDFIFOCNT	38	Count of A-D samples available in host FIFO. If available samples in FIFO is > 65535, count is bounded to 65535
IRQSETTHRESH	39	FIFO sample count value to set IRQ FF, must be >= IRQCLRTHRESH
IRQCLRTHRESH	40	FIFO sample count value to clear IRQ FF, must be <= IRQSETTHRESH

PARAMETERS THAT ACCESS INTERNAL HARDWARE

FIFOCLR	1042	Writes here clear the Host FIFO
IRQDRVREG	1044	IRQ setup register, specifies which IRQ to drive and has tri-state drive enable bit.
PORTADATA	1048	12 bit I/O port A data register
PORTADDR	1049	Port A Data Direction Register (1 = out)
PORTBDATA	1050	12 bit I/O port b data register
PORTBDDR	1051	Port B Data Direction Register (1= out)

OPERATION

PARAMETER DESCRIPTIONS

PARAMETERS THAT ACCESS INTERNAL HARDWARE

MUXTABLEWADD	1056	MUXTABLE write address (address in table where MUXTABLE data will be written)
MUXTABLEDATA	1057	MUXTABLE data write port
MUXTABLEPTR	1058	MUXTABLE read pointer initialized here
CONVSTARTREG	1060	A-D convertor start convert source mask register
SAMPLECOUNTLOW	1061	Low word of convert count
SAMPLECOUNTHIGH	1062	High word of convert count
PROCSTARTCONV	1064	Writes here by DSP start an A-D conversion
ADMODEREG	1065	A-D operation mode control register+ power supply enable
TIMERLOW	1066	A-D sample rate generator low word
TIMERHIGH	1067	A-D sample rate generator high byte
LED	1088	debug LED
HWREV	1089	Hardware Revision
DECODE	1090	PC/104 I/O base address

INTERNAL HARDWARE

MUXTABLE

The MUXTABLE is central to the operation of the 4A24, it determines the input channel scanning sequence, whether A-D data is averaged or used as-is, whether A-D data is to be presented to the host in raw or calibrated format, or whether A-D data is to be used for an internal auto-zero operation. The MUXTABLE is accessed with 2 pointers, the read pointer and the write pointer. The read pointer is automatically advanced when a A-D conversion is started. This allows input multiplexor changes and other sequential input operations to happen without host intervention The write pointer is used by the host to initialize the entries in the MUXTABLE.

OPERATION

INTERNAL HARDWARE

MUXTABLE

The MUXTABLE is a 16 bit wide memory with 1024 entries. Each entry has the following format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CLR	R/C	AZ	IM1	IM0	AV2	AV1	AV0	CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0

Bits 7 through 0 (CH7 through CH0)are the channel select bits. They determine the input channel selected by the 4A24's input multiplexor. Note that only CH3 through CH0 are used on the 4A24 card, CH7 through CH4 are available at the 4A24's input connector for use in submultiplexing or other uses.

Bits 10 through 8 (AV2 through AV0) determine whether and how many times the A-D input should be averaged. When AV2 through AV0 are 0, no averaging is done. The number of A-D readings averaged is 2^AV:

AV2	AV1	AV0	Number of A-D readings averaged
0	0	0	1
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

Note that when averaging is done, the MUXTABLE read pointer is not incremented in response to a A-D conversion start until the desired number of conversions have been averaged

OPERATION

INTERNAL HARDWARE

MUXTABLE

MUXTABLE bits 12 through 11 (IM1 and IM0) are the input mode bits. They select the input data routing to the A-D:

IM1	IM0	A-D input connection
0	0	Normal
0	1	Differential inputs reversed
1	0	A-D inputs connected to VREF
1	1	A-D inputs connected to GND (for Auto zero)

MUXTABLE bit 13 is the autozero bit. When it is 0, a normal reading takes place, when it is a one, the A-D reading is not sent to the host, but is saved in the ADOFFSET parameter for use by the DSP for offset calibration.

Bit 14 of the MUXTABLE entry is the RAW/CALIBRATE bit. It determines if the data from the A-D should be returned to the host in the raw or calibrated form. If RAW/CALIBRATE is 0, calibrated data is returned, if RAW/CALIBRATE is 1, raw data direct from the A-D is returned to the host.

Bit 15 is the CLR bit. It is used to determine the length of the MUXTABLE sequence. If the CLR bit is a 1, the MUXTABLE read pointer is cleared at the next sample time.

If the CLR bit is set to 1 in the first table entry (location 0), the Muxtable read pointer will never advance, and the MUXTABLE can be used as a simple latch. This is useful for simple polled mode operation where the automatic features of the MUXTABLE are not needed.

If less than the full 1024 MUXTABLE locations are used, the last table entry should always have the CLR bit set to 1 so that the table sequence will restart at 0 after the last entry.

OPERATION

INTERNAL HARDWARE

LOADING THE MUXTABLE

Before the MUXTABLE can be used it must be loaded with valid table data. This is done with a sequence of parameter writes to the MUXTABLEWADD location to set the table address where the data is written followed by a parameter write of the table data to the MUXTABLEDATA location.

STARTING OUT RIGHT

In order to synchronize the first A-D conversion with the first MUXTABLE entry It is necessary to set the MUXTABLEPTR to 0 *before* starting conversions.

CONVERT START REGISTER

The convert start register is a mask register that determines the source of the A-D start conversion signals. It also controls the GO bit that is used to control bursts of conversions.

The convert start register has the following format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GO	XX	XX	XX	XX	SB	HSB	EBP	ESB	PSB	FRE	HST	EPL	EST	PST	TST

Bit 0 through 4 are mask and polarity bits affecting the source and polarity of the start conversion signal.

Bit 0,TST is the TimerSTart bit. It enables the sample rate timer to start conversions when set to 1.

Bit 1,PST is the ProcessorSTart bit. It enables the embedded DSP processor to start conversions when set to 1.

