Evaluating the Parallax Propeller P2 Chip

Richard J. S. Morrison, Melbourne, AUSTRALIA December 16th, 2019

Motivation

The P2 is a powerful 8 core, 32 bit processor that can be over-clocked to speeds up to 360 MHz. The chip has a 512kB hubRAM space, with each of its 8 cores having a 512 long (32 bit) register space (cogRAM) and an additional 512 longs of lutRAM (look up table) space. A streamer allows high speed access to hubRAM, lutRAM and the chip's 64 I/O pins, which can also provide DAC outputs, and ADC and comparator inputs. In addition, any pin on the P2 can become a smart pin, capable of autonomously performing various analog and digital functions.

To support these impressive hardware capabilities, the P2 has an extensive instruction set. Most instructions execute in 2 clock cycles, which, if clocking at 250 MHz means 1 instruction every 8 nsec, or 125 MIPS/core – thereby reaching 1 GIPS if all cores are running code simultaneously ! With all of these capabilities and impressive horsepower the P2 clearly offers huge scope for sophisticated instrumentation projects.

With this in mind, this document describes my initial efforts to study the P2's many unique capabilities. I wanted to become familiar with all the different hardware options and also to investigate the P2's extensive and powerful instruction set. The experiments I'll describe used a Parallax P2-EVAL version B board, and assembly code running on the P2. I have found that a very convenient way to perform these evaluations is via "monitor" programs that respond to command strings sent from a host. My host is a PC running a National Instruments LabVIEW vi that interacts with the P2 via a USB port. Consistently reliable communications at 3Mbaud have proven possible between the PC and the P2-EVAL with the P2 clock set to 270 MHz throughout the course of this work.

I've developed a number of vi's and companion P2ASM codes to target testing of specific aspects of P2 functionality. My P2 code has borrowed heavily on numerous wonderful contributions described on the Parallax forums - but I particularly want to acknowledge those by Chip Gracey, ozpropdev and EvanH. Here, my focus will be on describing interactions with the hardware rather than on elaborating the software details, which I have documented via extensive comments in the code.

Clock Generator

Both P2-EVAL boards built by Parallax to date (rev. A and B) are fitted with a 20 MHz crystal oscillator. The oscillator output is fed to the P2's internal phase-locked loop (PLL) that first divides this down by a factor of 1 to 64 and then in a voltage-controlled oscillator (VCO), the resulting frequency is multiplied by 1 to 1024. The VCO's output frequency (which can be further divided down or used as-is) then becomes the P2's system clock.

This scheme allows considerable flexibility in being able to clock the P2 at many different frequencies. Although the engineering samples are specified for a maximum system clock of 180 MHz (check), over-clocking by a factor of 2 is possible. In all the work reported here I used a divide factor of 2 and a multiplier of 27 to yield a system clock at 270 MHz. Being curious as to how close the actual operating frequency would be to this, I used a software-

defined radio (SDR) dongle (https://www.rtl-sdr.com/buy-rtl-sdr-dvb-t-dongles/) and SDR# software to "listen" to an operating rev. A P2 chip and the results can be seen in Fig. 1. Here one sees that SDR# reports the clock generator to be oscillating at 269.9964 MHz, or 13 ppm from f=270 MHz. The dongle I am using has a TCXO and claims a maximum frequency error of 2 ppm which equates to 540 Hz at 270 MHz. I conducted another experiment using a DRVNOT instruction in a tight loop to generate a square wave output at f/4 MHz and with SDR# was able to observe the fundamental and numerous harmonics extending through the VHF/UHF regions.

