SADC10/18/20/30 - Communication protocol

Document revision 23rd March 2004

SARA di Mariotti Gabriele & C snc – Perugia info@sara.pg.it www.sara.pg.it

Overview

This document describe the communication protocol of the SADC10/18/20/30 rispectively: 16 bit 4 channel, 18 bit 4 channel, 24 bit 3 channel, 16 bit 16 channel. In order to provide as complete information as possible SEISMOWIN datalogger software is used as example of "client" application of the data "served" by the A/D board.
SEISMOWIN and the SADC10/18/20/30 board receives and transmits data using a simplified protocol in order to increase the data rate at low baud rates.
Previous versions of the SADCxx boards was supporting baud rate of 9600 and 14400, now all the boards are produced with 38400 baud. **All boards supporting only 9600 and/or 14400 baud must be considered obsolete, firmware upgrade is recomended for that boards**. In the next future the protocol will be modified in order to make all boards supports many standard baud rate with a special command to set it. In any case the official standard baud rate will remain 38400 baud and all the boards with factory setting are running at 38400 baud.

The communication protocol is bidirectional.
In some boards the PC can adjust acquisition speed indipendently for each channel.
The A/D board send to the PC its absolute time and the sampled data.
The PC can set the TIME of the board in any moment.
The COM port must be opened in binary mode, without handshake, 8 bits of data, 1 stop bit, no parity.

How to detect the a/d card speed

Two ways can be used. One is to read for some seconds the data of the TIME at 14400 and determine if there are transmission errors on the packet received. (The board always transmits the TIME tag). If there are errors the receiving procedure should try to change the baud rate at 38400 baud and check for error at this speed. If it is able to receive the TIME data correctly the correct baud rate has been found.
One another way is to ignore the TIME data and transmit to the a/d board a FIRMWARE VERSION read command at 14400 and after at 38400 until a knew version number or coherent data for version number are obtained.
As already explained boards supporting only 9600 and 14400 baud are obsolete. The new boards run at 38400. For the next development step the boards comm speed will be adjustable according to a restricted options of baud rates.

Supported commands

0x81 – read FIRMWARE VERSION
0x82 – set GMT correction setting
0x83 – set TIME
0x84 – set SAMPLING RATE
0x85 – set CLOCK compensation
0x86 – read EEPROM
0x87 – set DATE

Firmware version request
This command is necessary to know the firmware version of the a/d board.
According to the firmware versions some features can change, for example the number of bits supported for the a/d conversion or other features like the Sample per rates. The following table explain the changes from version to version.

Version number	*Baud rate*	*SPS rate*	*Bits Nr.*	*Timekeeping support*
1.51	14400	100	16	HH:MM:SS
1.61	38400	200	16	HH:MM:SS
1.62	38400	200	16	YY/MM/DD HH:MM:SS
1.80	38400	200	18	HH:MM:SS
1.81	38400	200	18	YY/MM/DD HH:MM:SS
2.00	38400	200	24	YY/MM/DD HH:MM:SS
3.00	38400	200	16	YY/MM/DD HH:MM:SS

Command syntax: 0x81 0x00 0x00 0x00 0x00 0x00

```
' example
PRINT #1,CHR$(VAL("&H81"));CHR$(0);CHR$(0);CHR$(0);CHR$(0);CHR$(0)
```

The answer could be: V181 meaning version 1.81 (the answer is in ASCII code)

Appropriate changes to the driving procedure of the boards must be applied if the support is necessary for all the released versions.

GMT correction setting
The a/d board can receive the accurate time from any DCF77 source, real or emulated.
This parameter allow to modify the received time according to the DCF77 source.
DCF77 always transmit Rome time with automatic changeover of daylight time and summer time according to the international regulation. If a radio DCF77 source is used a value of –1 must be applied as GMT or UTC correction.
If an emulated DCF77 time already corrected to UTC time a value of 0 must be applied meaning "no correction".
For firmware version not supporting datekeeping any time correction can be used if a particular time zone is desidered.
The parameter support –23 to +23 time correction range.

Command syntax: 0x82 GMT 0x00 0x00 0x00 0x00

The GMT correction should be provided in two's complement binary format, (signed integer of 8 bit).
If the correction is positive you can send 1, 2 or 3 and so on… if negative you must send –1 (255 in two's complement format), or –2 (254 in two's complement format). If no correction is needed (because you are using a GPS source) you can send 0 (zero).