Bit 2,EST is the ExternalSTart bit. It enables the external start conversion input when set to 1.

Bit 3,EPL is the ExternalPoLarity bit. It determines the active polarity of the external start conversion input. When 0, the external start convert input is active low, when 1, the external start convert input is active high.

Bit 4,HST is the HostSTart convert bit. It enables the host to start conversions by *writing* to the FIFO port.

OPERATION

INTERNAL HARDWARE

CONVERT START REGISTER

Bit 5, FRE is the sample rate generator free run bit. It gates the clock to the sample rate generator. This needs to be set to 1 if the sample rate generator is free running (non-burst mode)

Bits 6 through 9 are mask and polarity bits that determine if a start convert signal also sets the GO bit, enabling a burst of conversions

Bit 6, PSB is the ProcessorStartBurst bit. If set, a DSP processor start convert command will set GO, starting a burst of conversions.

Bit 7, ESB is the ExternalStartBurst bit. If set, an external start convert command will set GO, starting a burst of conversions.

Bit 8, EBP is the ExternalBurstPolarity bit. It determines the active polarity of the external start conversion input for starting bursts. When 0, the external start convert input is active low, when 1, the external start convert input is active high.

Bit 9, HSB is the HostStartBurst bit. If set, a host start convert command will set GO, starting a burst of conversions.

Bit 10, SB is the SingleBurst bit. If set, GO can not be set if the sample count register is 0. This makes it easy to generate a single burst of conversions.

Bit 14, EXT is a read-only bit that reflects the status of the External Start conversion input

Bit 15, GO is a read-only bit that reflects the status of the GO bit.

When using burst mode, the burst start signal normally should not start a conversion. In normal (synchronous) operation, It sets GO which in turn enables the the sample rate timer. Since we only want the sample rate timer to start conversions in the burst mode, only the timer start conversion bit and the desired start burst bit should be set in the convert start register.

When using burst mode, and the timing of the first conversion is important, the sample rate timer should be written before starting a burst. This will guarantee that the first conversion will start within ~70 nS of the burst start signal.

OPERATION

INTERNAL HARDWARE

SAMPLE COUNT REGISTER

The sample count register is a 32 bit register that determines the number of conversions done in burst mode. The sample count register is loaded with the desired convert count and is decremented at each conversion start. When the count changes from 1 to 0 , the GO bit is cleared. If the conversion count is programmed to 0, GO will not be cleared, and the burst will have indefinite length.

If only a single burst of conversions is desired, and the external start conversion input is used to start this burst, the SingleBurst mode bit should be set. This will prevent the GO bit from being set by external start conversion edges after the burst is complete.

Note that the sample count register is accessed in two halves SAMPLECOUNTLOW and SAMPLECOUNTHIGH. When initializing the sample count register the SAMPLECOUNTLOW register must be written first and the SAMPLECOUNTHIGH register written last. The register is updated when SAMPLECOUNTHIGH is written.

SAMPLE RATE TIMER

The sample rate timer is a 24 bit programmable divider that is used to set the A-D conversion rate in free run and burst modes. The input frequency to the rate timer is 50 MHz giving an A-D conversion rate of

$$50e6/TIMER$$

As TIMER is a 24 bit parameter, it must be updated by writing to two parameters, TIMERLOW and TIMERHIGH. TIMERLOW is the low 16 bits of the divisor and TIMERHIGH is the high 8 bits. When initializing the TIMER register the TIMERLOW register must be written first and the TIMERHIGH register written last.

The sample rate timer input clock is gated by the FRE bit in the convert start register and the GO bit. Either FRE or GO must be set to enable the sample rate generator.

OPERATION

INTERNAL HARDWARE

INTERRUPT SELECT REGISTER

The interrupt select register determines which interrupt is generated on FIFO full conditions. It also controls the IRQ line tri-state and has in interrupt mask bit.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
XX	XX	XX	XX	XX	XX	XX	XX	XX	MS	XX	TSE	IS3	IS2	IS1	IS0

Bits 0 through 3 (IS0..IS3) select which interrupt is generated. IS0 through IS3 are a binary representation of the interrupt number, for example:

	IS3	IS2	IS1	IS0
IRQ5	0	1	0	1
IRQ10	1	0	1	0

Bit 4 is the IRQ tri-state enable. When bit 4 is 0, the IRQ line is floated (not driven). This is the default startup state of the 4A24. If interrupts are used, this bit needs to be set to 1.

Bit 6 is the interrupt mask bit. It is ANDed with the internal interrupt request signal. When bit 6 is 0, no interrupts can be generated.

OPERATION

INTERNAL HARDWARE

DIGITAL I/O PORTS

Two 12 bit digital IO ports are available on the 4A24. Each port can have every bit individually programmed to be input or output. Port operation is controlled by the PORTADATA. PORTADDR. PORTBDATA. and PORTBDDR parameters. Bits in the DDR registers (PORTADDR and PORTBDDR) determine the signal direction of the corresponding ports. A one bit sets the corresponding I/O bit to the output mode and a zero bit sets the corresponding bit to input mode. At power up or DSP reset, the DDR registers are set 0x0000 so all I/O bits are in input mode initially.

PORTA and PORTB DATA REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
XX	XX	XX	XX	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0

PORTA and PORTB DDR REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
XX	XX	XX	XX	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

The digital I/O ports have a 24 mA current sink capability and built in 3.3K pullup resistors on each I/O bit. The I/O bits are driven from 3.3V power and will only swing up to ~ 3.3V.

OPERATION

SETUP EXAMPLES

SIMPLE POLLED MODE

In this mode conversions will be started by the host, and the host will poll the data available bit of the host interface to know when the conversion data is available

First we will setup the MUXTABLE and CONVSTART register for simple polled operation:

1. Write 0 to the CONVSTART register to mask all conversion start sources

This is to make sure no conversions are happening when we do our setup

2. Write 0 to MUXTABLEWADD

Set the MUXTABLE write pointer to 0

3. Write channel number+input mode + CLR bit (0x8000) to MUXTABLEDATA

We are writing the first entry in the MUXTABLE with our desired channel number plus the CLR bit. Having the CLR bit set and using the first entry(0) in the MUXTABLE makes the MUXTABLE behave like a simple latch.

