

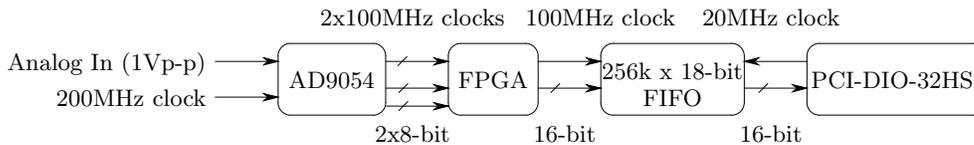
AD9054 Prototype Evaluation

Grant Hampson

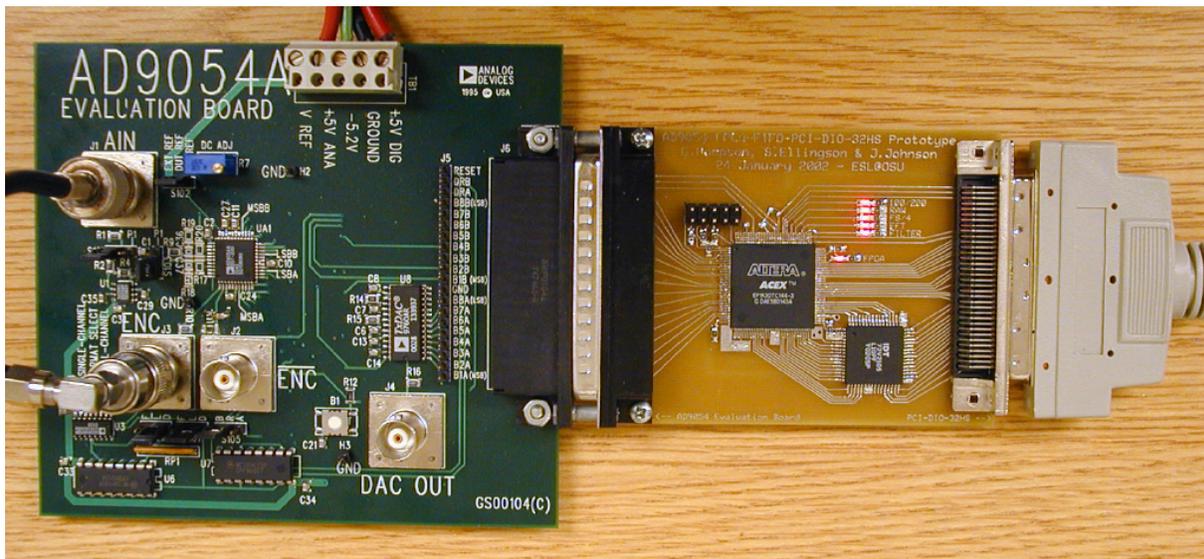
March 5, 2002

Introduction

This document describes the development of a 200MHz analogue-to-digital converter acquisition evaluation system. In particular an AD9054 Evaluation board [1] was purchased from Analog Devices for \$200. This board is interfaced via an Altera ACEX FPGA and 256k × 18 – bit FIFO to the PC using a PCI-DIO-32HS digital acquisition card [2]. Figure 1 illustrates the evaluation setup: the AD9054 evaluation board, interface card and SCSI-3 cable connected to the PCI-DIO-32HS board inside a PC.



(a) Evaluation System Block Diagram



(b) Photo of Evaluation System

Figure 1: AD9054 prototype board and PCI-DIO-32HS interface card. The PCI-DIO-32HS card is connected via the SCSI-3 cable on the right.

1 Interface Card Development

Once the AD9054 evaluation board is powered up (5V@500mA, -5.2V@150mA) and a 200MHz clock applied it will start producing digital samples. The output of the AD9054 is demultiplexed into two 100MSPS 8-bit buses since it is difficult to obtain a 200MSPS digital bandwidth across a single bus. These output buses are also skewed by half a clock cycle so as to distribute load on the system. The resulting outputs of the AD9054 evaluation board are two 8-bit buses, as well as two 100MHz clocks.

Since it is difficult to find a PC acquisition system cable of capturing this data directly, it is first captured in a high speed FIFO. The FIFO in this case is $256k \times 18$ - bit FIFO capable of capturing data at clock rates up to 100MHz [3]. In order to write data to the FIFO an Altera ACEX FPGA [4] is used to synchronize the two data streams and write them to the FIFO.

The FPGA also serves as an interface to the PCI-DIO-32HS digital interface card. This card has two main control signals - a read clock and a data flow control line. The clock frequency is controlled by the PCI-DIO-32HS card and can be up to 20MHz. The data flow control line indicates whether the PCI-DIO-32HS card can accept a new data sample, which in this case is connected to the FIFO read enable.