Note: Considering the DCF77 is always transmitted with +1 time offset (Germany Local time) you should always apply –1 to have the UTC time. UTC adjustement is provided to mantain free the user to set any desidered time. No variations are needed to adjust the time for the daylight save time changeover because the decoder recognize by itself the variation and apply the properly +1 correction. If a GPS-DCF converter is used the UTC adjustment can be used and the daylight changeover is not considered.

If the data has been correctly received the a/d card replies with one acknowledge byte: 0xF8, chr$(248)

Datekeeping considerations

As you can already read from the firmware version command, not all firmware versions support complete datekeeping. Some version only keep hours, minutes and seconds. For these boards time changeover from 23:59:59 to 00:00:00 should be detected by the driving software and the date should be updated +1 day according to the calendar.

For version from 1.61 and 1.81 full datekeeping has been added and year, month and day support is present. The board is also able to pick the accurate date (not only the time) if a GMT correction of 0 is used. In this case a reliable DCF77 source must be used (emulated) because no error controls are executed on the date information.
Considering also a GPS (emulated in DCF77) source always provide UTC date and time so time shifting with values different from zero would be not acceptable because the supplied time would be incoherent to the supplied date generating the needs to executes –1 day or +1 day internal corrections.

So the rules applied to the date/time decoded by the DCF77 input are the following:

1) If a GMT correction of 0 is used (tipically using an emulated reliable DCF77 source using a GPS already providing an GMT time), date and time are decoded and no hour shift is applied.

2) If a GMT correction different from 0 is used (tipically using a real DCF77 source transmitting +1 time) no automatic date-picking is executed by the DCF77 signal but only date-keeping and of course the time update but only for hour minutes and seconds.

3) If a GMT of 0 is used the date will be updated from the DCF77 source. If GMT <> 0 the user (or the application software) must provide the initial date set.

Date changeover will be always accurate if the system is kept syncronized with an accurate time. Date and time picking from the accurate DCF77 or GPS source are executed only in the odd minutes so the last picking of the day can occurs at 23:59:00 and the first of the day at 00:01:00 so the day changeover is always executed by the board without interference of the external source. If not a time tear or a time lag due to the a/d board crystal drift would cause an error in the day changeover.

If a real radio DCF77 source is used with a GMT correction of 0 date picking could be not reliable, because no error checking is executed on the year, month and day data packet, error checking is executed only for hour, minutes and seconds.

Time Adjust
You can adjust the board's time using the following command:

Command syntax: 0x83 sec min hour 0x00 0x00

At the issuing of the command the hundreds of seconds are cleared.

The parameters sec, min and hour must be send in binary, NOT in BCD or ASCII. For example to adjust the time to: 12:33:24 you must send:

```
' example
PRINT #1,CHR$(VAL("&H83"));CHR$(24);CHR$(33);CHR$(12);CHR$(0);CHR$(0)
```

If the data has been received correctly the card replies with one single acknowledge byte: 0xF8, chr$(248)

Date Adjust
You can adjust the board's date using the following command:

Command syntax: 0x87 year month day 0x00 0x00

The year offset is considered 2000.
The parameters year, month and day must be send in binary, NOT in BCD or ASCII. For example to adjust the time to: 2004/12/03 you must send:

```
' Example
PRINT #1,CHR$(VAL("&H83"));CHR$(4);CHR$(12);CHR$(3);CHR$(0);CHR$(0)
```

If the data has been received correctly the card replies with one single acknowledge byte: 0xF8, chr$(248)

Adjusting sampling rate
On power up the a/d card can send data or not, depending on the firmware.
If the firmware is previous the V.01.31 the a/d card not send data. If the firmware is higher than 1.31 the board retains the SPS selected the previous time and send data if, before powered down, it was sending data. The a/d card remember in a eeprom registers its last configuration of GMT and SPS for each channel. This allow the usage of standalone cards with a wired or radio modems, for example.

Version prior to the 1.60 are not able to samples at over 100 SPS.