4. Write 0 to FIFOCLR location

We clear the FIFO just in case it contains data from previous operations

5. Write 0 to MUXTABLEPTR

Set the MUXTABLE read pointer to 0. This is so that we are sure that the read pointer that determines the MUXTABLE data entry presented to the input multiplexors and post processing logic is from location 0

6. Write HST bit (0x010) to CONVSTART register to enable host started conversions

This enables writes to the DATA FIFO (at the 4A24 BASE address) to start conversions

At this point conversions can be started by writing to the DATA FIFO address. Data will be available for reading when the DAV bit becomes set in the status register.

Note that multiple host started conversions can be done as long as the time between conversion starts is greater than or equal to the ADC's conversion time. The result of these conversions will be stored in the data FIFO until read by the host.

OPERATION

SETUP EXAMPLES

TIMER START MODE

In this mode conversions are started at regular intervals by the timer. In this example we will also setup the MUXTABLE to automatically scan 3 input channels: 4, 2 and 7 repeatedly at a 100 KHz rate. We will also setup the 4A24 to generate interrupt 12 when the FIFO contains 32768 or more samples.

1. Write 0 to the CONVSTART register to mask all conversion start sources

This is to make sure no conversions are happening when we do our setup

2. Write 0 to MUXTABLEWADD

Set the MUXTABLE write pointer to 0

3. Write channel number (4) to MUXTABLEDATA

Setup entry 0 in the MUXTABLE to read channel 4

4. Write 1 to MUXTABLEWADD

Set the MUXTABLE write pointer to 1

5. Write channel number (2) to MUXTABLEDATA

Setup entry 1 in the MUXTABLE to read channel 2

6. Write 2 to MUXTABLEWADD

Set the MUXTABLE write pointer to 2

7. Write channel number (7) +CLR bit (0x8000) to to MUXTABLEDATA

Setup entry 2 in the MUXTABLE to read channel 7 and clear then read pointer so that the next conversion will use MUXTABLE entry 0.

8. Write 0 to FIFOCLR location

We clear the FIFO just in case it contains data from previous operations

OPERATION

SETUP EXAMPLES

TIMER START MODE

9. Write 0 to MUXTABLEPTR

Set the MUXTABLE read pointer to 0. This is so that we are sure that the read pointer that determines the MUXTABLE data entry presented to the input multiplexors and post processing logic starts from location 0

10. Write 32768 to parameter IRQSETTHRESH

Setup FIFO logic to generate interrupt when FIFO has 32768 or more entries.

11. Write 32767 to parameter IRQCLRTHRESH

Setup FIFO logic to clear the IRQ when the FIFO count has 32767 or fewer entries

12. Write (12 + TSE (0X10) + MS (0x40)) to parameter IRQDRIVEREG

Select interrupt 12, enable the IRQ tristate driver, and unmask the interrupt

13. Write 500 (50 MHz/100 KHz) to parameter TIMERLOW

Setup low word of our rate generator for 100 KHz

14. Write 0 to parameter TIMERHIGH

Setup high byte of our rate generator. Note that the high byte must always be written even when it is 0 or has not changed from previous writes, as write to the high byte updates the 24 bit rate divisor latch.

15. Write (CST (0x01) + FRE (0x20)) bits to parameter CONVSTARTREG

Last thing is to enable the timer to free run by setting FRE and enable timer conversion starts with CST)

At this point conversions will be running continuously at a 100 KHz rate, sequencing through channel 4,2,and 7 repeatedly. The FIFO logic will cause an interrupt (if enabled by the host) when it gets 32768 samples. When the interrupt occurs the host can do a block read of all 32768 samples, since it is guaranteed that at least 32678 samples are available in the FIFO.

REFERENCE

SPECIFICATIONS

	MIN	MAX	NOTES:
POWER SUPPLY	4.75V	5.25V	(5V only)
POWER CONSUMPTION:			
ACTIVE	----	350 mA	
IDLE	----	200 mA	
ACCURACY	-.05%	+.05%	(Over full operating temperature)
CONVERSION RATE 4A24-2	---	200 KHz	Less than 2 LSBs settling time error for 1/2 scale step
CONVERSION RATE 4A24-5	---	500 KHz	Less than 10 LSBs settling time error for 1/2 scale step
INPUT CURRENT	- 200 nA	+200 nA	
INPUT COMMON MODE RANGE	-5.5V	+5.5V	
OPERATING TEMP.	0^oC	$+70^oC$	
OPERATING TEMP. (-I version)	-40^oC	$+85^oC$	
OPERATION HUMIDITY	0	95%	NON-CONDENSING

APPENDIX A - DSP OPCODES

MEMORY REFERENCE INSTRUCTIONS

(all instructions = 1 clock cycle = 20 nS)

A = accumulator M = memory C= carry bit

MNEMONIC	OPCODE	OPERATION
OR	7XXX	A<=A | M
XOR	8XXX	A<=A XOR M
AND	9XXX	A<=A & M
ADD	AXXX	A<=A+M
ADC	BXXX	A<=A+M+C
SUB	CXXX	A<=A-M
SUBC	DXXX	A<=A-M-C
LDA	EXXX	A<=M
STA	FXXX	M=>A

MEMORY REFERENCE INSTRUCTION ADDRESSING MODES:

MODE	EXAMPLE	NOTES
ABSOLUTE:	ADD M	M = 0..2047 absolute address
INDEXED	ADD @X	X = index register x
INDEX+OFFSET	ADD@Y,OFF	Y = index register y, OFF = 0..511 offset

APPENDIX A - DSP OPCODES

JUMP INSTRUCTIONS

MNEMONIC	OPCODE	OPERATION
JMP P	1PPP	PC <= P
JMPNZ P	2PPP	IF A<>0000 THEN PC <= P ELSE PC <= PC+1
JMPZ P	3PPP	IF A=0000 THEN PC <= P ELSE PC <= PC+1
JMPNC P	4PPP	F C=0 THEN PC <= P ELSE PC <= PC+1
JMPC P	5PPP	IF C=1 THEN PC <= P ELSE PC <= PC+1
JSR P	6PPP	PC <= P, R<= PC+1

All jumps take 1 clock cycle = 20 nS but conditional jumps have 2 delay slots following jump, this means the 2 instructions following the jump will always be executed whether or not the jump is taken. Unconditional jumps (JMP and JSR) take 1 clock cycle.