The FPGA code is contained in Appendix A. Simulations in the FPGA compilation package, Quartus II, indicate that the device we are using, the EP1K30TC144-3, will be capable of clock speeds up to 90MHz. The "-3" on the end of the part number indicates the speed grade of the device - the lower the number the faster the device. In this case we had this device left over from another project. Simulations with "-1" and "-2" devices indicate clock speeds in excess of 100MHz. Both a "-1" and "-3" devices were tested and both were capable of operating at a clock speed of 100MHz. The prices for each of the speed grades "-1", "-2" and "-3" are \$29, \$20, \$14, respectively.

The firmware for the FPGA can be found in Appendix A. It is self documenting code. The circuit board layout can be found in Appendix B. This circuit board was generated with ExpressPCB, a free software package.

2 LAB Windows CVI Interface

The LAB Windows CVI interface is relatively straight forward and will not be discussed in detail here. Refer to Appendix C for a source code listing. The code is self documenting.

A screen capture of the software is illustrated in Figure 2, where a 4MHz sinusoid is the input. The 8-bit samples are illustrated at the top and the the Fast Fourier Transform shown at the bottom. The number of samples (and FFT length) can be adjusted between 1 and 524288 (512k). It is also possible to window the incoming data with a Blackman window to stop FFT edge effects.

The raw data can also be written to disk in a text format (which can be later loaded into Matlab for more analysis.) An integration option also exists which integrates the FFT output for a given number of integrations. This output can also be written to disk.

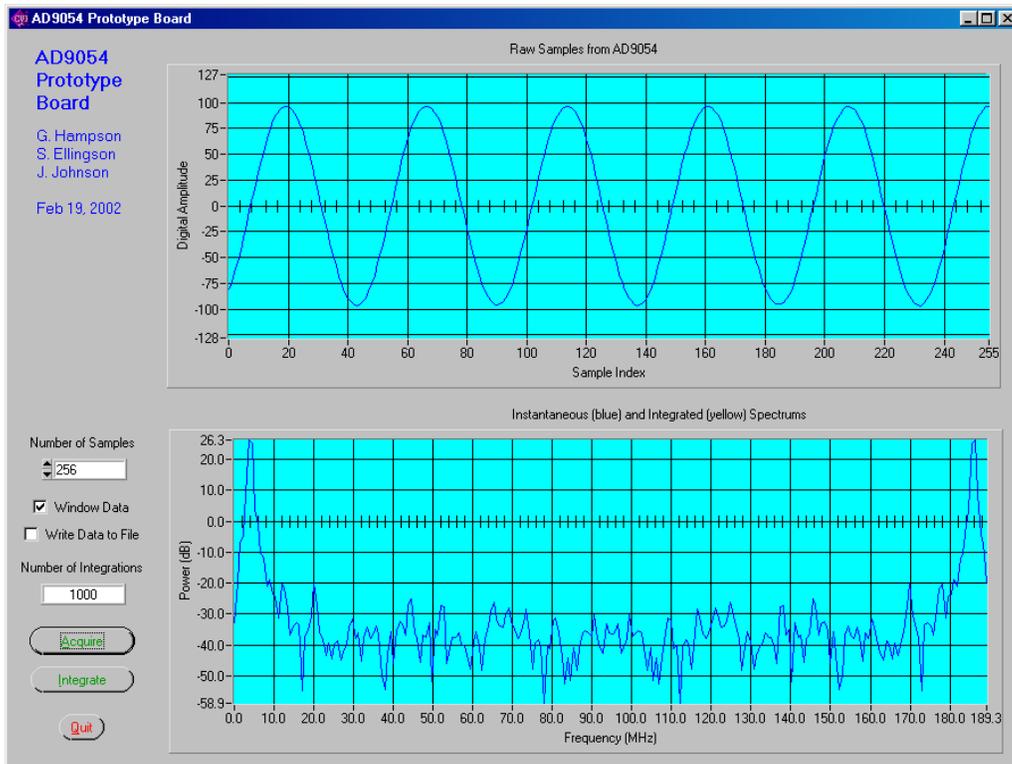


Figure 2: Example screen capture from the LabWindowsCVI user interface.

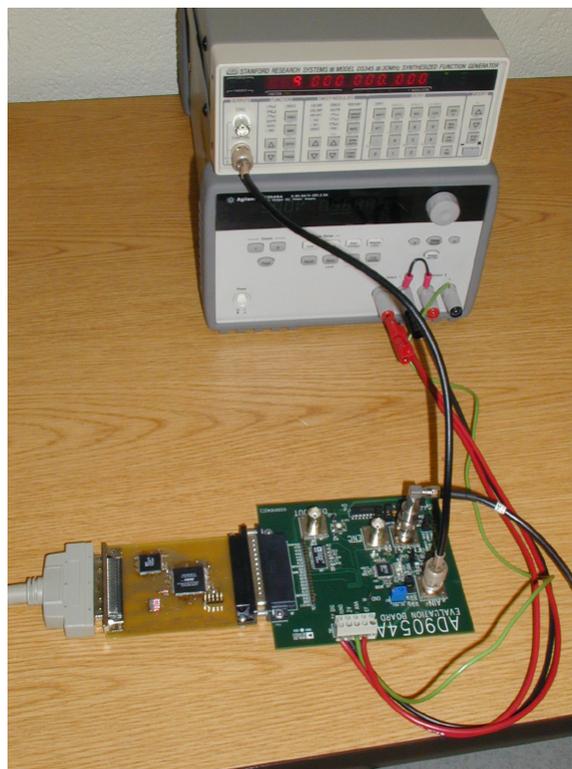


Figure 3: Measurement setup with Agilent E3848A power supply and SRS Synthesizer.

