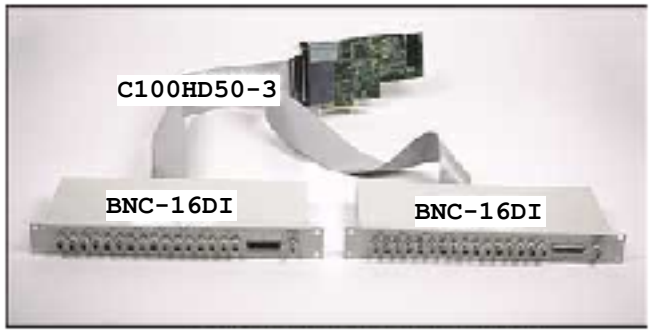
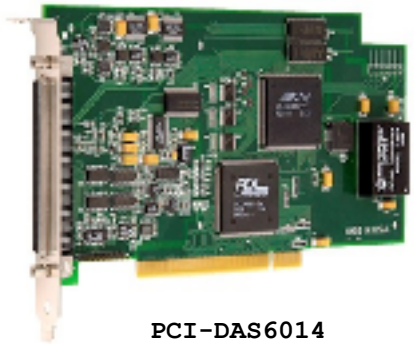
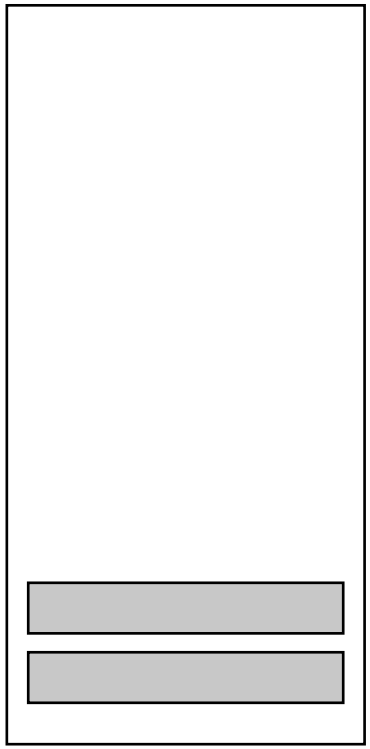
TAMA HDAQ 2013 version





PCI-DAS6014



Computer:
Intel Core2 Quad CPU
Q6600 2.4GHz
2.4 GHz, 2.00 GB RAM

OS:
Windows XP
Professional
Version 2002
Service Pack 3

PCI-DAS6014 x 2 boards
Measurement Computing

8 analog input in differential mode
with 16 bit resolution.