JUMP INSTRUCTION ADDRESSING MODES

ABSOLUTE	JMP P	0..2047 range
INDEXED	JMP@R	0..4095 range

JSR is ABSOLUTE ONLY with 0..4095 range

OPERATE INSTRUCTIONS

LOAD IMMEDIATE GROUP all 1 clock = 20 nS

LDLI	A[7..0] <= IMM BYTE	Load low Immediate
LDHI	A[15..8] <= IMM BYTE	Load High Immediate
LXBI	A <= sign extended IMM BYTE	Load Sign extended byte immediate

APPENDIX A - DSP OPCODES

MISC OPERATE GROUP

NOP	No OPeration
BSW	NEWA[7..0] <= A[15..8], NEWA[15..8] <= A[7..0] BYTE SWAP
ROTCL	ROTate through Carry Left
ROTCR	ROTtate through Carry Right
SXW	Sign eXtend Word: If A[15[= 1 NEWA <= 0xFFFF else NEWA <= 0x0000
BNDU	BouND Unsigned: If A <> 0 then NEWA <= 0xFFFF, Carry <= 1 else Carry <= 0
BNDS	BouND Signed: if A[15] = 0 then NEWA <= 0x7FFF else NEWA <= 0x8000; If A <> 0x0000 and A <> 0xFFFF then Carry <= 1 else Carry <= 0
SATS	SATurate Signed: If previous add operands were positive and we have a negative result, NEWA <= 0x7FFF; If previous add operands were negative and we have a positive result, NEWA <= 0x8000;
SATU	SATurate Unsigned: If previous add operation caused a carry, NEWA <= 0xFFFF; if previous subtract operation caused a carry, NEWA <= 0x0000
CTOF	Copy Carry to Flag
STOF	Copy sign bit to Flag
ZTOF	Copy Zero status to Flag

APPENDIX A - DSP OPCODES

SKIP INSTRUCTION GROUP

SKPNZ	SKiP if Not Zero
SKPZ	SKiP if Zero
SKPNC	SKiP if No Carry
SKPC	SkiP if Carry
SKPP	SkiP if Plus
SKPM	SkiP if Minus
SKPF	SkiP if Flag
SKPF	SkiP if No Flag

INDEX REGISTER GROUP

LDA,X	A <= X	
LDA,Y	A <= X	
LDA,R	A <= R	Get return address reg
STA,X	X <= A	
STA,Y	Y <= A	
STA,R	R <= A	Set return address reg (for computed jump)

APPENDIX B - DSP CODE

```
.MSFIRST
.include "global.equ"

;some useful constants

; new cpu sweet16m with bound and skip instructions
; new hardware with combined hostfifo
; update irq added to simplify irq code

signaturel          equ 0FEh
signatureh          equ 0CAh
onekhztimer         equ 0C350h
codeversion         equ 0B4h
cardmodeauto        equ 0
cardmode5b          equ 1
cardmode2d5b        equ 2
cardmode5u          equ 3
cardmode2d5u        equ 4

;ram locations

zeroloc             equ 0
oneloc              equ 1
minusoneloc         equ 2
msbb                equ 3
msbw                equ 4
posmax              equ 5
topnibbleloc        equ 6
ledstate            equ 8
srdavflagloc        equ 9
srhostbusyflagloc   equ 10
inmodezeroloc       equ 11
inmodenormalloc     equ 12
inmoderefloc        equ 13
crparmmaskloc       equ 14
crwritemaskloc      equ 15
testAAAA            equ 16
test5555            equ 17
mtestmask           equ 18
mtdptrclrloc        equ 20
mtfunction          equ 21
adoffset5b          equ 22
adgain5b            equ 23
adoffset2d5b        equ 24
```

APPENDIX B - DSP CODE

```
adgain2d5b          equ 25
adoffset5u          equ 26
adgain5u            equ 27
adoffset2d5u        equ 28
adgain2d5u          equ 29
adoffset            equ 30
adgain              equ 32
rawaddata           equ 33
caldata             equ 35
boundedfifocnt      equ 38
irqsetthresh        equ 39
irqclrthresh        equ 40
datatag             equ 41
avemask             equ 42
mtdraw_calloc       equ 43
mtdautozeroloc      equ 44
cardmode            equ 45
detectread          equ 46
cardmode5udet       equ 47
cardmode2d5bdet     equ 48
singleend           equ 49
unipolar            equ 50
maxdata             equ 51
mindata             equ 52

temp                equ 60


; eeprom routine frame offsets
eedata              equ 0     ; data to/from routine
eeadd               equ 1     ; eeprom address
eecom               equ 2     ; read or write command
eemask              equ 3     ; temp
eetemp              equ 4     ; temp
eereturn            equ 5     ; temp

eeadoffset5b        equ 1     ;
eeadgain5b          equ 2     ;
eeadoffset2d5b      equ 3     ;
eeadgain2d5b        equ 4     ;

eeadoffset5u        equ 5     ;
eeadgain5u          equ 6     ;
```