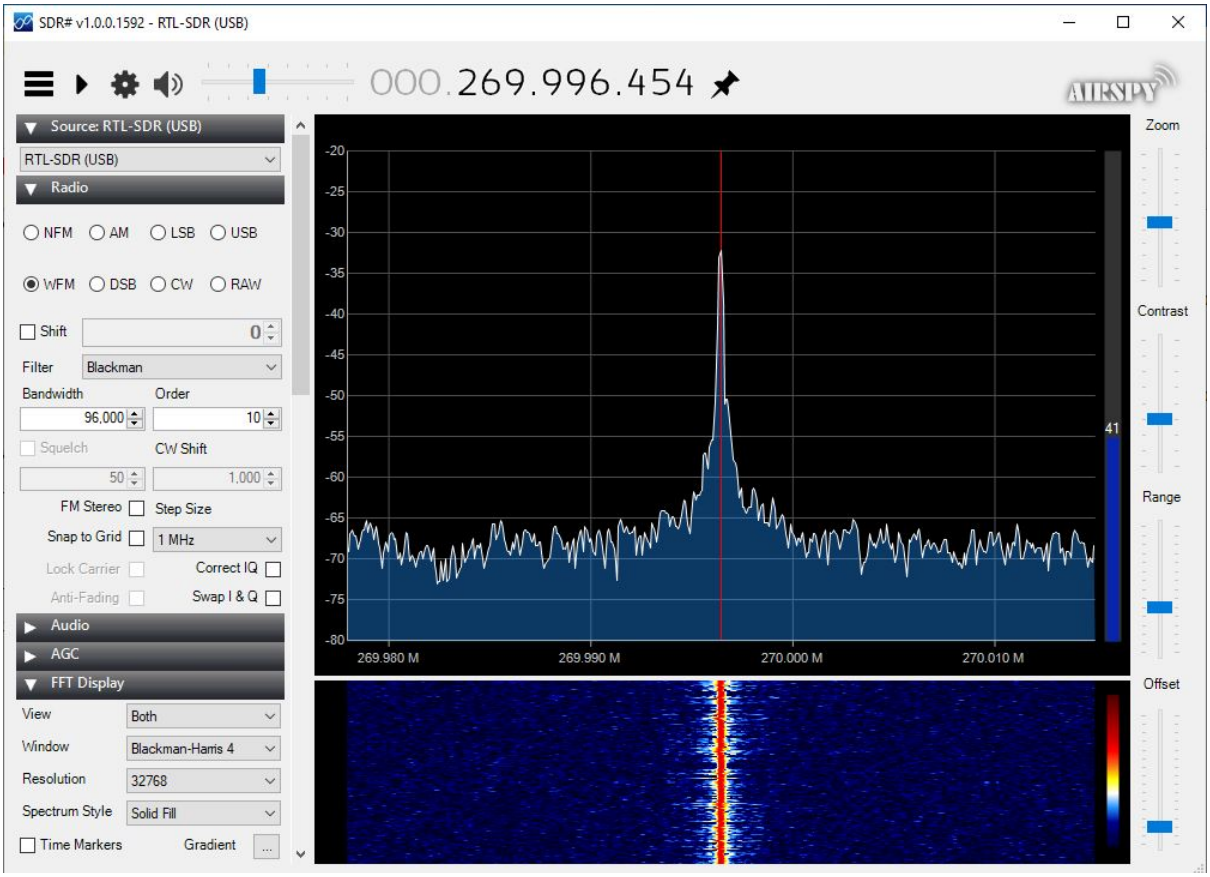


Fig. 1 The SDR# software detecting the P2 system clock, nominally at 270 MHz

Analog Functions – ADC's and DAC's (+ HyperRAM)

Analog-to-digital converters (ADC's) are essential for digitizing analog signals from sensors, while digital-to-analog converters (DAC's) are needed to generate analog output voltages to control experiments.

Using the P2's smart pin functionality, any pin can be configured to serve as an ADC or DAC without requiring any external circuitry. Let's begin by looking at analog input performance - connecting the output of a function generator to an ADC configured on pin 16. Fig. 2 shows the result of applying sine and triangle waveforms at 1 Hz with the ADC sampling 12000 points at a sampling rate of ~10 kHz. These waveforms are reproduced cleanly and performance looks excellent.