3 Measurements

The measurement setup is shown in Figure 3. An Agilent E3848A dual power supply provides $+5V$ and $-5.2V$. The 200MHz clock generator is not shown in the photo.

The analog input into the AD9054 is synthesized using the SRS Model 345 synthesized function generator (ARB). The synthesizer is capable of producing various waveforms (noise, sinusoids, etc..) For example, sinusoids can be generated up to 30MHz, but care must be taken that some harmonics are quite high and need low pass filtering.

Additional to the ARB, is a HP8350B Sweep Oscillator which is capable of generating sinusoids from DC to 20GHz. This oscillator will be used in the high frequency tests.

The following three sections illustrate the results from various stimulus.

3.1 Noise Input

The first input to the system was a noise input generated by the ARB. The noise bandwidth of the ARB is relatively small compared to the sampling rate of the system, so a 10.7MHz low pass filter was used on the input to limit the output BW. The result of 1000 integrations of length 32k FFTs is illustrated in Figure 4. The frequency response at 10.7MHz is -6dB and rolls off sharply to almost 48dB. Two tones exist in the result, one at 50MHz and another at 100MHz, which are likely to be the result of leaking clock harmonics.

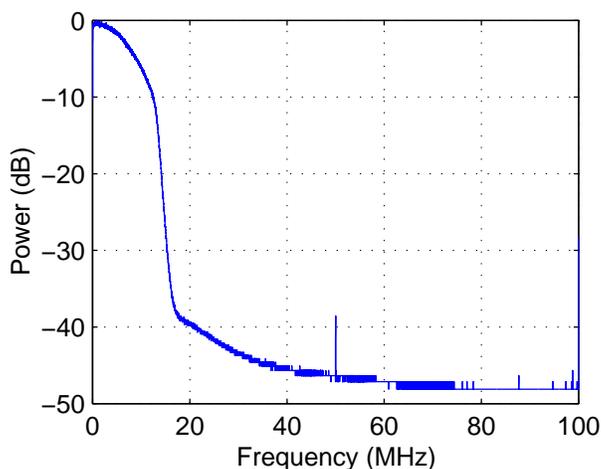
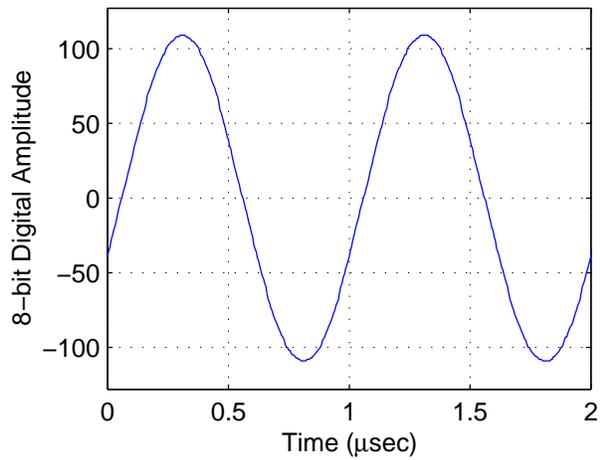


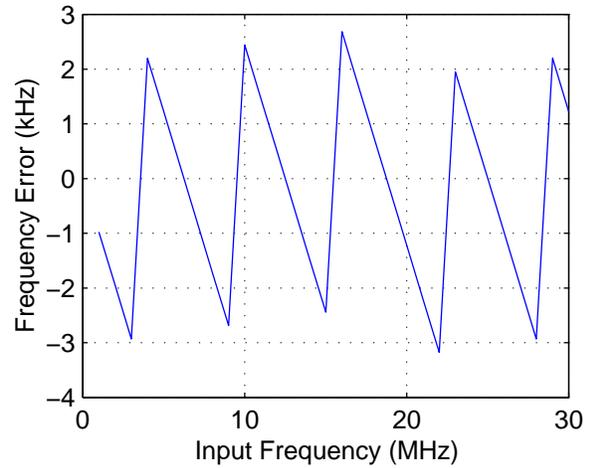
Figure 4: Noise spectrum from ARB with a 10.7MHz low pass filter. 1000 length 32k FFTs are integrated for this result. Clock leakage at 50 and 100MHz occurs.