APPENDIX B - DSP CODE

```
eeadoffset2d5u      equ 7     ;
eeadgain2d5u        equ 8     ;

eecardmode          equ 9     ; card mode


; test and maintenance stuff

doeepromflag        equ 80
eepromtestdatain    equ 81
eepromtestdataout   equ 82
eepromtestadd       equ 83
eepromtestcom       equ 84
eepromframe         equ 90

dotestramflag       equ 100


;internal i/o port locations

commandreg          equ 0400h
; command reg bits
crparmmaskhigh      equ 07h
crparmmasklow       equ 0FFh
crwritemaskhigh     equ 080h
crwritemasklow      equ 00h

clearbusyflag       equ 0401h
datareglow          equ 0402h
datareghigh         equ 0403h

statusreg           equ 0405h
; statusreg bits:
srdavflag           equ 01h
srhostbusyflag      equ 02h
srhostfifofull      equ 04h
srhostfifoemty      equ 08h

multa               equ 0406h
multb               equ 0407h
prodlow             equ 0408h
prodhigh            equ 0409h
```

APPENDIX B - DSP CODE

```
addata              equ 040Ah
adclrdav            equ 040Ah

fifopush            equ 0410h
fifopop             equ 0411h
fifoclr             equ 0412h
fifocntlow          equ 0412h
fifocnthigh         equ 0413h

irqdrivereg         equ 0414h
setirq              equ 0415h
clrirq              equ 0416h
updateirq           equ 0417h

portadata           equ 0418h
portaddr            equ 0419h
portbdata           equ 041Ah
portbddr            equ 041Bh

muxtablewadd        equ 0420h
muxtabledata        equ 0421h

; mux table data bits (in msb)
mtdave0             equ 01h
mtdave1             equ 02h
mtdave2             equ 04h
mtdinmode0          equ 08h
mtdinmode1          equ 010h
mtdautozero         equ 020h
mtdraw_cal          equ 040h
mtdptrclr           equ 080h

muxtableptr         equ 0422h
muxtabledest        equ 0423h

convstartreg        equ 0424h
;convmaskreg bits:
cstimerstart        equ 01h       ; ena for timer start conv
csprocstart         equ 02h       ; ena for internal proc start
                    conv
csextstart          equ 04h       ; ena for external start conv
csextpol            equ 08h       ; polarity of external start
                                    conv
cshoststart         equ 010h      ; ena for host start conversion
```

APPENDIX B - DSP CODE

```
cstimerfre          equ 020h  ; enable timer clock free run

samplecountlow      equ 0425h
samplecounthigh     equ 0426h

clrgo               equ 0427h

procstartconv       equ 0428h


; A-D mode register
modereg             equ 0429h
; modereg bits:
mrpowerdown         equ 01h
mradreset           equ 02h
mrimpulse           equ 04h
mrwarp              equ 08h
mrob_2c             equ 010h
mrenapwr            equ 020h
mrrdc               equ 040h

timerlow            equ 042Ah ; low 16 bits or 24 bit prog divider
timerhigh           equ 042Bh ; high 8 bits (writes here do 24 bit
                                update)

shiftboxlow         equ 042Ch
shiftboxhigh        equ 042Dh
shiftboxcount       equ 042Eh

; eeprom bit i/o -- all in lsb

eedi                equ 0430h ; out eeprom data in;
eedo                equ 0430h ; in eeprom data out
eeclk               equ 0431h ; out eeprom clock
eecs                equ 0432h ; out eeprom cs

eereadcom           equ 06h
eewritecom          equ 05h
eemisccom           equ 04h
eecommask           equ 04h
eeaddmask           equ 020h

led                 equ 0440h
```

APPENDIX B - DSP CODE

```
;program
begin               ; constant initialization
     lxbi 0
     sta  zeroloc
     lxbi 1
     sta  oneloc
     lxbi m1b
     sta  minusoneloc
     ldli 080h
     ldhi 00h
     sta  msbb
     ldli 00h
     ldhi 080h
     sta  msbw

     ldli 00h
     ldhi 0C0h
     sta  cardmode5udet

     ldli 00h
     ldhi 060h
     sta  cardmode2d5bdet

     ldli 0
     ldhi 0F0h
     sta  topnibbleloc
     ldli 0FFh
     ldhi 07Fh
     sta  posmax
     lxbi srdavflag
     sta  srdavflagloc
     lxbi srhostbusyflag
     sta  srhostbusyflagloc
     ldli crparmmasklow
     ldhi crparmmaskhigh
     sta  crparmmaskloc
     ldli crwritemasklow
     ldhi crwritemaskhigh
     sta  crwritemaskloc
     ldli 0AAh
     ldhi 0AAh
     sta  testAAAA
     ldli 055h
```

APPENDIX B - DSP CODE

```
     ldhi 055h
     sta  test5555
     lxbi 0         ; careful! this is used by several sta's
     sta  ledstate
     sta  led
; masks for mux table destination stuff (datatag)
     ldhi 07h
     sta  avemask
     ldhi mtdinmode0 + mtdinmode1
     sta  inmodezeroloc
     ldhi mtdinmode1
     sta  inmoderefloc
     ldhi mtdraw_cal
     sta  mtdraw_calloc
     ldhi mtdautozero
     sta  mtdautozeroloc
     ldli 00h
     ldhi mtdptrclr
     sta  mtdptrclrloc
     lxbi 00h
     sta  portaddr  ; make both of our gpio ports all inputs
     sta  portbddr
     ldli mrenapwr  ; just turn on analog power - 2s comp mode
     sta  modereg
     ldli csprocstart+cshoststart  ; enable proc,and host start
                                   conversions
     sta  convstartreg
     ldli 050h      ; program timer for 1 KHz
     ldhi 0C3h           ; (divide by 50000)
     sta  timerlow
     lxbi 0
     sta  timerhigh
     sta  fifoclr        ; clear the fifo
     ldli codeversion    ; put the firmware rev in status reg
     sta  commandreg     ;
     ldli signaturel     ; put signature in hi word of data reg
     ldhi signatureh     ;
     sta  datareghigh    ;
     lda  muxtablewadd   ; get our single ended bit
     and  msbw      ;
     bndu           ;
     sta  singleend ; set single end flag
```

