AD484

Programmer's guide

4DSP Inc.
955 S Virginia Street, Suite 214, Reno, NV 89502, USA.
Email: support@4dsp.com

Table of Contents

1 Introduction

The programmer's guide describes the API functions for the AD484 as well as the firmware settings implemented in each FPGA . This guide is intended to be used in conjunction with the AD484 user manual.

2 API functions

2.1 AD484_Arm

Arms a previously opened AD484 device.

Syntax
int AD484_API AD484_Arm (AD484_HANDLE *dev*)

Parameters:
dev the handle to the device to be armed.

Returns:
0 on success, negative on failure.

2.2 AD484_BeginRead

Starts an asynchronous read from the AD484.

Syntax

int AD484_API AD484_BeginRead (AD484_HANDLE *dev*, char * *buffer*, int *count*, unsigned * *status*, unsigned * *temp*, AD484_OVL * *ovl*)

Parameters:

dev the handle to the device to start reading from.

buffer pointer to the memory where the data will be stored. Note that the data will not be valid until AD484_EndRead is called.

count the size in bytes of the requested amount of data.

status a pointer to the memory location where the status information is to be stored. If this pointer is NULL, no status information will be retrieved from the board. Note that the data will not be valid until AD484_EndRead is called.

temp a pointer to the memory location where the board temperature is to be stored. If this pointer is NULL, no temperature information is retrieved from the board. Note that the data will not be valid until AD484_EndRead is called.

ovl a pointer to an **AD484_OVL** structure that needs to be used in AD484_EndRead to complete the transfer.

Returns:

0 on success, negative on failure.

2.3 AD484_Close

Close a previously opened AD484 device.

Syntax
void AD484_API AD484_Close (AD484_HANDLE *dev*)

Parameters:
dev the handle to the device to be closed.

Returns:
N/A.

2.4 AD484_Configure

Configure an AD484. See enumerators below.

Syntax

int AD484_API AD484_Configure (AD484_HANDLE *dev*, unsigned *channel*, unsigned *triggerMode*, unsigned *triggerInterval*, unsigned *clockSource*, unsigned *clockFrequencyMHz*, unsigned *samplingMode*, unsigned *burstLength*, unsigned *numberOfBursts*, unsigned *decimationFactor*, unsigned *twosComplement*, unsigned *fullScale*)

Parameters:

dev the handle to the device to be configured.

channel the channel to use

triggerMode the triggermode on which the AD484 will trigger.

triggerInterval the trigger interval specified as <triggerInterval> x 20 ns.

clockSource the clock source used as the sample clock.

clockFrequencyMHz the clock frequency generated when internal clock is selected as the clock source.

samplingMode the mode in which to sample the data; ie. burst or continuous.

burstLength the number of samples in a burst. In continuous mode, a multiple of burstLength is acquired.

numberOfBursts the number of bursts to acquire.

decimationFactor the factor with which to decimate the data.

twosComplement if non-zero, the output data is in two's complemt surrounding zero. Else the data is unsigned and offset with 2048.

fullScale select the scale of the ADC.

Returns:

0 on success, negative on failure. Parameter related failures are listed below:

INVALID_CHANNEL = -101, INVALID_TRIGGER_MODE = -102,
INVALID_TRIGGER_INTERVAL = -103, INVALID_CLOCK_SOURCE = -104,
INVALID_CLOCK_FREQUENCY = -105, INVALID_SAMPLING_MODE = -106,
INVALID_BURST_LENGTH = -107, INVALID_NUMBER_OF_BURSTS = -108,
INVALID_DECIMATION_FACTOR = -109

Enumerators

channel: CHAN_A, CHAN_B, CHAN_AB

triggerMode: TRIG_SW, TRIG_FALLING_EDGE_A, TRIG_RISING_EDGE_A, TRIG_LOW_LEVEL_A, TRIG_HIGH_LEVEL_A, TRIG_FALLING_EDGE_B, TRIG_RISING_EDGE_B, TRIG_LOW_LEVEL_B, TRIG_HIGH_LEVEL_B

clockSource: CLOCK_INTERNAL, CLOCK_EXTERNAL_A, CLOCK_EXTERNAL_B

samplingMode: SAMPLE_CONTINUOUS, SAMPLE_BURST

2.5 AD484_DecodeStatus

Decode a status word.