3.2 Low Frequency (1-30MHz) Sinusoidal Inputs

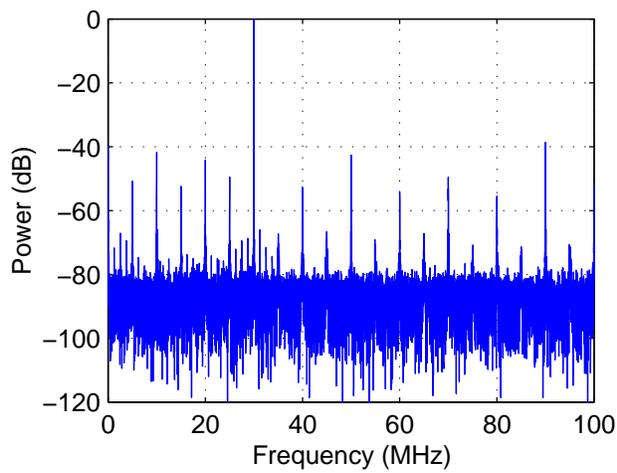
It is difficult to determine the SNR from a noise input and as such the system was tested with sinusoidal inputs. Figure 5(a) illustrates the time domain output of a input 1MHz sinusoid from the ARB. The time domain output is *clean* and contains no bad sampling artifacts. The input has an amplitude of 5dBm, or $1.12V_{p-p}$. Full scale of the AD9054 is rated at $1.024V_{p-p}$. There seems to be a small discrepancy between the two, but +5dBm will be assumed to be a safe input level (1dBm below full scale.)



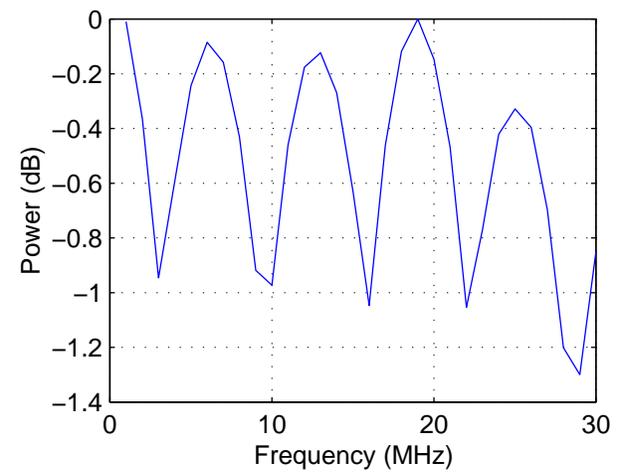
(a) 1MHz Sinusoid



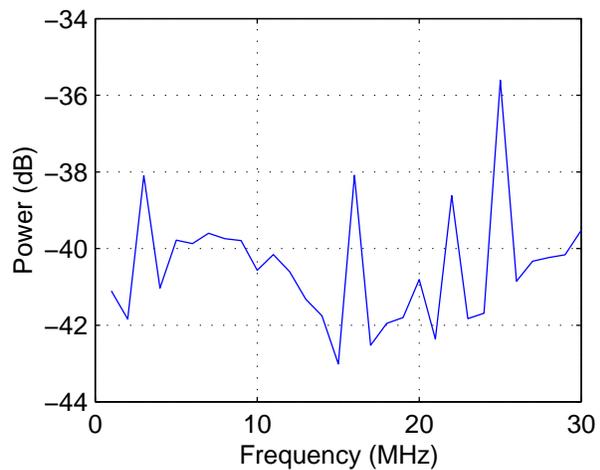
(b) Frequency Error



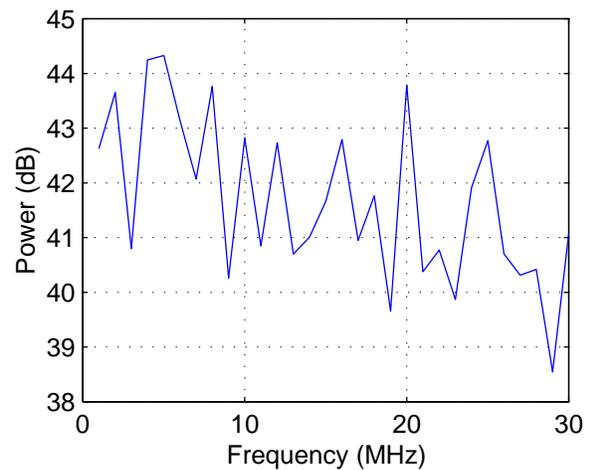
(c) 30MHz Spectrum



(d) Fundamental Power



(e) First Harmonic Power



(f) Signal-to-Noise Ratio

Figure 5: Results from the ARB for sinusoids between 1 and 30MHz, using a 50MHz low pass anti-alias filter.

The first sinusoidal test to be performed was to determine how accurate the frequency of the sampling frequency and ARB are. To do this the ARB was stepped in increments of 1MHz between 1 and 30MHz and a measurement taken. Figure 5(b) illustrates the difference between the ideal and measured frequencies. Using a 32k FFT, the frequency resolution is 6.1kHz. All of the results are within this range. It is possible to sample up to 512k samples (frequency resolution of 381Hz) but this was felt unnecessary.