APPENDIX B - DSP CODE

```
     lxbi 0         ; do some hardware cleanup in case we got
                      here
     sta  clrirq    ; via a DSP reset instead of a system
                      reset
     sta  updateirq ;
     sta  clearbusyflag  ;
     sta  samplecountlow  ;
     sta  samplecounthigh ;
     sta  doeepromflag   ; in case hung
     lda  inmoderefloc   ; for table fill
     jsr  initmuxtab0    ; init the first mux table entry for
                           polled mode

;*****************************************************************
; read the calibration constants into memory

     lxbi eeadoffset5b
     jsr  readeeprom
     sta  adoffset5b
     lxbi eeadgain5b
     jsr  readeeprom
     sta  adgain5b
     lxbi eeadoffset2d5b
     jsr  readeeprom
     sta  adoffset2d5b
     lxbi eeadgain2d5b
     jsr  readeeprom
     sta  adgain2d5b
     lxbi eeadoffset5u
     jsr  readeeprom
     sta  adoffset5u
     lxbi eeadgain5u
     jsr  readeeprom
     sta  adgain5u
     lxbi eeadoffset2d5u
     jsr  readeeprom
     sta  adoffset2d5u
     lxbi eeadgain2d5u
     jsr  readeeprom
     sta  adgain2d5u
     lxbi eeadoffset5b
     jsr  readeeprom
     sta  adoffset5b
     lxbi eeadgain5b
     jsr  readeeprom
     sta  adgain5b

;****************************************************************
; card gain mode setup (via data in eeprom)
     lda  zeroloc
     sta  unipolar  ; assume unipolar
     lxbi eecardmode
     jsr  readeeprom
     sta  cardmode
     sub  cardmodeauto
     jmpnz     adsetup   ; if not zero then do manual setup
     nop                 ; otherwise detect A-D range settings
     lda  inmoderefloc   ; we are going to measure reference
                           voltage
     jsr  procreadad     ; to determine A-D input range
mode5utest
     sta  detectread     ; save our A-D data
     sub  cardmode5udet
     jmpc mode2d5btest   ; its less than so check next
     nop
     lxbi cardmode5u
     sta  cardmode
     jmp  adsetup
mode2d5btest
     lda  detectread     ; get our A-D data
     sub  cardmode2d5bdet
     jmpc mode5btest     ; its less than so check next
     nop
     lxbi cardmode2d5b
     sta  cardmode
     jmp  adsetup
mode5btest
     lxbi cardmode5b
     sta  cardmode
     nop
     nop
adsetup
     lxbi cardmode5b
     sub  cardmode
     jmpnz     init2d5b
     nop
     lda  adoffset5b
     sta  adoffset
     lda  adgain5b
     sta  adgain
init2d5b
     lxbi cardmode2d5b
     sub  cardmode
     jmpnz     init5u
     nop
     lda  adoffset2d5b
     sta  adoffset
     lda  adgain2d5b
     sta  adgain
init5u
     lxbi cardmode5u
     sub  cardmode
     jmpnz     init2d5u
     nop
     lda  adoffset5u
     sta  adoffset
     lda  adgain5u
     sta  adgain
     ldli mrenapwr+mrob_2c    ; Power on + offset binary
     sta  modereg
     lda  minusoneloc
     sta  unipolar
init2d5u
     lxbi cardmode2d5u
     sub  cardmode
     jmpnz     cardmodedone
     nop
     lda  adoffset2d5u
     sta  adoffset
     lda  adgain2d5u
     sta  adgain
     ldli mrenapwr+mrob_2c    ; Power on + offset binary
     sta  modereg
     lda  minusoneloc
     sta  unipolar

cardmodedone
```

APPENDIX B - DSP CODE

```
;*****************************************************************
;    test our external RAM (FIFO memory)
;
     jsr  testram


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

mainloop
     lda  statusreg
     and  srdavflagloc
     jmpnz     getaddata
     lda  statusreg
     and  srhostbusyflagloc
     jmpnz     hostinterface
     lda  doeepromflag
     nop
     jmpnz     doeeprom
     nop
     nop
     jmp  dofifo

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

;*****************************************************************
; initialize location 0 in muxtable for polled use
; enter with muxtable data in acc
; sets up muxtable to loop at 0
; sets ptr to 0

initmuxtab0
     sta  mtfunction     ; save the function
     lxbi 0              ;
     sta  muxtablewadd   ; setup for write to loc 0
     sta  muxtableptr    ; clr the pointer
     lda  mtfunction     ; get the function
     or   mtdptrclrloc   ; set clear_muxtabptr flag
                         ; so the ptr will stick at 0
     sta  muxtabledata   ; write loc 0 of table
     jmp  @R             ; return
```

APPENDIX B - DSP CODE

```
;*****************************************************************
; get and calibrate the a-d data
;

getaddata
     sta  adclrdav  ; first clear the dav flag
     lda  muxtabledest   ; get tag
     sta  datatag        ; save it for later
     lda  shiftboxlow    ; we always accumulate
     add  addata         ; add the A-D data
     sta  shiftboxlow    ; save it
     lda  addata         ;
     sxw                 ; sign extend for high word
     addc shiftboxhigh   ; add high part
     sta  shiftboxhigh   ; save it back
     lda  datatag        ; get data tag
     and  msbw           ; check if this is just to be accumulated
     jmpz mainloop       ; if so, we're done!
     lda  datatag        ;
     and  avemask        ; drop all but ave
     bsw                 ; get ave into lsbs
     sta  shiftboxcount  ; do asr
     lda  unipolar       ; check if unipolar
     jmpz bipolar1       ;
     lda  adgain         ; get fractional part of gain;
     sta  multa          ; store in _signed_ port;
     jmp  waitforshift   ;
bipolar1
     lda  adgain         ; (sta delay) get fractional part of gain;
     sta  multb          ; (sta delay) store in unsigned port;
waitforshift
     lda  shiftboxcount  ;
     jmpnz     waitforshift   ; when shift count is zero (max 5)
calibratedata
```