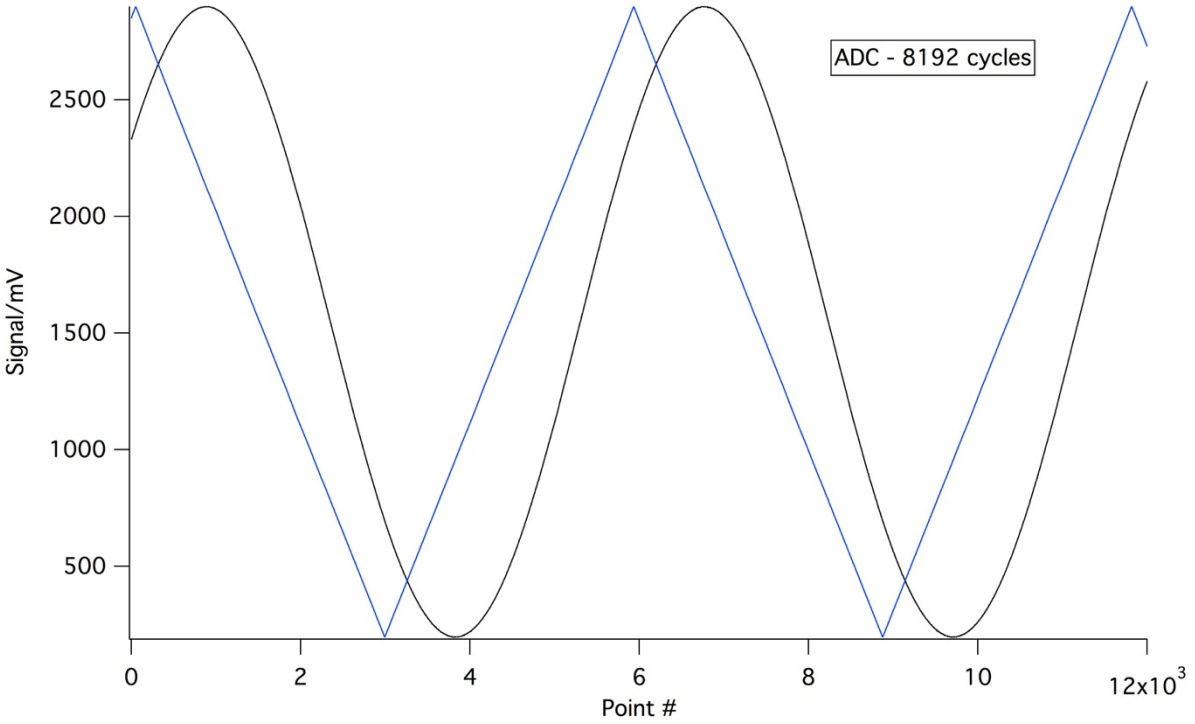


Fig. 2 Acquisition of sine and triangle waveforms by a P2 smart pin (#16) configured as an ADC. The sampling rate here was ~10 kHz (more on this shortly).

The vi being used here is shown in Fig. 3. To conduct an experiment the key things a user needs to specify are the number of points to acquire, an ADC pin #, an optional DAC pin # and a parameter ADC_Cycles (see later). When sampling from a single pin (as in Fig. 2) the DACPin? button is turned off. After pressing the Set # Pts and Set pins buttons, the vi's Run button is clicked (the forward arrow in LabVIEW; not shown in this screen capture).