The same data collected for the frequency tests was also used to determine the approximate Signal-to-Noise Ratio (SNR) of the AD9054. An example spectrum (30MHz input sinusoid) is shown in Figure 5c. Several pieces of information can be extracted from this spectrum. First the fundamental frequency power is noted. Secondly the harmonics and spurious spikes are removed such removal continues until the variance of the remaining noise varies less than 5%. Approximately 1-2% of the spectrum is zeroed using this method. The remaining *noise* is summed. The difference between the fundamental power and total noise power is taken to be the SNR.

Figure 5(d) illustrates the fundamental power for frequencies between 1 and 30MHz. In this case a 50MHz LP filter is used to anti-alias the input and the filter response can be seen here. The filter specification states that there is a 1dB ripple in the pass band.

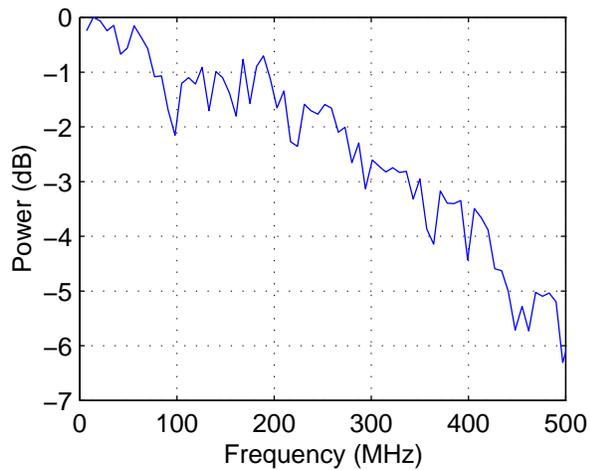
Figure 5(e) illustrates the power level of the first harmonic which ranges between 36 and 43dB below the fundamental. The SNR is illustrated in Figure 5(f) and ranges from 39 to 44dB. The AD9054 specifications state that a SNR of 40 to 43dB is possible, which is a relatively close match. There appears to be some degradation of performance with frequency. Additionally, when the fundamental input power drops due to the 1dB filter ripple - the noise floor of the AD9054 doesn't drop and the resulting SNR is degraded. So it is likely that the SNR is better than shown for certain frequencies.

3.3 High Frequency Sinusoidal Inputs

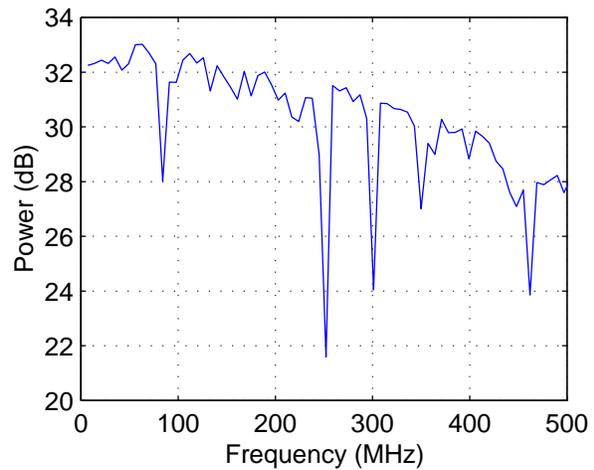
The next tested performed on the AD9054 were conducted using the HP8350B Sweep Oscillator which has a much greater frequency range. Unfortunately, it was not possible to install the appropriate anti-aliasing filters for the frequency range desired. As a result there were no anti-aliasing filters used in the these tests.

An input frequency ranging from 7 to 504MHz in 7MHz increments was input into the AD9054 at a power of +5dBm and a measurement taken for each frequency, similar to that of the low frequency tests. The resulting data was analyzed in the same way. The results of this analysis are shown in Figures 6(a) and (b). The analogue bandwidth of the AD9054 is rated at 350MHz - which is in close agreement with measurements. The SNR is however about 10dB below specification, which may be due to many factors - the main one being no anti-aliasing filters.

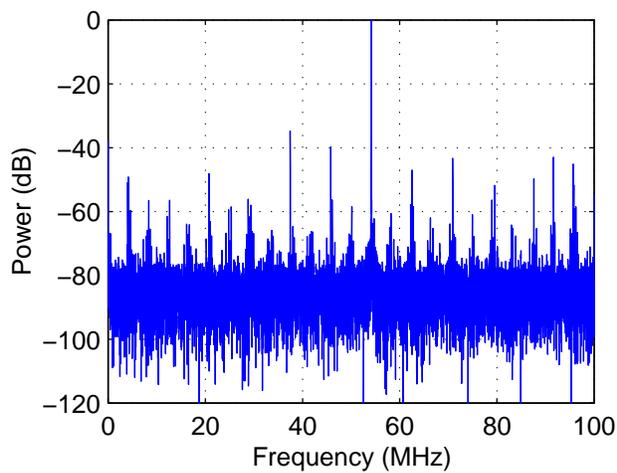
To verify that the anti-aliasing filters are required, we installed a 145MHz band pass filter - such that the AD9054 could be tested at this frequency. The resulting spectrum from the test is shown in Figure 6(c) - where the SNR was calculated to be 39dB which is close to acceptable. This SNR is significantly higher than the previously measured 32dB suggesting that anti-aliasing filters are required.