APPENDIX B - DSP CODE

```
;*****************************************************************
; Most of this stuff is to bound the calibrated data so that there
; are no overflows and therefore seriously bad 16 bit data given to
; the host. Scale calibration is done effectively by a 17 bit
; multiply
; accomplished by adding our original A-D data to the scaled
; fractional part (prodhigh) from our signed 16x16 bit muliplier.
;
     lda  shiftboxlow    ; (in delay slot) get data/averaged data
     nop                 ; (in delay slot)
     sta  rawaddata      ; save raw (maybe averaged) A-D data
     lda  unipolar       ; see if we are unipolar or bipolar
     jmpz bipolar2       ;
     lda  rawaddata      ;
     sub  adoffset       ;
     satu                ; unsigned saturate
     sta  multb          ; write zeroed a-d data to unsigned port
                           of mult
     lxbi 0              ; clear the average accumulator
     sta  shiftboxlow    ;
     sta  shiftboxhigh   ;
     lda  multb          ; get zeroed a-d data again
     add  prodhigh       ; scale calibration
        satu             ; unsigned saturate
     jmp  savecal        ;
bipolar2
     lda  rawaddata      ; get raw data
     sub  adoffset       ; zero it
     sats                ; signed saturate
     sta  multa          ; write zeroed a-d data to signed port of
mult
     lxbi 0              ; clear the average accumulator
     sta  shiftboxlow    ;
     sta  shiftboxhigh   ;
     lda  multa          ; get zeroed a-d data again
     add  prodhigh       ; scale calibration
     sats                ; signed saturate
savecal
     sta  caldata        ; save low 16 bits
     lda  datatag        ; now find out what to do with data
     and  mtdautozeroloc ;
     jmpz raw_cal        ; check if this is an autozero
     lda  rawaddata      ; if so get the raw a-d data
     nop
     sta  adoffset  ; and store it in the offset reg
     jmp  mainloop  ; we're done!
raw_cal             ; check if raw or calibrated data is desired
     lda  datatag        ;
     and  mtdraw_calloc  ;
     jmpnz     useraw         ;
     lda  rawaddata ; (in delay slot - load raw data)
     nop            ; (in delay slot)
     lda  caldata         ; (if we fell through, use cal data)
useraw
     sta  fifopush  ; send data to host
     jmp  mainloop  ; return

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

hostinterface
     lda  commandreg     ; get command
     and  crwritemaskloc ; check if read or write
     jmpnz     hostwrite ;
     lda  commandreg     ; (in delay slot) its a read, get parm
                           address
     and  crparmmaskloc  ; (in delay slot) drop all but parm
                           address
     stx
     nop            ; unavoidable delay from ldx to indirect access
     nop
     lda  @X        ; get parameter
     sta  datareglow     ; send to host
     sta  clearbusyflag  ; clear busy flag to signal that data is
                           avail
     jmp  mainloop       ; return
hostwrite
     stx
     lda  datareglow     ; get the data to write
     sta  clearbusyflag  ; now that we've got the data, signal host
     sta  @X             ; write the parameter
     jmp  mainloop       ; return
```

APPENDIX B - DSP CODE

```
;****************************************************************
; manage fifo count and interrupts
;

dofifo
     lda  fifocnthigh    ; get bit 17,16 of count
     bndu                ; if <> 0 bound at 65535
     skpc                ; skip lda if bounded
     lda  fifocntlow     ; if less than 65536 just use low word
     sta  boundedfifocnt ; save if for later and host access
checkirq
     sub  irqsetthresh   ; see if we need to set IRQ
     skpc                ; if carry, (count < thresh) dont set irq
     sta  setirq         ; if no carry (count => thresh), set irq

     lda  irqclrthresh
     sub  boundedfifocnt ; see if we need to clear irq
     skpc                ; if carry (count > thresh), dont clr
     sta  clrirq         ; if no carry, (count <= thresh) do clr
     sta  updateirq      ; update the flag and irq signal
     jmp  mainloop

;*****************************************************************
; simple debug routine to display acc on 16 bits of I/O
;

displaydata
     sta  portadata ; display lower 12 bits on port a
     and  topnibbleloc   ; drop lower 12 bits
     bsw            ; sr 12
     rcr
     rcr
     rcr
     rcr
     sta  portbdata ; display top 4 bits on port b lsbs
     jmp  @R        ; return
```

APPENDIX B - DSP CODE

```
;***************************************************************
;  routine to do asynchronous raw A-D read
;  start with muxtable data in acc, return with data in acc

procreadad
     stx            ; save muxtable data
     ldr            ;
     sty            ; save return address
     ldx            ; get muxtab data
     jsr  initmuxtab0    ; init table with mux tab data
     lxbi 50        ; wait 4 usec for input to settle
pwaitloop
     sub  oneloc
     jmpnz     pwaitloop
     nop
     nop
     ldy            ; restore return address
     str            ;
     sta  procstartconv  ; start a conversion

waitfordata
     lda  statusreg
     and  srdavflagloc
     jmpz waitfordata
     nop
     nop
     sta  adclrdav  ; first clear the dav flag
     lda  addata         ; get the A-D data
     jmp  @R        ; return
```