Syntax

int AD484_API AD484_DecodeStatus (unsigned *status*, struct AD484_SR * *result*)

Parameters:

status the status word to be decoded.

result pointer to an **AD484_SR** structure that will contain the different status fields on completion.

Returns:

0 on success, negative on failure.

2.6 AD484_Read

Performs a synchronous read from the AD484.

Syntax

int AD484_API AD484_Read (AD484_HANDLE *dev*, char * *buffer*, int *count*)

Parameters:

dev the handle to the device to read from.

buffer pointer to the memory where the data will be stored

count the size in bytes of the requested amount of data.

Returns:

0 on success, negative on failure.

Requirements

Data must be 8 byte aligned and preferably 4K aligned. This can be achieved as shown in the following C code example:

```
int count = 32768;
int alignmentBytes = 8;
char *P = malloc(count + alignmentBytes – 1);
chat *pAligned = (char*)(((unsigned long)p + alignmentBytes – 1) &
~(alignmentBytes – 1);
.
.
AD484_Read(handle, pAligned, count);
.
free(p);
```

2.7 AD484_DecodeTemperature

Decode a temperature word.

Syntax
int AD484_API AD484_DecodeTemperature (unsigned *temp*, float * *result*)

Parameters:
temp the temperature word to be decoded.
result pointer to a float that will contain the temperature in Celsius on completion.

Returns:
0 on success, negative on failure.

2.8 AD484_Disarm

Disarms a previously opened AD484 device.

Syntax
int AD484_API AD484_Disarm (AD484_HANDLE *dev*)

Parameters:
dev the handle to the device to be armed.

Returns:
0 on success, negative on failure.

2.9 AD484_EndRead

Completes previously started asynchronous transfer.

Syntax

int AD484_API AD484_EndRead (AD484_HANDLE *dev*, AD484_OVL *ovl*)

Parameters:

dev the handle to the device on which the transfer is to be completed.

ovl the **AD484_OVL** structure that was returned by the AD484_BeginRead call that started the asynchronous transfer that is to be completed.

Returns:

0 on success, negative on failure.

2.10 AD484_GetStatus

Obtain the AD484 status synchronously.

Syntax
int AD484_API AD484_GetStatus (AD484_HANDLE *dev*, unsigned * *status*)

Parameters:
dev the handle to the device to get the status from.
status pointer to the memory location where to store the status information.

Returns:
0 on success, negative on failure.

2.11 AD484_GetTemperature

Obtain the AD484 temperature synchronously.

Syntax
int AD484_API AD484_GetTemperature (AD484_HANDLE *dev*, unsigned * *temp*)

Parameters:
dev the handle to the device to get the temperature from.
temp pointer to the memory location where to store the temperature information.

Returns:
0 on success, negative on failure.

2.12 AD484_Open

Open a AD484 device and obtain a handle that is usable with the AD484 functions.

Syntax
AD484_HANDLE AD484_API AD484_Open (int *devno*)

Parameters:
devno the device number to open.

Returns:
an AD484_HANDLE, which is INVALID_HANDLE_VALUE on failure.

2.13 AD484_Reset

Reset an AD484 device. This will restore the device into a defined state. All previously set settings will be lost.

Syntax

int AD484_API AD484_Reset (AD484_HANDLE *dev*)

Parameters:

dev the handle to the device to be reset.

Returns:

0 on success, negative on failure.

3 Structures

The following structures are defined in FM480_ADC290.h

3.1 AD484 Overlapped I/O Structure

```
typedef struct tagAD484_OVL
{
  OVERLAPPED ovl;
  struct tagAD484_OVL *next;
} *AD484_OVL;
```

3.2 AD484 Status Register Structure

```
struct AD484_SR {
  unsigned long BO  : 1;
  unsigned long DOA : 1;
  unsigned long DOB : 1;
  unsigned long DOC : 1;
  unsigned long DOD : 1;

  unsigned long _p0 : 13;
  unsigned long CC  : 16;
};
```

BO: Buffer overflow detected if 1
DOA: Data over-range channel A detected if 1
DOB: Data over-range channel B detected if 1
DOC: Data over-range channel A detected if 1
DOD: Data over-range channel B detected if 1

CC: Clock counter. Convert to MHz with 50. * CC / 8192.