Input range: +-10 Volts

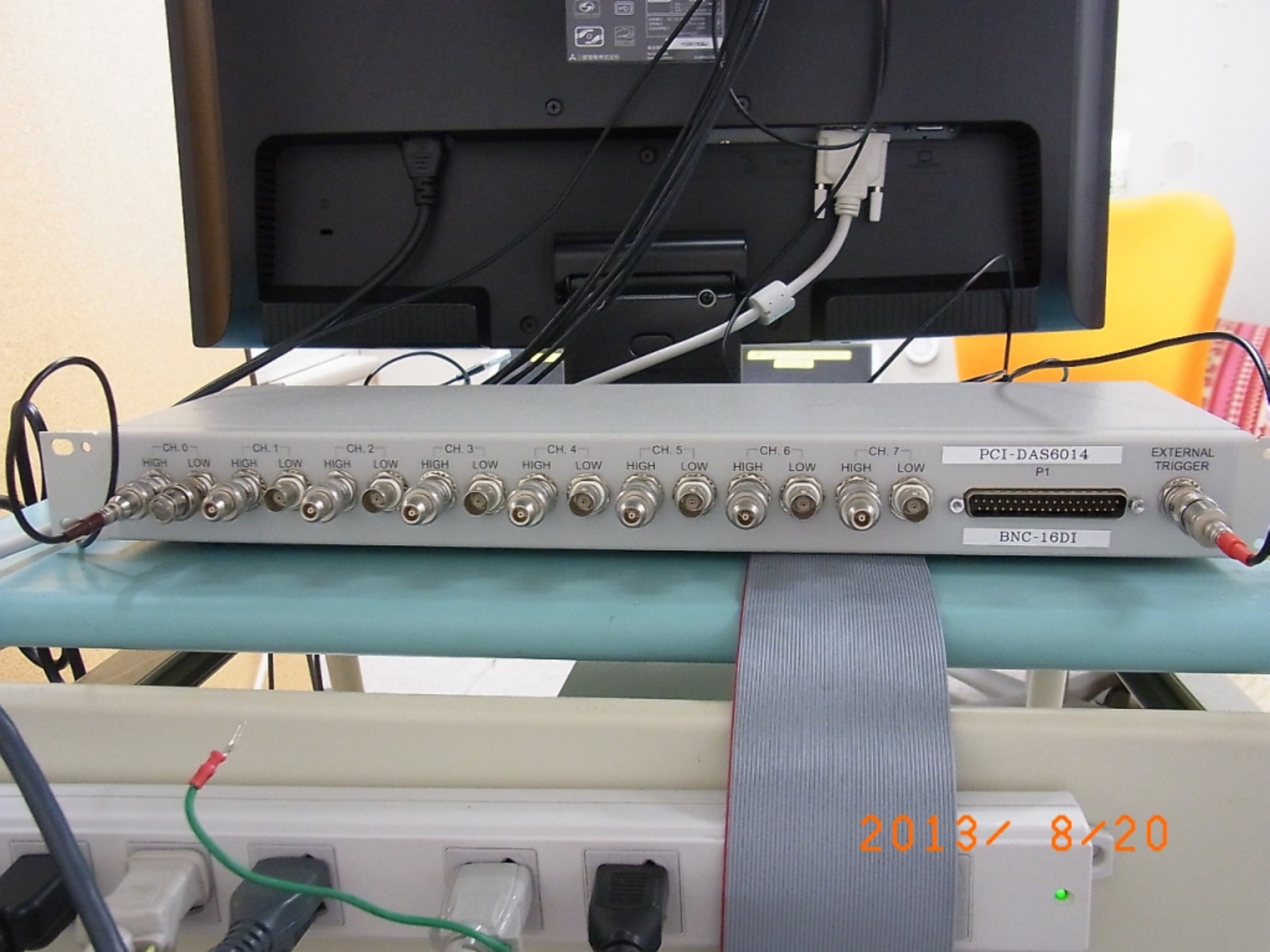

2013/ 8/ 2
Stuff Only
JOINTEX
smartoffice
smartschool

Tektronix
2013/ 8/ 2


MITSUBISHI
PCI-DAS6014
P1
EXTERNAL TRIGGER
BNC-16DI
PCI-DAS6014
P1
BNC-16DI
2013/ 8/ 2


自然科学研究機構
小額資産
〈天文台
資産番号 91001054853
取得日 20080128
品名 デスクトップパソコン(TAMAデータ取得用)
規格 DOSPARA PRIME MONARCH GX
POWER
HDD
RESET
intel
Core 2
Quad
inside
GRAPHICS BY
NVIDIA
OUT
IN
Prime
JOINTEX
2013/8/20


2013/ 8/20


CH. 0
HIGH
LOW
CH. 1
HIGH
LOW
CH. 2
HIGH
LOW
CH. 3
HIGH
LOW
CH. 4
HIGH
LOW
CH. 5
HIGH
LOW
CH. 6
HIGH
LOW
CH. 7
HIGH
LOW
PCI-DAS6014
P1
BNC-16DI
EXTERNAL
TRIGGER
2013/ 8/20

Specifications

Analog Input section:

A/D converter	Successive Approximation type. min 200k Samples/sec conversion rate
Resolution	16 bits, 1-in-65536
Number of channels	8 differential / 16 single ended, Software selectable
Ranges	+-10V, +-5V, +-500mV, +-50mV, Software selectable
A/D pacing	Internal counter - ASIC. Software selectable time base: * Internal 40 MHz, 50ppm stability * External source via AUXIN<0:5>, Software selectable.
Burst mode	Software selectable option, burst rate = 5us.
A/D gate sources	External digital; A/D GATE
A/D gating modes	External digital: Programmable, active high or active low, level or edge
A/D trigger sources	External digital; A/D START TRIGGER A/D STOP TRIGGER
A/D triggering modes	External digital; Software-configurable for rising or falling edge. Pre-/Post-trigger: Unlimited number of pre-trigger samples, 16 Meg post-trigger samples.
ADC Pacer Out	Available at user connector: A/D PACER OUT
RAM buffer size	8K samples
Data transfer	DMA (Direct Memory Access) Programmed I/O
DMA modes	Demand or Non-Demand using scatter gather
Configuration Memory	Up to 8K elements. Programmable channel, gain, and offset
Streaming-to-disk rate	200k Samples/sec, system dependent.

Accuracy

After 15-minute warm-up.
They are valid for operational temperatures within ±1° C of internal calibration temperature
and ±10° C of factory calibration temperature

Table 6-1 Absolute Accuracy

Range	Absolute Accuracy	
+-10V	+-29.4 LSB	8.983 mV
+-5V	+-13.1 LSB	2.003 mV
+-500mV	+-30.0 LSB	0.471 mV
+-50mV	+-45.2 LSB	0.069 mV

Table 6-2 Absolute Accuracy Components

Range	% of Reading	Offset (mV)	Noise + Quan. (mV)	Temp Drift (%/DegC)	Abs. Accuracy (mV)
+-10V	0.070	1.800	0.180	0.001	8.983 mV
+-5V	0.020	0.918	0.085	0.001	2.003 mV
+-500mV	0.070	0.109	0.012	0.001	0.471 mV
+-50mV	0.070	0.027	0.007	0.001	0.069 mV

Table 6-3 Differential non-linearity

All Ranges	+-0.5 LSB typ., +-1.0 LSB max.
No Missing Codes	16 bits, guranteed

Settling Time

Settling time is defined as the time required for a channel to settle to within a specified accuracy in response to a full-scale (FS) step. Two channels are scanned at the specified rate. A -FS DC signal is presented to Channel 1; a +FS DC signal is presented to Channel 0.