APPENDIX B - DSP CODE

```
;*****************************************************************
;  routine to read eeprom data
;  start with eeprom add in acc, return with data in acc

readeeprom
     sty            ; save eeprom add
     ldr
     sta  temp      ; save return address
     lxbi eepromframe
     stx            ; create eeprom frame
     ldy            ; get eeprom address to read
     sta  @X,eeadd
     lxbi eereadcom
     sta  @X,eecom
     jsr  eeprom
     lda  temp
     str
     lda  @X,eedata ; leave eeprom data in acc
     jmp  @R


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

wait1u
     lxbi 12        ; wait about a usec (doesnt count jsr)
waitloop1u
     sub  oneloc
     jmpnz     waitloop1u
     nop
     nop
     jmp  @R
```

APPENDIX B - DSP CODE

```
;*****************************************************************

geneeclk

     lxbi 0         ; set clock low
     sta  eeclk
     lxbi 24        ; wait about 2 usec
geneeloop1
     sub  oneloc
     jmpnz     geneeloop1
     nop
     nop
     lxbi 1         ;set clock high
     sta  eeclk
     lxbi 24        ;wait about 2 usec
geneeloop2
     sub  oneloc
     jmpnz     geneeloop2
     nop
     nop
     jmp  @R        ; return


eeprom
     ldr
     sta  @X,eereturn    ; save return address
     lxbi 0
     sta  eecs           ; set all outs low to start
     sta  eedi
     sta  eeclk
     lxbi eecommask;
     sta  @X,eemask
     lxbi 1
     sta  eecs
eecomloop
     lda  @X,eecom
     and  @X,eemask
     jmpz eecomz
     lxbi 1
     nop
     sta  eedi
     jmp  eecomboth
```

APPENDIX B - DSP CODE

```
eecomz
     lxbi 0
     sta  eedi
eecomboth
     jsr  geneeclk
     lda  @X,eemask
     rcr
     sta  @X,eemask
     jmpnz     eecomloop
     lxbi eeaddmask
     nop
     sta  @X,eemask
     nop
eeaddloop
     lda  @X,eeadd
     and  @X,eemask
     jmpz eeaddz
     lxbi 1
     nop
     sta  eedi
     jmp  eeaddboth
eeaddz
     lxbi 0
     sta  eedi
eeaddboth
     jsr  geneeclk
     lda  @X,eemask
     rcr
     sta  @X,eemask
     jmpnz     eeaddloop
     lxbi eewritecom
     sub  @X,eecom  ; check if write
     jmpz eewrite   ; if not, start the read
     nop
     lxbi 0         ; data part of the loop
     sta  @X,eetemp
     lda  msbw      ; 08000h
     sta  @X,eemask
```

```
eereaddataloop
     jsr  geneeclk
     lda  eedo
     and  oneloc
     jmpz eereaddataz
     lda  @X,eetemp
     or   @X,eemask
     sta  @X,eetemp
eereaddataz
     lda  @X,eemask
     rcr
     sta  @X,eemask
     jmpnz     eereaddataloop
     nop
     lda  @X,eetemp
     sta  @X,eedata
     jmp  eecleanup ; done with read, go cleanup

eewrite
     lda  msbw      ; 08000h
     sta  @X,eemask
     nop
eewritedataloop
     lda  @X,eedata
     and  @X,eemask
     jmpz eewritedataz
     nop
     lxbi 1
     sta  eedi
     jmp  eewritedataboth
eewritedataz
     lxbi 0
     sta  eedi
```

```
eewritedataboth
     jsr  geneeclk
     lda  @X,eemask
     rcr
     sta  @X,eemask
     jmpnz     eewritedataloop
     nop
     lxbi 0
     sta  eecs      ; clear and set eecs to poll busy bit
     jsr  wait1u
     lxbi 1
     sta  eecs
     jsr  wait1u
eewritewaitfordone  ; note that if writes are not enabled
     lda  eedo      ; and you do an eeprom write
     and  oneloc    ; you will hang here forever
     jmpz eewritewaitfordone  ;(until rev b with pullup on eedout)
     nop
eecleanup
     lda  @X,eereturn    ; restore return address
     str
     lxbi 0
     sta  eecs      ; set all outs low when done
     sta  eedi
     sta  eeclk
     jmp  @R        ; return
```

```
;****************************************************************

doeeprom
     lxbi eepromframe
     stx
     lda  eepromtestadd
     sta  @X,eeadd
     lda  eepromtestcom
     sta  @X,eecom
     lda  eepromtestdatain
     sta  @X,eedata
     jsr  eeprom
     lda  @X,eedata
     sta  eepromtestdataout
     lxbi 0
     sta  doeepromflag
     jmp  mainloop


testram
     ldr
     sta  temp      ; save return address
     lxbi 0
     sta  led
     lda  zeroloc
     sta  mtestmask
     jsr  doramtest
     lxbi 1
     sta  led
     lda  minusoneloc
     sta  mtestmask
     jsr  doramtest
     lxbi 0
     sta  led
     lda  testAAAA
     sta  mtestmask
     jsr  doramtest
     lxbi 1
     sta  led
     lda  test5555
     sta  mtestmask
     jsr  doramtest
     lxbi 0
```

APPENDIX B - DSP CODE

```
sta  led
     lda  temp
     str
     jmp  @R
doramtest
     lxbi      0
     stx
     sta  fifoclr   ; clear the fifo
fillseq
     xor  mtestmask
     sta  fifopush
     sxw
     nop
     nop
     nop
     nop
     sta  fifopush
     ldx
     add  oneloc
     stx
     jmpnz     fillseq
     nop
     nop
     lxbi 0
     stx
testseq
     xor  mtestmask
     sub  fifopop
     sta  fifopop
     jmpnz     memerror
     nop
     ldx
     xor  mtestmask
     sxw
     sub  fifopop
     sta  fifopop
     jmpnz     memerror
     nop
     nop
     ldx
     add  oneloc
     stx
     jmpnz     testseq
```

```
nop
     nop
     jmp  @R        ; return

memerror
     lda  temp
     str
     lxbi 1
     sta  led
     nop
     jmp  @R        ; return
```