4 FPGA firmware

The FPGA firmware implemented by default on the AD484 is described below. Because of the modularity of the design a user can easily modify this reference design in order to implement for example some signal processing tasks on the incoming digitized samples.

4.1 Commands to AD484

Commands are transmitted to the FPGA via a 32-bit mailbox. The commands are defined as follows:

Command	Mailbox value	Description
Update AD484 settings	0x00000001	Upon reception of this command, the AD484 will read the registers containing the settings.
Arm AD484	0x00000002	This command arms the AD484 for starting data acquisition. Data acquisition will start as soon as this command is received if the trigger source was set to be a software trigger.
Disarm AD484	0x00000003	This command disarms the AD484 and stops acquiring samples and offloading them to the host as soon as it is received. This command should be followed by a reset command to the card.
Acknowledge reading the status register	0x00000004	This command is sent to the FPGA when the host has read the status register after it received an interrupt in the form of a *status register alert*.

4.2 Commands from FPGA

These commands are transmitted to the software from the FPGA via a 32-bit mailbox. The commands are defined as follows:

Command	Mailbox value	Description
Status alert	0x00000002	This interrupt to the host is set when the status register requires the attention of the host program. This interrupt will be enabled only if bit 16 EI of register 0 is set to 1. Please note that this interrupt is not set in the case the AD484 routinely updates the status register on a 10 seconds basis.

4.3 Settings

Settings are passed from the host program to AD484 via the Custom registers. The FPGA B will read the Custom Registers contents and update its settings when it has received the corresponding command. Please note that settings can be read back by the host software at any time.

Custom register 0 : trigger, clock control and decimation factor

31 17	16	15 14 13 5	4	3	2	1	0
DF	EI	CDS	CMS	CSD	CSC	CSB	CSA	NM/TM

NM/TM: Normal mode/ Test mode. If 1, the normal mode is enabled. If 0, the test mode is enabled and the AD484 uses its own internal clock (100MHz) to generate and buffer test samples (clock settings are ignored). Please note that in the case the test mode is used the following parameters must be set:

- Software trigger only
- Data source must either be the counter or linear feedback shift register

CSA: clock source channel A. b0 = External, b1 = on-board synthesizer

CSB: clock source channel B. b0 = External , b1 = on-board synthesizer

CSC: clock source channel C. b0 = External , b1 = on-board synthesizer

CSD: clock source channel D. b0 = External , b1 = on-board synthesizer

CDS: clock division setting (in decimal $0 \leq CDS \leq 3$)

CMS: clock multiply setting (in decimal $200 < CMS < 475$)

On board clock frequency (clock source is on-board AD484 clock only) = 16MHz x CMS / $2^{(CDS+4)}$

Example: CMS = 250 and CDS = 1 then clock frequency = 125MHz

EI: Enable Interrupt. Does not generate an interrupt to the host in the case of a status alert if 0, generates an interrupt to the host in the case of a status alert if 1

DF: decimation factor. From 0 to 2^{15} -1 . If the decimation factor is equal to 0 or 1, there is no decimation. If the decimation factor is equal to n (with n>1) then the AD484 will acquire only every n^{th} sample.

Custom Register 1: channels, buffering and burst length

31 ... 10	9	8	7	6 ... 5	4	3	2	1	0
BL	CM	FS	DM	DS	CS	CDE	CCE	CBE	CAE

CAE: channel A enable. Data acquisition from channel A is enabled if 1. Disabled if 0

CBE: channel B enable. Data acquisition from channel B is enabled if 1. Disabled if 0

CCE: channel C enable. Data acquisition from channel C is enabled if 1. Disabled if 0

CDE: channel D enable. Data acquisition from channel D is enabled if 1. Disabled if 0

CS: Clock source. Clock source for the status register clock counter. Channel A if 0, Channel B if 1**.**

DS: Data source. Data comes from the ADC if b00, from a counter if b01, from a linear feedback shift register if b10 .

DM: Data mode. If 1, samples from the ADCs are coded in 2's complement format. If 0 samples are coded in binary format.

FS: Full scale. If 1, full scale is 2.4Vp-p differential. If 0, full scale is 1.15Vp-p differential.