To setup the sps rate you must send: 0x84 sps1 sps2 sps3 sps4 0x00

Where sps1..4 are the speed of the 4 channels. If a channel is not used, st this spsX to zero. If a channel is used setup it with the following formula:

spsX = 100 / SPS required

For example if you want set up the channel 1 at 20 SPS, 2 at 50 SPS, 3 at 25 SPS and turn of the 4th, set:

ex: PRINT #1,CHR$(VAL("&H84"));CHR$(5);CHR$(2);CHR$(4);CHR$(0);CHR$(0)

For firmware version 1.6, 1.61, 1.62, 1.80, 1.81, 2.00 and 3.00 (all capable of 200 SPS) the formula is:

spsX = 200 / SPS required

Immediately the card start to send datas at the various frequencies.

Notice: Version 2.00 and 3.00 have some difference to setup the sampling rate for the various channels please referr to the next page for details.

DATA RATE of the SADC20 24 bit board
The 24 bit board capable to record 3 channel simultaneusly have the limitation that it cannot samples the channels at different sample rates. This means that you have to set all the channel at the same sample rate. Anyway you can STOP a channel to issuing data setting its spsX to zero. By the way the SADC20 board offer a sampling at phisically ZERO skew time. So no phase shift is present in the sampling being all 3 channels sampled simultaneously.

DATA RATE of the SADC30 16 bit 16 channel board
The 16 channel board can record up 16 channels. While the SADC1x board can be adjusted to give different data rate for the 4 channel the 16 channel board can be adjusted to a single data rate for all channels. Anyway it is allowed to stop the unneeded channels. To adjust the SPS rate and enable or disable channels use the following rules.

To setup the sps rate you must send: 0x84 SPS enabL enabH 0x00 0x00

Where SPS is the data rate common to all channels and it is ruled by:

SPS = 200 / SPS required

As for the a 1.62 version 16 bit board (see previous paragraphs).

The bytes "enabL" and "enabH" enable and disable the channels to issuing data.
The 16 bits composing "enabL" and "enabH" are 1 for enable 0 to disable, MSB is channel 16, LSB is channel 1.
For example to turn on channel 1, 2 and 3 and channel 9 at 50 SPS the command is:

ex: PRINT #1,CHR$(VAL("&H84"));CHR$(4);CHR$(7);CHR$(1);CHR$(0);CHR$(0)

Decoding data from the A/D board

The board transmits packet of 4, 5, 6 or 9 bytes. The end of the packet is identified by a byte with the value greather than 240 (0xF0). As general rule the data greather or equal than absolute value of 128 are control bytes or special byte. Raw data bytes are lower than 128. This mean that the protocol uses only 7 bits of data for each byte. 4 Bytes are used to encode data of samples from a 16 bit or 18 bit board; 5 bytes are used to encode boards with 24 bits of resolution. 6 bytes are used to encode time in boards that don't have the datekeeping in the real time clock. 9 bytes are used on the boards having the complete time and date keeping.

Time decoding

TIME	0x81	sec	min	hour	extra	0xFF

For version 1.62 and 1.81 the TIME command contains also the date information as following:

TIME	0x81	year	month	day	sec	min	hour	extra	0xFF

The TIME is transmitted on time per second. It contains raw binary data. The extra byte of the TIME packet contains the data of the two auxiliary lines L1 and L2 available on the card. L1 is used to detect the DCF77 or GPS time information.

TIME packet EXTRA byte encoded as follows:

Bit0	Reserved
Bit1	Reserved
Bit2	Reserved
Bit3	L1 line status
Bit4	L2 line status
Bit5	SYNC received (remains = 1 for 6 seconds)
Bit6	Reserved
Bit7	Reserved

Data decoding on 16 or 18 bits boards

The data provided by boards with 16 or 18 bits resolution are encoded as follows:

Sample from CH1	0x82	low	high	extra/end
Sample from CH2	0x83	low	high	extra/end
Sample from CH3	0x84	low	high	extra/end
Sample from CH4	0x85	low	high	extra/end

Bytes *low* and *high* have the 7th bit always equal to 0.