(a) Fundamental Power



(b) Signal-to-Noise Ratio



(c) Output with 145MHz Filter

Figure 6: (a)-(b) Results from the HP8350B Sweep Oscillator for sinusoids between 7 and 504MHz, with no anti-aliasing filters. (c) Spectrum for 145MHz input and a 145MHz BP filter. The SNR of this data is calculated to be 39dB - much greater than when no anti-aliasing filter is used in (b).

4 Conclusions

This document has evaluated the AD9054 8-bit 200MSPS analog to digital converter evaluation board. In order to evaluate the AD9054 ADC and interface card containing a FPGA and FIFO were constructed and programmed. The resulting system could be clocked at 200MSPS and many measurements were taken. The measured SNR's were in close agreement to the specifications, but the measurements revealed the need for good anti-aliasing filters in order to achieve the desired result.

References

- [1] *8-bit, 200MSPS A/D Converter*, Analog Devices Corporation, July 2001.
<http://www.analog.com/productSelection/pdf/AD9054A.d.pdf>.
- [2] *High-Speed 32-bit Digital Pattern I/O and Handshaking*, National Instruments, 2002.
<http://www.ni.com/pdf/products/us/2mhw332-333e.pdf>.
- [3] *IDT72V2105 3.3V High Density CMOS SuperSync FIFO, 262144×18-bit*, Integrated Device Technology, Inc., March 2001.
http://www.idt.com/docs/72V2105_DS_52815.pdf.
- [4] *ACEX 1K: Programmable Logic Device Family*, Altera Corporation, September 2001.
<http://www.altera.com/literature/ds/acex.pdf>.

Appendix A: FPGA Source Code

```
-- Interface between AD9054 Prototype Board and PCI-DIO-32HS
-- Grant Hampson 18 Feb 2002

SUBDESIGN ad9054proto
(
    dra,          -- clock from AD9054 development board for Data bus A
    drb,          -- clock from AD9054 development board for Data bus B

    busa[7..0],  -- Data bus A from AD9054 development board
    busb[7..0],  -- Data bus B from AD9054 development board

    fifo_full,   -- fifo status lines
    fifo_empty,

    pclk,        -- clock from PCI-DIO-32HS interface
    ack,         -- indicates start of transfer/transfer possible

    dioa[7..0],  -- data bus A from PCI-DIO32HS interface
    diob[7..0]   -- data bus B from PCI-DIO32HS interface

    :INPUT;

    adc_reset,   -- reset line for AD9054 development board

    fifo_data[15..0], -- data for the output 256k fifo
    fifo_wen,    -- FIFO controls
    fifo_wclk,
    fifo_mrs,
    fifo_rclk,
    fifo_ren,

    led[5..0],   -- Six status LEDs 100/200,RAW,FS/4,FFT,FILTER,FPGA

    req         -- tell PCI-DIO32HS that FIFO is ready
    :OUTPUT;
)

VARIABLE
    busa_in_reg[7..0] : DFF;  -- first input registers
    busb_in_reg[7..0] : DFF;
    busa_syncreg[7..0] : DFF; -- synchronisation register
    busa_reg[7..0] : DFF;    -- data is synchronised now
    busb_reg[7..0] : DFF;

BEGIN
    --
    -- Write to FIFO with 100MHz input
    --
    busa_in_reg[].clk = !dra;      -- Both buses directly into registers
    busa_in_reg[7..0].d = busa[0..7];
    busb_in_reg[].clk = !drb;
    busb_in_reg[7..0].d = busb[0..7];

    busa_syncreg[].clk = !dra;    -- dra 100MHz clock arrives first
```

```

busa_syncreg[].d = busa_in_reg[].q;

busa_reg[].clk = !drb;          -- same clock since data is synchronised now
busb_reg[].clk = !drb;
busa_reg[].d = busa_syncreg[].q;
busb_reg[].d = busb_in_reg[].q;

--
-- ADC controls
--
adc_reset = GND;

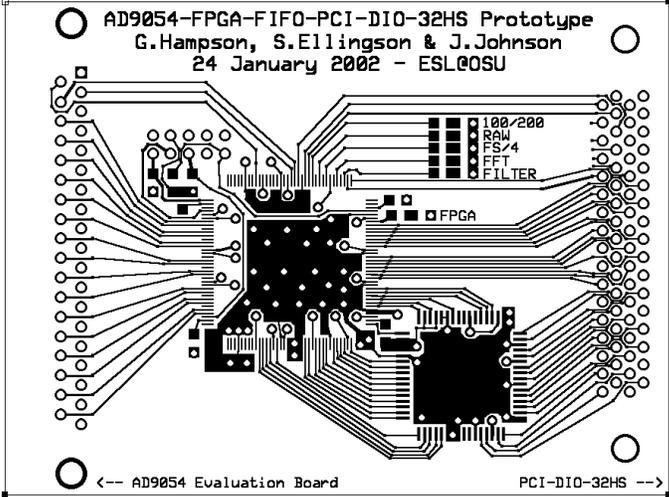
--
-- FIFO controls
--
fifo_data[15..8] = busa_reg[].q; -- First sample is high byte
fifo_data[7..0] = busb_reg[].q;  -- Second sample is low byte
fifo_wen = GND;                  -- always writing to PC
fifo_wclk = drb;

req = VCC;                        -- ask to transmit to PC
fifo_mrs = diob[0];               -- PC can reset the FIFO before reading
fifo_rclk = pclk;                 -- PC clock (20MHz)
fifo_ren = !ack;                  -- stops FIFO read when PC stops reading

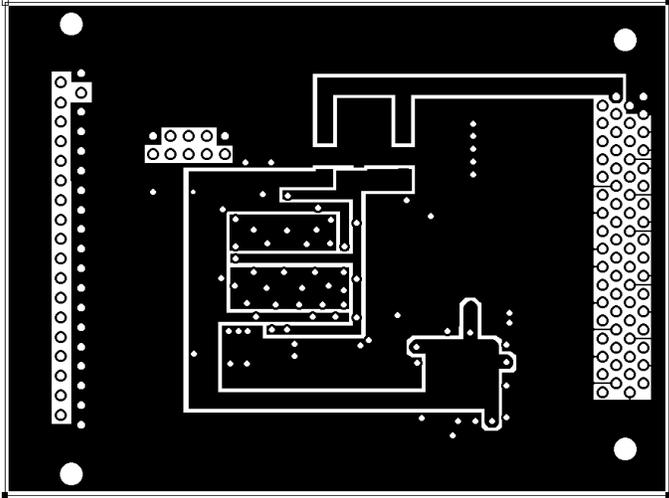
--
-- status leds
--
led[5..0] = VCC;                  -- nothing to monitor
END;