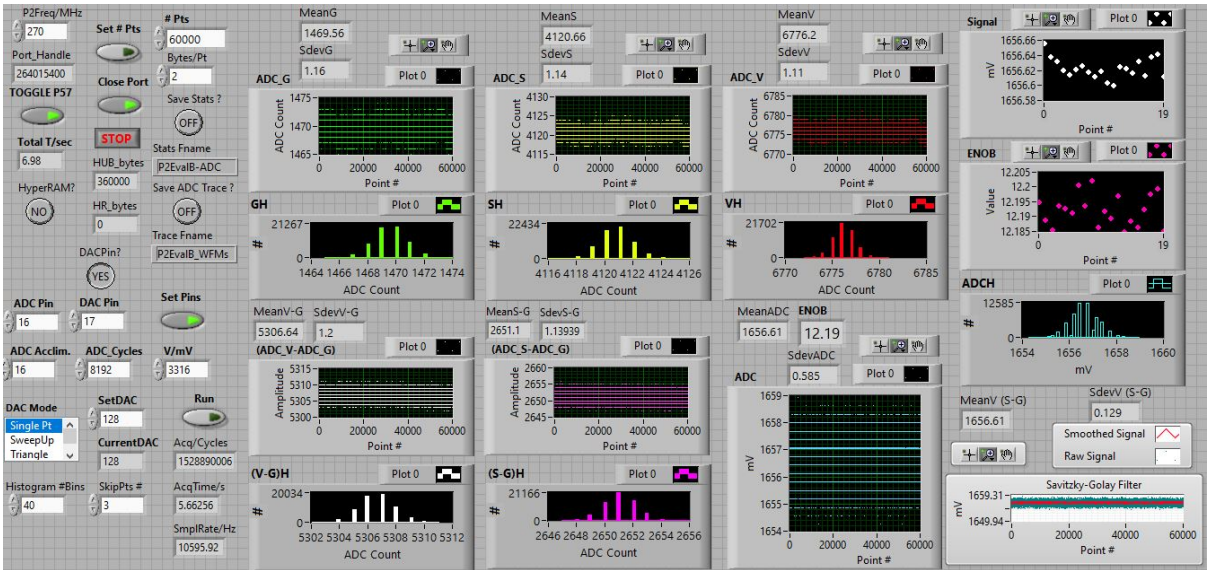


Fig. 3 The P2_Analog Test LabVIEW vi

Let's now look at the details of analog sampling on the P2. For optimum DC performance an ADC read operation actually involves three measurements – first sampling ground, then the input signal and finally the power rail, Vcc. The vi in Fig. 3 repeats these measurements multiple times (depending on the number of points requested by the user) and displays the results in 3 graph windows (ADCG, ADCS and ADCV) at the top of screen. Below these are histograms of the readings (GH, SH and VH).

The ADC resolution and sample rate are controlled by the ADC_cycles setting – the number of clock cycles over which to take each measurement. A larger value of ADC_cycles results in larger ADC counts (as the measurement takes longer) but this reduces the effective sampling rate. In practice the measured span between Vcc and ground readings is typically ~ 2/3 of ADC_cycles. The effective number of bits of resolution (ENOB) achieved in a measurement is then determined by this span and the noise determined in the measured signal readings.

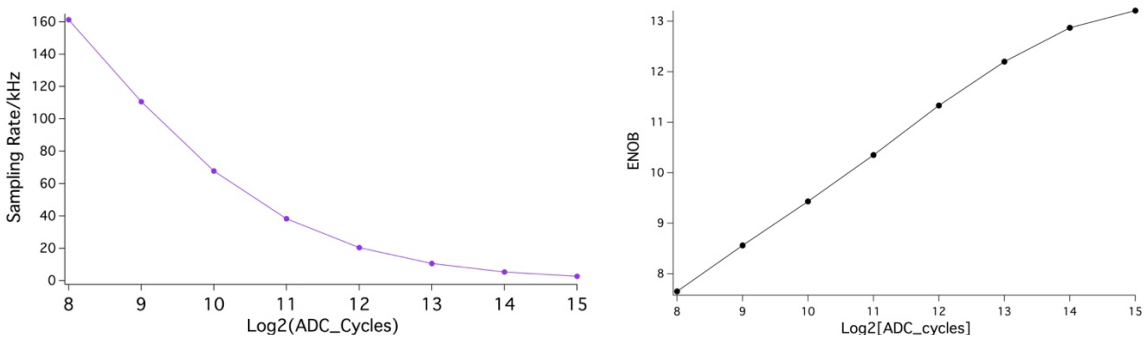
To determine any actual measured voltage (in mV) we take our 3 readings – call them ADC_g, ADC_s and ADC_v and calculate $(ADC_s-ADC_g)/(ADC_v-ADC_g)*3300$ mV. Additional graph windows at the bottom of the screen show (ADC_s-ADC_g) and its corresponding histogram as well as the final measured voltage in mV (ADC) and its histogram ADCH.

An effective # of bits of resolution (ENOB) is also computed and displayed – this is calculated by working out the # of bits represented by the span : $ln_2[(ADC_v-ADC_g)]$, then subtracting the # of bits in the noise, $ln_2[\sigma(ADC_s-ADC_g)]$, where σ is one standard deviation.

When a smart pin is configured in ADC mode it can also be instructed to sample from an adjacent pin. If this adjacent pin is configured as a DAC we can then read its output voltage. To do this we set the ADC pin to 16 (as before), but now turn DACPin? on and set the DAC pin = ADC pin + 1 - here 17. When operating in this manner three modes of operation (DAC_mode) have been catered for – single point, sweep and triangle mode. Constant_DAC mode does just that – the nominated DAC pin is set to the requested 8 bit value (the SetDAC field) and the run proceeds as before.