Condition	Range	Accuracy	
		+-2.0 LSB	+-4.0 LSB
Same range to same range	+-10V	5us typ	*
	+-5V	5us max	*
	+-500mV	5us typ	*
	+-50mV	*	5us typ

Parametrics

Max working voltage		+-11V
CMRR @ 60Hz	+-10V range	85dB
	+-5V	85dB
	+-500mV	93dB
	+-50mV	93dB
Small signal bandwidth, all ranges		424 kHz
Input coupling		DC
Input Impedance		100G Ohm in normal operation 2k Ohm typ in powered off or overload condition
Input bias current		+-200pA
Input offset current		+-100pA
Absolute maximum input voltage		+-25V powered on, +-15V powered off. Protected inputs: CH<0:15>IN AISENSE
Cross-talk	Adjacent Channels:	-75dB
	All other Channels:	-90dB

Noise Performance

Table 6-4 below summarizes the noise performance for the PCI-DAS6014.
Noise distribution is determined by gathering 50K samples with inputs tied to ground at the user connector. Samples are gathered at the maximum specified single-channel-sampling rate. Specification applies to both single-ended and differential modes of operation.

Table 6-4 Analog Input Noise Performance

Range	Typical Counts	LSB rms
+-10V	7	0.9
+-5V	7	0.9
+-500mV	11	1.1
+-50mV	45	6.7

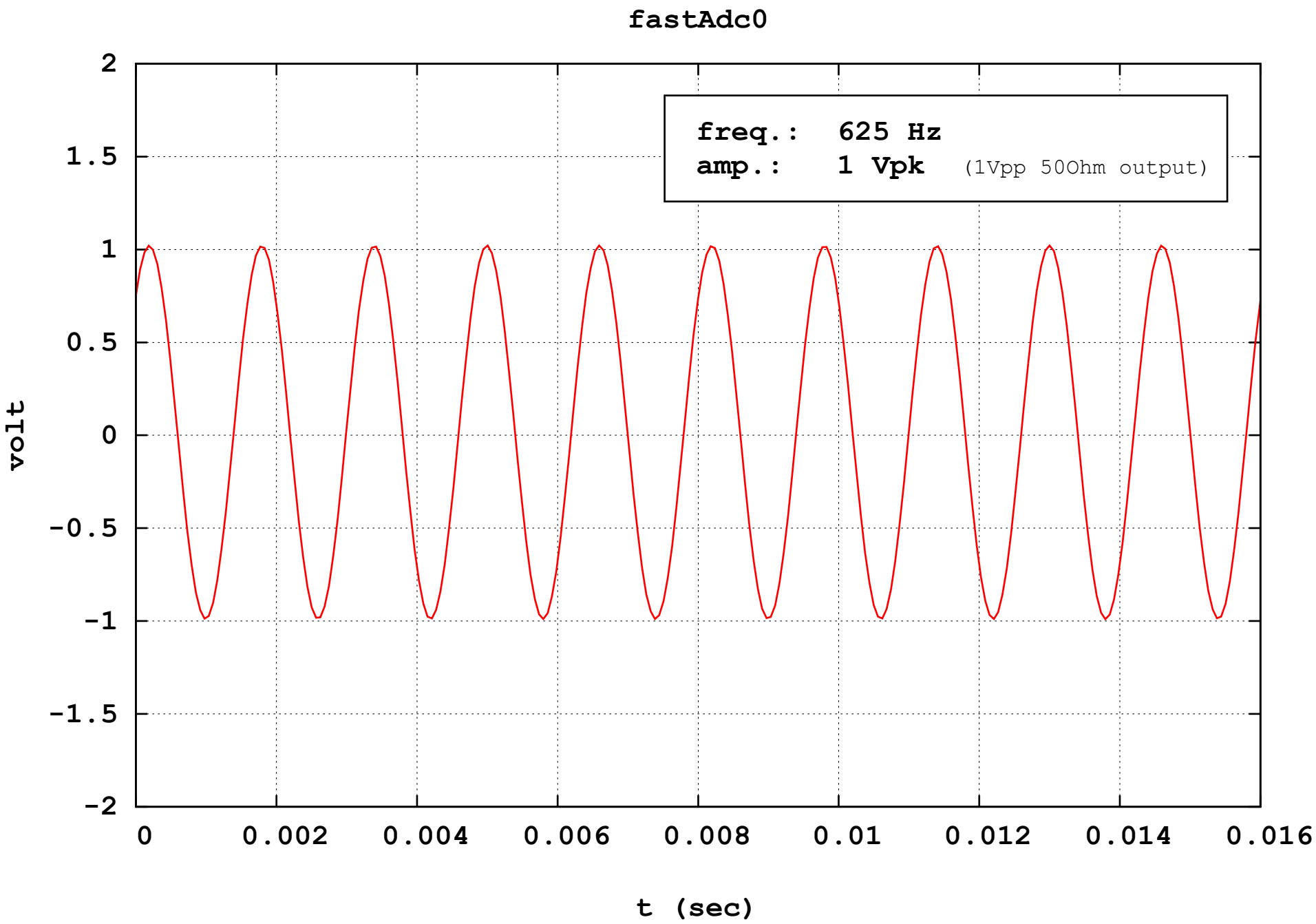
CHECK1

ADC input に発振器で作ったサイン波を入力。

周波数：625 Hz
振幅：　1 Vpk

Frame Data から保存された ADC 値を電圧値に換算してプロットしたもの。

この段階では問題なくデータが所得出来ている様に見える。


fastAdc0
freq.: 625 Hz
amp.: 1 Vpk (1Vpp 50Ohm output)
volt
t (sec)
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
0.002
0.004
0.006
0.008
0.01
0.012
0.014
0.016

CHECK2

サイン波信号

周波数：　625 Hz
振幅：　　1 Vpk

４秒間のデータを FFT 変換した結果。

16kHz sampling と書いてあるの時は、正確には 16.384 KHz sampling に設定されている。

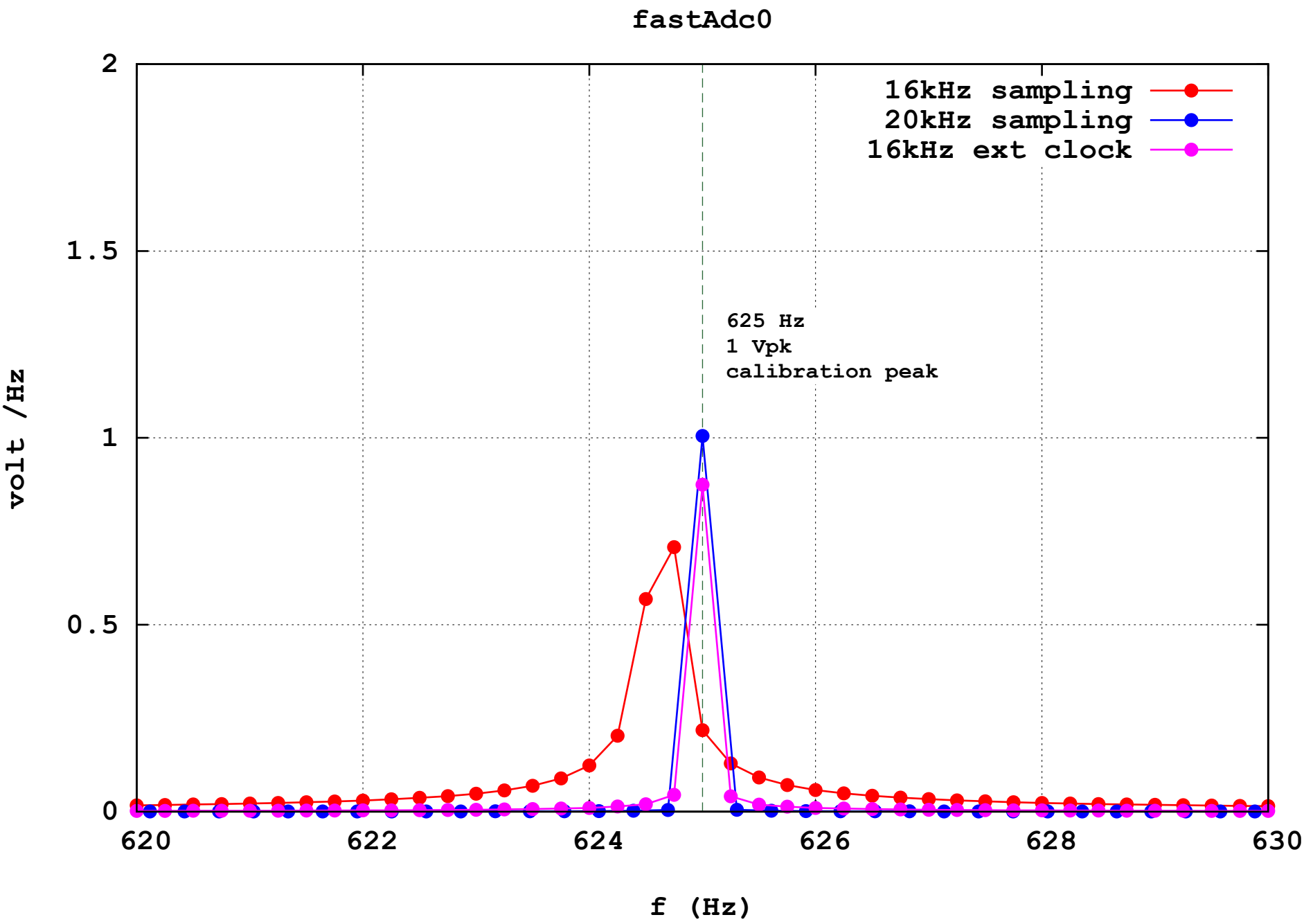
これを見ると、16.384 kHz sampling は Internal Clock から software select により作られている。
Internal Clock の元発振器は 40MHz , 50ppm stability というスペックと書いてあるが
どうにもピーク周波数のズレや、周波数の広がり（不安定性）は許容しがたい。