```

Appendix B: Circuit Board Layout



(a) Top side of layout



(b) Bottom side of layout

Figure 7: Interface board for AD9054 evaluation board and PCI-DIO-32HS interface card.

Appendix C: LAB Windows CVI Source Code

```
#include <utility.h>

/* PCI-DIO-32HS interface to AD9054proto board */
/* Grant Hampson 20 February 2002      */

#include <analysis.h>
#include <dataacq.h>
#include <ansi_c.h>
#include "ad9054proto.h"

#define FS 200
#define SAMPLES 524388
#define TINY (1e-20)
#define TRUE 1
#define FALSE 0
#define NO_WAIT 0
#define PCI_DIO_32HS 1 /* Board number assigned by NI-DAQ config utility */

static double xreal[SAMPLES];
static double ximag[SAMPLES];
static double power_intg[SAMPLES];
static short iBuf[SAMPLES];
static int hpMain; /* handle to panel "MAIN" */

void setup_daq_board()
{
    short iBoardType;

    Init_DA_Brds (PCI_DIO_32HS,&iBoardType); // Set up the PCI-DIO-32HS Board

    DIG_Grp_Config(PCI_DIO_32HS, 1, 2, 2, 0); /* group 1:16-bit,ports C&D, input */

    DIG_Grp_Mode(PCI_DIO_32HS, 1, 3, 0, 0, 0, 0); /* group 1:burst handshake
                                                    REQ ACK active high
                                                    100ns PCLK */

    DIG_Block_PG_Config( PCI_DIO_32HS, 1, 0, 1, 1, 25, 0); /* disable pat gen
                                                            external REQ */

    DIG_Line_Config(PCI_DIO_32HS, 1, 0, 1); /* Port B bit-0 is an output: LED1 on board */
}

void get_raw_data(int numsamples, int wtf)
{
    int i, raw_handles[8];
    char rawsample, filename[100];
    long nRemaining;
    FILE *filepntr;
    double *real_ptr, *imag_ptr;
    short *short_ptr;

    DIG_Out_Line(PCI_DIO_32HS, 1, 0, 0); // reset FIFO
}
```

```

Delay(0.001);    // Reset for 1ms
DIG_Out_Line(PCI_DIO_32HS, 1, 0, 1); // start capturing

DIG_Block_In (PCI_DIO_32HS, 1, iBuf, (numsamples/2)+2);
nRemaining = 1;
while(nRemaining>0) DIG_Block_Check (PCI_DIO_32HS, 1, &nRemaining);

real_ptr = &xreal[0]; // point to start of memory block
short_ptr = &iBuf[2]; // need to skip first 2 samples in FIFO
for (i=0; i<numsamples/2; i++)
{
    rawsample = ((*short_ptr) >> 8) - 127; // signed binary
    *real_ptr++ = (double)rawsample;
    rawsample = ((*short_ptr++) & 0x00FF) -127;
    *real_ptr++ = (double)rawsample;
}

if (wtf)
{
    sprintf(filename,"rawdata.dat");
    if((filepnter=fopen(filename,"w"))==NULL)
        printf("File could not be opened");
    for(i=0; i<numsamples; i++)
        fprintf(filepnter,"%f\n",xreal[i]); // write to file
    fclose(filepnter);
}

DeleteGraphPlot (hpMain, MAIN_GRAPH_RAW, -1, VAL_IMMEDIATE_DRAW);
PlotWaveform (hpMain, MAIN_GRAPH_RAW,
xreal, numsamples, VAL_DOUBLE, 1.0, 0.0, 0.0, 1,
VAL_THIN_LINE, VAL_NO_POINT, VAL_SOLID, 1, VAL_BLUE);
}

void do_raw_fft(int numsamples, int integrate)
{
    int i, fft_handles[8], wd;
    double real, imag, abspower, startfreq, stepfreq;

    GetCtrlVal(hpMain, MAIN_WINDOWDATA, &wd);
    if (wd)
        BkmanWin(xreal, numsamples); // Blackman window if needed

    ReFFT(xreal, ximag, numsamples); // xreal is input real and output real
                                     // ximag is output imaginary
    for(i=0; i<numsamples; i++)
    {
        real = xreal[i] / ((double)numsamples); // normalise real
        imag = ximag[i] / ((double)numsamples); // normalise imag
        abspower = real*real + imag*imag; // power == x^2+y^2
        xreal[i] = 10.0*log10(abspower + TINY); // 10*log10(abspower)

        if (integrate) power_intg[i] = power_intg[i] + abspower;
    }

    if (integrate)