Byte *Extra /end* is encoded as follows:

Bit0	7th of byte *low*
Bit1	7th of byte *high* (meaning the sign bit if a 16 bit protocol is used)
Bit2	bit 16th of the 18 data bits (if 18 bit board is used, otherwise = 1)
Bit3	bit 17th of the 18 data bits, meaning the sign bit (if 18 bit board is used, otherwise = 1)
Bit4	always 1
Bit5	always 1
Bit6	always 1
Bit7	always 1

After decoded you can find the digitized data expressed in two's complement format as provided by the A/D converter. The LOW and HIGH bytes needed to be completed because they are transmitted without the bit number 7 (counting bits from 0 to 7). Also bits number 16 and 17 are needed to be attached to the data word if a 18 bit protocol is used.
To do this, the fourth and last byte, is used to identify the end of the packet because is always greather or equal than 240 (0xF0). It is also the container of the bit number 7 of the *low* and *high* byte, and the bits number 16 and 17 if a 18 bits protocol is used). Due to the fact to be always greather or equal to 240d (0xF0) this byte indicates that the packet is terminated.

```
' Example in Basic Language assuming the following variable meaning:
' low = lower byte of the 16 bits
' high = high byte of the 16 bits
' bits = extra bits trasmitted with the packet Byte EXTRA/END
' 16/18 bits encoding example
high = high + (bits And 2) * 64            ' complete high byte
low = low + (bits And 1) * 128             ' complete low byte

IF board_type = AD_SADC16 THEN             ' 16 bit deconding
   value = high * 256 + low                ' Notice! Check the sign rules on your compiler!
END IF
IF board_type = AD_SADC18 THEN             ' 18 bit decondig
   temp = (high * 256 + low) + (bits And 4) ' complete the 18 bit long word
   IF bits And 4 THEN temp = temp + 65536
   IF bits And 8 Then                      ' adjust the sign
       value = -131072 + temp
   ELSE
       value = temp
   END IF
END IF
' variable named "value" now contains the final sample
```

Data decoding on 16 bit 16 channel board (V. 3.00)
The data provided by board with 16 bit resolution and 16 channel capability are encoded as follows:

Sample from CH1	0x82	low	high	extra/end
Sample from CH2	0x83	low	high	extra/end
Sample from CH3	0x84	low	high	extra/end
Sample from CH4	0x85	low	high	extra/end
Sample from CH5	0x86	low	high	extra/end
Sample from CH6	0x87	low	high	extra/end
Sample from CH7	0x88	low	high	extra/end
Sample from CH8	0x89	low	high	extra/end
Sample from CH9	0x8a	low	high	extra/end
Sample from CH10	0x8b	low	high	extra/end
Sample from CH11	0x8c	low	high	extra/end
Sample from CH12	0x8d	low	high	extra/end
Sample from CH13	0x8e	low	high	extra/end
Sample from CH14	0x8f	low	high	extra/end
Sample from CH15	0x90	low	high	extra/end
Sample from CH16	0x91	low	high	extra/end

It is easily identifiable that the decoding method is exact the same of the 4 channel board. The only difference is the 16 channel board issue a channel identifier greater than 0x85, from 0x86 to 0x91 identifing channel from 5 to 16.

Bytes *low* and *high* have the 7th bit always equal to 0.

Byte *Extra /end* is encoded as follows:

Bit0	7th of byte *low*
Bit1	7th of byte *high* (sign bit)
Bit2	always 1
Bit3	always 1
Bit4	always 1
Bit5	always 1
Bit6	always 1
Bit7	always 1

After decoded you can find the digitized data expressed in two's complement format as provided by the A/D converter. The LOW and HIGH bytes needed to be completed because they are transmitted without the bit number 7 (counting bits from 0 to 7).

```
' Example in Basic Language assuming the following variable meaning:
' low = lower byte of the 16 bits
' high = high byte of the 16 bits
' bits = extra bits trasmitted with the packet Byte EXTRA/END
' 16/18 bits encoding example
high = high + (bits And 2) * 64            ' complete high byte
low = low + (bits And 1) * 128             ' complete low byte

IF board_type = AD_SADC16 THEN             ' 16 bit deconding
   value = high * 256 + low                ' Notice! Check the sign rules on your compiler!
END IF

' variable named "value" now contains the final sample
```

Decoding data from the A/D board with 24 bit format
The data provided by boards with 24 bits resolution are encoded as follows:

Sample from CH1	0x82	low	middle	high	extra/end
Sample from CH2	0x83	low	middle	high	extra/end
Sample from CH3	0x84	low	middle	high	extra/end

After decoded you can find the digitized data expressed in two's complement format as provided by the AD converter. The LOW, MIDDLE and HIGH bytes needed to be completed because they are transmitted without the bit number 7 (counting bits from 0 to 7).
To do this, the fourth and last byte, is used to identify the end of the packet because is always greather or equal than 240 (0xF0). It is also the container of the bit number 7 of the *low* and *high* byte, and the bits number 16 and 17 if a 18 bits protocol is used).