確認のため、20 kHz sampling に software select してみると
ピーク周波数ズレも周波数の広がりも見ることが出来ない。（問題なし）

40MHz の整数分周でないと Internal Clock の精度が悪くなるのではないかと思われる。

そこで、外部クロックを ADC に入力することにした。
クロック源は「Tektronics AFG3251」を使用した。
信号レベルは TTL (0:5V)。周波数は 16384 Hz x 8 CH = 131072 Hz に設定。

結果は、20 kHz internal clock 時と同様に、ピーク周波数シフトも周波数広がりも
見られないので、問題が解決したと言える。


fastAdc0
16kHz sampling
20kHz sampling
16kHz ext clock
625 Hz
1 Vpk
calibration peak
volt /Hz
f (Hz)
2
1.5
1
0.5
0
620
622
624
626
628
630

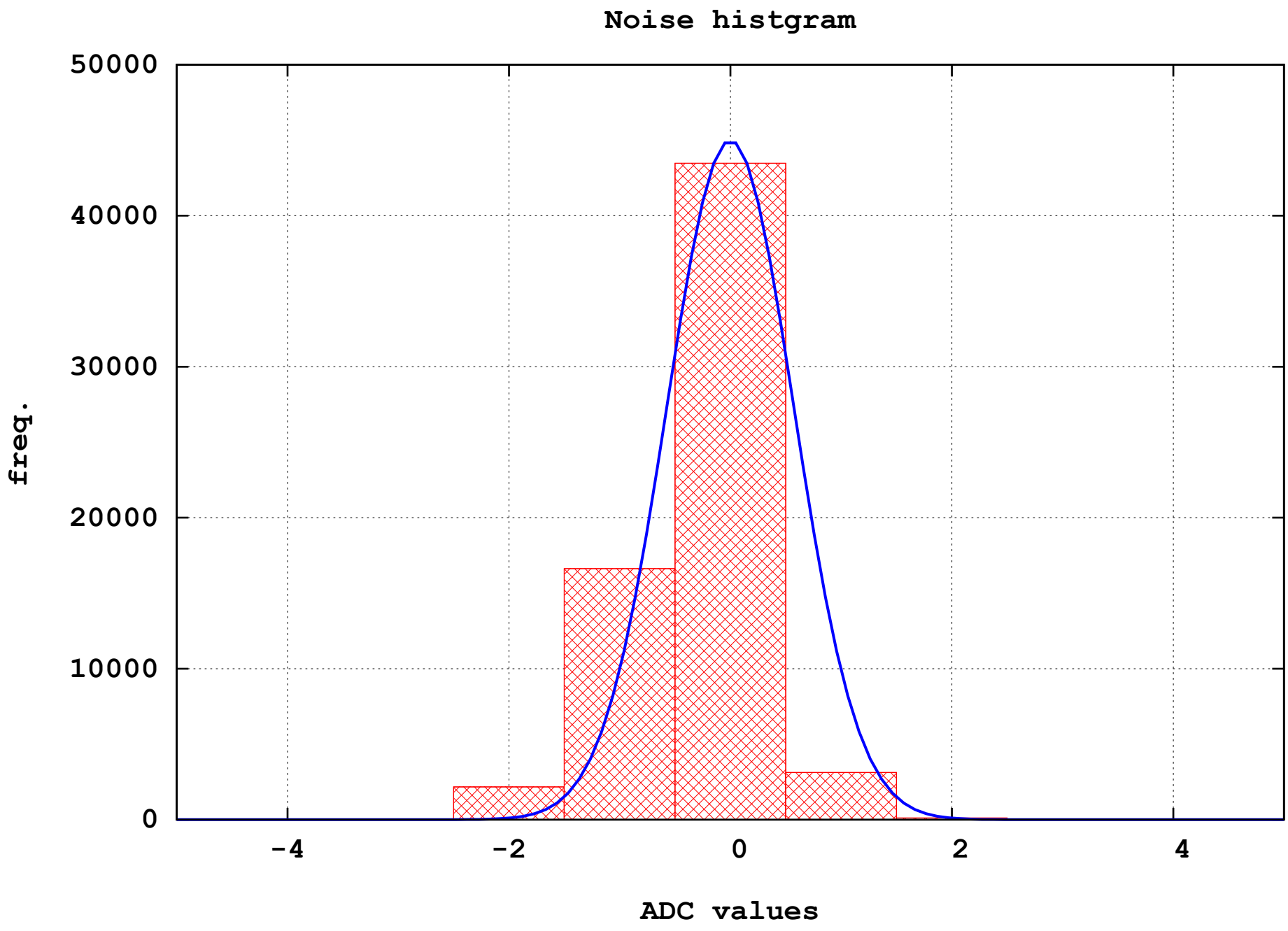
CHECK3

CH0 の入力に 50Ω端子をつないで、雑音レベルを調べた。

```
# -- HISTGRAM DATA --
#
# ADC      FREQ.
-3  -2        40
-2  -1      2165
-1   0     16612
 0   1     43480
 1   2      3122
 2   3       117
```

ざっと 0.9 LSB でカタログ値通り。

ただし絶対精度は、9 mV = 29.4 LSB (at +-10V range)。


Noise histgram
50000
40000
30000
20000
10000
0
freq.
-4
-2
0
2
4
ADC values

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <math.h>
#include <time.h>
#include <conio.h>
#include <dirent.h>
#include <windows.h>
#include <dir.h>

#include "FrameL.h"
#include ".¥cbw.h"

HANDLE hFMWrite;

void
cleanup(int BoardNum, WORD *ADData) {
        fprintf(stderr, "cleaning up¥n");
        cbStopBackground(BoardNum,AIFUNCTION);
        cbWinBufFree(ADData);
}


int
main (int argc, char *argv[])
{
        /*************************************************************/
        /**********   A / D   B o a r d   P a r a m e t e r s   **********/
        /*************************************************************/
        int                             BoardNum                = 0;
        int                             LowChan                 = 0;
        int                             HighChan                = 7;
        unsigned short  Gain                            = BIP10VOLTS;
//      int                             Options                 =  BACKGROUND  +
CONTINUOUS;
        int                             Options                 =  BACKGROUND  +
CONTINUOUS + EXTCLOCK;
        /*************************************************************/

        /*************************************************************/
        /*****   D A T A   A C Q U I S T I O N   P A R A M E T E R S   *****/
        /*************************************************************/
        long                    nData                   = 65536;
/*      double                  sampleRate              = 20000.; */
/*      double                  sampleRate              = 65536.; */
        double                  sampleRate              = 16384.;