```

```

    {
        for(i=0; i<numsamples; i++)
        {
            imag = power_intg[i]/((double)integrate);
            ximag[i] = 10.0*log10(imag + TINY); // 10*log10(abspower)
        }
    }

    startfreq = 0.0;
    stepfreq = FS/((double)numsamples);

    DeleteGraphPlot (hpMain, MAIN_GRAPH_FFT, -1, VAL_IMMEDIATE_DRAW);
    PlotWaveform (hpMain, MAIN_GRAPH_FFT,
        xreal, numsamples, VAL_DOUBLE, 1.0, 0.0, startfreq, stepfreq,
        VAL_THIN_LINE, VAL_NO_POINT, VAL_SOLID, 1, VAL_BLUE);

    if (integrate)
        PlotWaveform (hpMain, MAIN_GRAPH_FFT,
            ximag, numsamples, VAL_DOUBLE, 1.0, 0.0, startfreq, stepfreq,
            VAL_THIN_LINE, VAL_NO_POINT, VAL_SOLID, 1, VAL_YELLOW);
}

void main(void)
{
    int ns, numsamples, i, numint, wtf;
    char sampstring[100], filename[100];
    int iControlID, hHandle, bQuit = FALSE; // LabWindows/CVI-specific GUI stuff
    FILE *filepnt;

    hpMain = LoadPanel (0, "ad9054proto.uir", MAIN); // Load the GUI panel
    DisplayPanel (hpMain);
    setup_daq_board(); // configure the DAQ board (once!)

    while (GetUserEvent(NO_WAIT, &hHandle, &iControlID) || !bQuit)
    {
        if (hHandle==MAIN)
        {
            GetCtrlVal(hpMain, MAIN_WRITETOFILE, &wtf); // write to file check box

            GetCtrlVal(hpMain, MAIN_NUMBERSAMPLES, &ns); // number of samples
            numsamples = (int)pow(2,ns);

            GetCtrlVal(hpMain, MAIN_INTEGRATIONS, sampstring); // number of integrations
            numint = atoi(sampstring);

            switch (iControlID)
            {
                case MAIN_QUIT:
                    bQuit = TRUE;
                    break;

                case MAIN_GET_RAW:
                    get_raw_data(numsamples, wtf);
                    do_raw_fft(numsamples, 0);
                    break;
            }
        }
    }
}

```

```

case MAIN_INTEGRATE:
    for (i=0; i<numsamples; i++)
        power_intg[i] = 0.0; // zero integrator

    for (i=1; i<=numint; i++)
    {
        get_raw_data(numsamples, 0); // no write to file
        do_raw_fft(numsamples, i);
    }

    if (wtf)
    {
        sprintf(filename,"integdata.dat");
        if((filepnttr=fopen(filename,"w"))==NULL)
            printf("File could not be opened");
        for(i=0; i<numsamples; i++)
            fprintf(filepnttr,"%f\n",power_intg[i]/((double)numint)); // write to file
        fclose(filepnttr);
    }
    break;
}
}
Delay(0.1); // stop PC polling so hard
}
}

```