CM: Continuous mode. Once a trigger has been detected or enabled, the sampling of data is continuous (no lost samples) until <Number of Burst> * <Burst length> samples have been digitized.

BL: Burst length (from 4 to 2^22-4 in multiples of 4). When detecting a trigger, the AD484 will acquire from each channel that is enabled the number of samples specified by the burst length setting.

Please note that a <burst length> of 0 is equivalent to infinity.

Custom Register 2: Number of bursts

23 ... 0
NB

NB: number of bursts. From 1 to 2^{24}-1 The number of samples read back by the host software program over one or multiple DMAs is : <Number of Burst> * <Burst length>.

Please note that a <Number of Burst> of 0 is equivalent to infinity.

Custom Register 3: Trigger settings

31 … 6	5	4	3	2	1	0
TI	TM		TSD	TSC	TSB	TSA

TSA: trigger source channel A.

- Software trigger if b0
- External trigger if b1

TSB: trigger source channel B.

- Software trigger if b0
- External trigger if b1

TSC: trigger source channel C.

- Software trigger if b0
- External trigger if b1

TSD: trigger source channel D.

- Software trigger if b0
- External trigger if b1

TM: trigger mode for external trigger.

- Falling Edge detection if b00
- Rising Edge detection if b01
- Low level detection if b10.
- High level detection if b11.

TI: trigger interval. Number of clock cycles at 31.25MHz (32ns) between starting two consecutive burst acquisitions. Maximum value is 2^{26}-1 ≈ 2.15 seconds. This setting is ignored if the trigger source is set to external trigger.

Minimum value is 16.

Please note that if continuous data acquisition is required in the software trigger mode, having <trigger interval> = <burst_length> is not the proper way to specify it and will result in faulty behaviour. A continuous data acquisition must be specified by setting the register 1 CM bit to 1.

4.4 Status register

The status registers (custom register 0x1F) contains information about the firmware status. It is recommended that the software reads the status register before performing any action.

31 ... 16		4	3	2	1	0
CC		DOD	DOC	DOB	DOA	BO

BO: buffer overflow. If 1, it indicates that the AD484 memory buffer overflowed, samples may be corrupted / lost.

DOA: data over range channel A. If 1, it indicates that some samples digitized on channel A were over the voltage range of the ADC.

DOB: data over range channel B. If 1, it indicates that some samples digitized on channel B were over the voltage range of the ADC.

DOC: data over range channel C. If 1, it indicates that some samples digitized on channel C were over the voltage range of the ADC.

DOD: data over range channel B. If 1, it indicates that some samples digitized on channel D were over the voltage range of the ADC.

CC: Clock counter. Counts the number of clock cycles at the ADC frequency that happen in a 163.64µs time lapse. The ADC frequency can be calculated as follows:

Fadc = 50MHz * <CC value > / 8192.

The status register is updated every 10 seconds or as soon as an event that requires user attention is recorded:

- clock frequency calculated from CC varied of more than 2% in 163.84us time lapse.
- A transition from low to high occurred on DOA, DOB or BO (theses bits are automatically cleared when the host acknowledges reading the status)

4.5 Data formatting

When coming from the AD484 the digitized samples are packed on 64-bit.
In the case only one channel is enabled the samples are packed as follows:

59	48	43	32	27	16	11	0
Sample i+3		Sample i+2		Sample i+1		Sample i	
Sample i+7		Sample i+6		Sample i+5		Sample i+4	

In the case both channels are enabled the samples are packed as follows:

59	48	43	32	27	16	11	0
Sample i+1 ch B		Sample i+1 ch A		Sample i ch B		Sample i ch A	
Sample i+3 ch B		Sample i+3 ch A		Sample i+2 ch B		Sample i+2 ch A	

In the case all channels are enabled then the samples are packed as follows:

59	48	43	32	27	16	11	0
Sample I ch D		Sample I ch C		Sample i ch B		Sample i ch A	
Sample i+1 ch D		Sample i+1 ch C		Sample i+1 ch B		Sample i+1 ch A	

Please note that if two or four channels are enabled, the sampling rate and trigger must be exactly the same to make sure that the channels are interleaved properly when sent to the host system.

When the data is coded in 2's complement format (register 1 bit 5 set to 1), the sign bit is extended to the two unused MSB for each sample, thus making the data in a 16-bit integer format ready for processing.