/*      double                  sampleRate              = 32768.; */

        int             NF_PER_FILE     = 1;     /* Number of frames per file */

        /*  External Clock                                      */
        /*  It should be (sampling rate) * (number of channels)  */
        /*************************************************************/
```

```
        /* ---------------   A / D   r e l a t e d   ----------------------*/
        int EXTCCONV        ULStat;
        short                           Status = RUNNING;
        long                            CurCount, CurIndex=0, OldIndex;
        float                           revision = (float) CURRENTREVNUM;
        long                            first_data = 1;
        INT64                           samp_count=0;
        WORD                            *ADData;
        static WORD                     *dbuf;
        long                            r, wrote_this_pass;
        int                 TrigType;
        unsigned short      LowThreshold, HighThreshold;

        /* ---------   F R A M E   L I B R A R Y   r e l a t e d   --------*/
        int                                     true = 1 , Stop = 0, nb, debug;
        long                            Rate;
        long                            nframe = 0, nfile = 1 ;
        struct tm                       *ttm ;
        time_t                          tt , localTime, UTC ;
        struct FrFile           *oFile;
        struct FrameH           *frame;
        struct FrAdcData        *adc[64];
        int                 ch, nADC, nrun;
        long                j, k;
        const unsigned int  BuffSize = 2000000 ;  // Buffer size for FrameL
        char                name[16], fr_file_name[128], *buff ;
        char                            fr_dir_name[1024];
        /*********************************************************************/
    static                              LPSTR lpStr;
        /*********************************************************************/


        debug = 0;
        Rate  = (long) sampleRate;
        fprintf (stderr, "Rate = %d¥n", Rate);
        if (argc  != 2) {
//              printf ("¥nUsage : hdaq <#run>¥n");
//              exit (EXIT_FAILURE);
                printf ("Please input run number: ");
                scanf ("%d", &nrun);
        }
        else
        {
                nrun = atoi (argv[1]);
        }
    printf ("RUN# = %d¥n", nrun);