Fig. 3 above shows the results of 180k (60k pts x 3) ADC conversions with ADC_cycles = 8192 and SetDAC = 128 – producing a voltage on pin 17 midway between ground and Vcc.
In practice the 3 successive measurements (G, S and V) are highly correlated over the course of the run, meaning that our measurement protocol tends to flatten out fluctuations that would otherwise appear if S was just measured in isolation. This vi was run 20 times resulting in the Signal and ENOB plots in the upper right corner of screen to verify measurement reproducibility.

Figs. 4a and 4b below show some results obtained using the aforementioned ADC/constant DAC combination (again with the DAC set to mid-range of 128) and where the ADC_cycles parameter has been varied from 256 to 32768 in the successive powers of 2. The plots here show the ENOB and the achieved sampling rates (in Hz, determined in code using the GETCNT instruction) versus log2(ADC_cycles).



Figs. 4a (left) Sampling rate and **4b** (right) effective number of bits (ENOB) as a function of the # of ADC_cycles when a smart pin ADC is sampling a smart pin DAC on an adjacent pin.

When using ADC_cycles = 8192 (as in acquisition of the waveform data shown earlier in Fig. 1) the sampling rate obtained is 10.6 kHz and the ENOB is 12.2. The sampling rates quoted here are those obtained when doing 3-point measurements : i.e. G, S and V in sequence as discussed before. One can go faster, at lower resolution, or slower, at higher resolution. ADC_cycles = 8192 appears to be a sweet spot as there is some roll-off in ENOB for the longer acquisitions (16384 and 32768). The data obtained here provides confidence that the P2 pins can acquire 12 bit quality data @ 10 kHz.

LabVIEW has many in-built functions for signal processing and as a demonstration, a Savitsky-Golay smoothing routine was also implemented in this vi. The resulting plot can be seen in the bottom right hand corner of Fig. 3. After this operation the standard deviation of the smoothed signal has improved by nearly a factor of 5 compared to the original dataset.

The vi in Fig. 3 allows for two additional DAC modes – SweepUp and Triangle. These acquire ADC data while generating linear ramps - (0-255) and (0-255-0), respectively, on the nominated DAC pin. During a run the measured voltage at each DAC setting (CurrentDAC) and the ENOB are displayed in the two graphs at the top right of screen. These modes allow us to check on DAC linearity.

Fig. 5 below shows data recorded in the SweepUp mode. A regression line that is superimposed on the data points yields a slope of 13 mV per DAC unit; the fit is virtually indistinguishable from the data, validating the DAC's good performance.

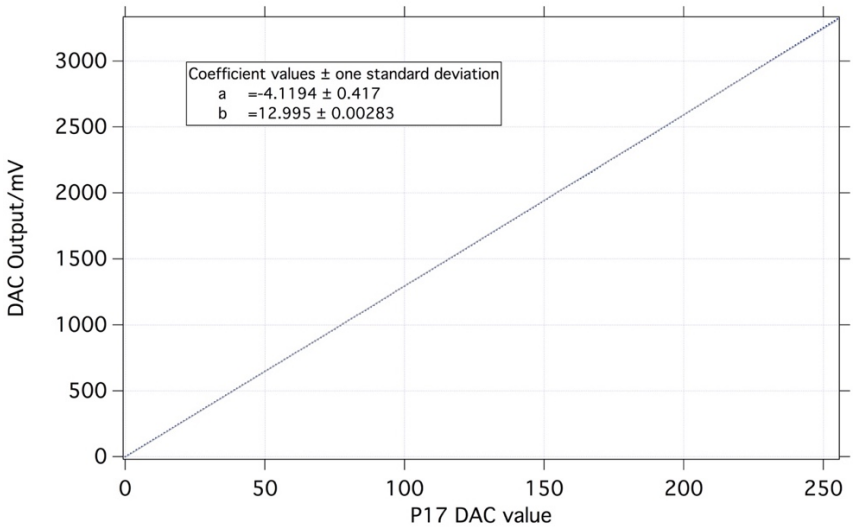


Fig. 5 ADC measurement of a smart pin-generated DAC ramp (ADC_cycles = 8192)