Byte *Extra /end* is encoded as follows:

Bit0	bit nr 7 of byte *low*
Bit1	bit nr 7 of byte *middle*
Bit2	bit nr 7 of byte *high*
Bit3	always 1
Bit4	always 1
Bit5	always 1
Bit6	always 1
Bit7	always 1

For the fact to be always greather or equal to 240d (0xF0) this byte indicates that the packet is terminated.

```
' Example in Basic Language

' example assume that
' low = lower byte of the 24 bits
' middle = middle byte of the 24 bits
' high = high byte of the 24 bits
' bits = extra bits trasmitted with the packet Byte EXTRA/END

low = low + (bits And 1) * 128            ' complete low byte
middle = middle + (bits And 2) * 64       ' complete middle byte

' compute absolute value
tmp = (high * 65536 + middle * 256 + low)
'
' apply the sign
IF bits And 4 THEN
   value = -8388608 + tmp
ELSE
   value = tmp
END IF
' variable named "value" now contains the final sample
```

Error detection
This protocol is not 100% reliable on error detection and no error correction is possible. It has been conceived for data transmission at small distances (few meters) and at low speeds, so error correction and detection is not much important for good results. Counting the bytes received after one header (0x81 for TIME stamp or 0x82,0x83,0x84,0x85 for datas) and before the end data packet (any value >= 240) and comparing the count with the number of expected data for each packet, you can be quite sure if the packet is well received or not and, if not, discard it. Obviously, if your system experience a packet error the data for that second cannot be considered syncronized to the main clock and the time error will reflect the number of wrong packets multiplicating the time fraction 1/SPS of the channel experienced the error.

Crystal error compensation trim
The a/d board crystal's error can be compensated applying the appropriate with this command.
This command instruct the a/d board to compensate 1 hundred of second every X hundreds of seconds.
There is also the possibility to null the digital trimming and use an analogic compensation.

Command syntax: 0x85 low_trim med_trim high_trim dir_trim 0x00

```
' Example
PRINT #1,Chr$(Val("&h85")) + Chr$(low_trim) + Chr$(med_trim) + Chr$(high_trim) + Chr$(dir_trim) + Chr$(0)
```

If the data has been received correctly the card replies with one single acknowledge byte: 0xF8, chr$(248)

Notice 1
Check the package of your board to know if it is analogically compensated or digitally compensated. If it is analogically compensated it is a good thing to issue a 0x85 0xFF 0xFF 0xFF 0xFF 0x00 command to null the digital trimming at the board start.

Notice 2
If a SPS of 100 SPS or more is used with the digital compensation a variation of the SPS presents between two time tags can occurs due to the time compensation exeucuted in 1/100 of seconds. The data between two time tags could oscillates from 99 to 101 for 100 SPS and from 198 to 202 for 200 SPS. The SADC20 board is not affected by this problem.

Notice 3
Crystal compensation will be no longer supported on the new boards. New more stable and precise crystal will be used making the digital compensation unneeded after the hardware factory calibration. If your new board has an hardware crystal calibration please don't use this command or issue simply a null string as the following example:

```
PRINT #1,Chr$(Val("&h85")) + Chr$(255) + Chr$(255) + Chr$(255) + Chr$(255) + Chr$(0)
```

Internal EEPROM location read
The a/d board have an internal EEPROM memory that retains the settings during power off periods.
The data retained are: digital compensation, GMT correction, SPS settings

The memory map is the following:

0x00 GMT correction
0x01 compensation low
0x02 compensation mid
0x03 compensation high
0x04 compensation direction
0x05 sps ch1
0x06 sps ch2
0x07 sps ch3
0x08 sps ch4

Command syntax: 0x86 address 0x00 0x00 0x00 0x00

```
' Example
PRINT #1,Chr$(Val("&h86")) + Chr$(address) + Chr$(0) + Chr$(0) + Chr$(0) + Chr$(0)
```

If the command is correctly recognized the a/d board replies with a single byte containing the data found in the specified EEPROM address.