        /*-----------     Making a Data Directory  ------------*/
        sprintf (fr_dir_name, "D:¥¥data¥¥R%04d", nrun);
        if (!opendir (fr_dir_name))
        {
                /* mkdir (fr_dir_name, NULL) */;
                mkdir (fr_dir_name);
                printf ("Data directory was made. %s¥n", fr_dir_name);
        }
        if (debug) printf ("Step:0¥n");
```

```
        /*--------------     Memory Allocation   --------------*/
        buff = (char *) malloc (BuffSize);
        if (buff == NULL) {
                fprintf (stderr, "Error can't malloc for buff.¥n") ;
                exit( EXIT_FAILURE );
        }
        if (debug) printf ("Step:1¥n");

        /*---------   Initialization of Frame Library   -------*/
        if (FrLibIni ("D:¥¥data¥¥log_files¥¥hdaq-frame.log", NULL, 0) == NULL) {
/*              fprintf (stderr, "Error Frame Library Initialization.¥n"
                " %s ", FrErrorGetHistory()); */
                exit (EXIT_FAILURE);
        }
        if (debug) printf ("Step:2¥n");

        /************    Create header region on memory   **************/
        frame = FrameNew ("TAMA-HDAQ") ;
        if (frame == NULL) {
                fprintf (stderr, "Frame header create error¥n") ;
                exit( EXIT_FAILURE ) ;
        }
        frame->run       = nrun ;
        frame->ULeapS    = 35 ;       /* as of 2013 */
        frame->dt        = (double) 1.0 / sampleRate * nData ;
        printf ("Frame length : %g¥n", frame->dt);
        if (debug) printf ("Step:3¥n");

        /************    Create data region on memory    **************/
        nADC = HighChan - LowChan + 1 ;
        for (ch=LowChan; ch<HighChan+1; ch++)
        {
                sprintf (name, "fastAdc%d", ch);
        adc[ch] = FrAdcDataNew( frame, name, sampleRate, nData, 16 );
        /* adc[ch]->slope = 2.56 * 2 / pow( 2.0, 16.0 ); */
                adc[ch]->slope = 10.0 * 2 / pow( 2.0, 16.0 );
        adc[ch]->bias  = 0.0 ;
        if (adc[ch] == NULL)
                {
                FrameFree( frame );
                fprintf(stderr, "ADC data area create error¥n") ;
                exit( EXIT_FAILURE ) ;
        }
        }
        if (debug) printf ("Step:4¥n");

        /***************      F i l e   M a p p i n g    *************/
    if (!hFMWrite)      CloseHandle (hFMWrite);
    hFMWrite = CreateFileMapping(
       (HANDLE)-1,
       NULL,
       PAGE_READWRITE,
       0,
       1024,
       "TAMA_HDAQ_INFO");
```

```
    if (hFMWrite == NULL) exit (-1);

    lpStr = (LPSTR) MapViewOfFile (
                hFMWrite,
       FILE_MAP_ALL_ACCESS, 0, 0, 0);
    if (lpStr == NULL) exit (-2);
    if (debug) printf ("Step:5¥n");

        /*****************************************************************/
        /*******     R e a d y   F o r   A / D   S T A R T     ********/
        /*****************************************************************/
        ULStat = cbDeclareRevision(&revision);

        /*--------------     Memory Allocation   --------------*/
        dbuf = (WORD *) calloc (sizeof(WORD), nData * (HighChan - LowChan + 1));
        if (dbuf == NULL) {
                fprintf (stderr, "Error can't malloc for dbuf.¥n") ;
                exit( EXIT_FAILURE );
        }

        /*-----------  ring buffer creation  --------------*/
        ADData = (WORD*) cbWinBufAlloc(nData);
        if (!ADData)
        {
                fprintf(stderr, "cbWinBufAlloc failed¥n");
                exit(1);
        }
        cbErrHandling (PRINTALL, DONTSTOP);

//      cbSelectSignal(BoardNum, SIGNAL_IN,
//                      ADC_CONVERT, AUXIN1, NEGATIVEEDGE);

        /*-----------  start A/D conversion  -------------*/
        cbAInScan (BoardNum, LowChan, HighChan,
                nData, &Rate,
                Gain, ADData, Options);
        /*****************************************************************/

        time (&tt);
        ttm = localtime (&tt);
        printf ("¥n¥nNow start HDAQ system : %s¥n", asctime( ttm ));

        sleep (1);

/*********************************************************/
/*******           MAIN LOOP FOR DAQ             ********/
/*********************************************************/
        while (true && Status == RUNNING) {
                /* * * * * *    Frame File Open   * * * * * * */
                sprintf (fr_file_name, "D:¥¥data¥¥R%04d¥¥R%04dF%01Old.gwf",
                        nrun, nrun, nfile);

                oFile = FrFileONew (fr_file_name, 0);
                if (oFile == NULL)
                {
                        FrameFree (frame);
```

```
                        fprintf   (stderr,   "Frame   file   open   error:   %s¥n",
fr_file_name);
                        exit( EXIT_FAILURE ) ;
                }


                /************************/
                /*****    NFRAME     *****/
                /************************/
                for (j=0; j<NF_PER_FILE; j++)
                {
                        /* * * * *    fill Frame header   * * * * */
                        localTime = time( NULL ) ;
                        UTC = mktime( localtime( &localTime ) ) ;
                frame->GTimeS    = UTC - 315964800 + frame->ULeapS ;
                frame->GTimeN    = 0 ;
                frame->frame     = nframe ;
            nb = 0;

                        while (true)
                        {
                                r=0; wrote_this_pass=0;
                                fflush(stdout);
                                Sleep(50); // msec
                                ULStat = cbGetStatus (BoardNum,
                                        &Status,      &CurCount,      &CurIndex,
AIFUNCTION);

                                if (CurIndex < 0) { // no valid data yet
                                        printf("CurIndex < 0¥n");
                                        continue;
                                }
                                if (CurIndex >= 0 && first_data) { // we'll wait for
more data
                                        if (debug) printf("seeing first data¥n");
                                        first_data = 0;
                                        OldIndex = CurIndex;
                                        continue;
                                }
                                if (CurIndex == OldIndex)
                                {
                                        printf("CurIndex didn't advance since last
pass¥n");
                                }
                                if (CurIndex > OldIndex)
                                { // write out the region from old to cur-1
                                        if (debug) printf("case A, ");
                                        memcpy (        &(dbuf[OldIndex + nData *
nb]),

        &(ADData[OldIndex]),

        sizeof(WORD)*(CurIndex-OldIndex)
                                                        );
                                        wrote_this_pass += r;
                                        OldIndex = CurIndex;
```

```
                                }
                                if (CurIndex < OldIndex)
                                { // write out old to end and start to cur-1
                                        if (debug) printf("case B, ");
                                        r = memcpy (    &(dbuf[OldIndex + nData * nb]),
        &(ADData[OldIndex]),
        sizeof(WORD)*(nData-OldIndex));
/*****************************************************************************/
                                        if (nb == 7)
                                        {
                                                if    (debug)    printf    ("memcpy start.¥n");
                                                for (k=0; k<nData; k++)
                                                {
                                                        for (ch=0; ch<nADC; ch++)
                                                        {
                                                                if (debug && ch == 0)
                                                                        printf ("%ld %d ¥n", k, dbuf[k*nADC+ch]);
        adc[ch]->data->dataS[k] = dbuf[k*nADC+ch] - 32768;
                                                        }
                                        }
                        if (debug) printf ("memcpy was done.¥n");
                                        if (FrameWrite (frame, oFile) != FR_OK)
                                                {
                                                FrFileOEnd(oFile);
                                                FrameFree(frame);
                                                remove(fr_file_name);
                                                        exit( EXIT_FAILURE );
                                        }
                                        if   (debug)   printf   ("FrameWrite   was done: %d¥n", nframe);
                                                localTime = time (NULL);
                                        fprintf  (stdout,  "#run  =  %5d  :  #file = %5ld : #frame = %7ld : %s",
                                                nrun,    nfile,    nframe,
        asctime(gmtime(&localTime))
                                                );
                                        fflush (stdout);
                                                sprintf (lpStr, "%ld %ld %ld %s",
                                                nrun,    nfile,    nframe,
        asctime(gmtime(&localTime)));
                                        if (Stop)
                                                {
                                                fprintf  (stderr,  "MDAQ has been stoped at %s¥n",
```

```
                                                                    asctime
(gmtime(&localTime))
                                                                );
                                                                break ;
                                                }
                                                nframe++;
                                                        wrote_this_pass += r;

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

                                                        r    =    memcpy    (&(dbuf[0]),
&(ADData[0]), sizeof(WORD)*CurIndex);
                                                        wrote_this_pass += r;
                                                        OldIndex = CurIndex;
                                                        break;
                                                }
                                                else
                                                {
                                                        nb++;
                                                        if (debug) printf ("nb = %d¥n",
nb);
                                                        r = memcpy (    &(dbuf[nData   *
nb]),

        &(ADData[0]), sizeof(WORD)*CurIndex);
                                                        wrote_this_pass += r;
                                                        OldIndex = CurIndex;
                                                }
                                        }
                                }
                                if (debug) printf("wrote %d words, CurIndex = %d¥n",
wrote_this_pass, CurIndex);
                                OldIndex = CurIndex;
                                samp_count += wrote_this_pass;
                        }
        FrFileOEnd (oFile);
        nfile++;
                Status = RUNNING;
        }
/*********************************************************/
/*******            MAIN LOOP FOR DAQ              ********/
/*********************************************************/
        cleanup (BoardNum, ADData);
        return (EXIT_SUCCESS);
}
```