A further experiment has been conducted to see if there are any significant differences in performance when selecting different ADC/DAC pin pairs on the P2. A new vi was developed incorporating the features of P2_Analog-Test.vi, but now allowing the user to scan the ADC/DAC pin numbers – 0/1, 2/3, 4/5 … and for each pin number setting to scan the DAC value specifying a start, an increment and the number of DAC values. In the vi shown in Fig. 6, the ADC/DAC pin pairs have been scanned from 0/1 up to 28/29 (n.b. attached boards were using the higher numbered pins) and for each selection, 16 DAC values have been generated – 0, 16, 32 … 240. This vi was run twice, first using ADC_cycles = 4096 and then repeating but with ADC_cycles = 8192. As can be seen, the ENOB traces are fairly flat, indicating that the P2 pins, when used in analog mode are showing consistent behaviour. The "banding" seen in the ADC trace due to quantization effects is most prominent when sampling near the supply rails (GND and Vcc) but is much less noticeable at mid-range.

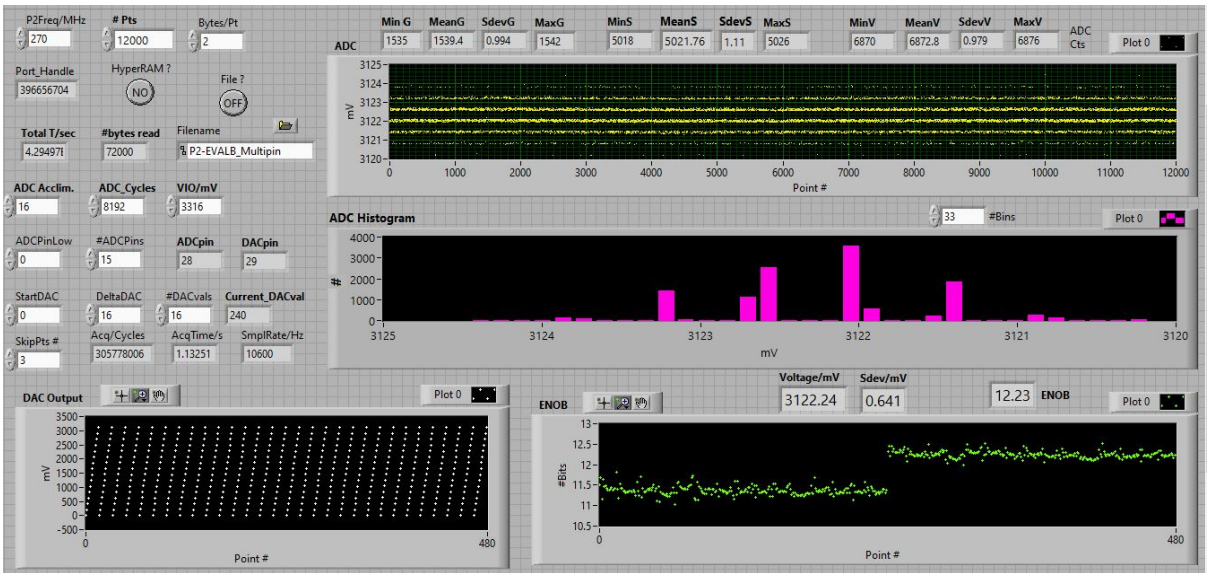


Fig. 6 Results obtained by scanning over a range of DAC values as the ADC/DAC pin pairs are incremented. Two distinct traces can be seen in the ENOB plot; the first half of the dataset is for ADC_cycles = 4096 and the second half for ADC_cycles = 8192.

SINC2 and SINC3 modes

The P2 employs a sigma-delta technique to generate a bitstream at the system clock frequency when performing analog-to-digital conversions. In this scheme, conversions occur at a slower rate than the system clock that is determined by ADC_cycles, the number of clocks over which to accumulate the number of "1's" in the bitstream.

There is an excellent tutorial at https://www.analog.com/en/design-center/interactive-design-tools/sigma-delta-adc-tutorial.html# that explains how a bitstream evolves over successive clock cycles, given an input voltage and a reference voltage.

The rev B P2 chip provides us with two new ADC modes – SINC2 and SINC3 filtering. These modes add extra accumulations during the acquisition process to improve resolution. The former adds an extra bit of resolution (and is intended for improved DC measurements) while the latter is claimed to double the ENOB and targets dynamic signals.

To experiment with these new modes we now discuss the vi shown in Fig. 7. Here one can make a MODE selection of simple bit summing or invoke either SINC2 or SINC3 filtering. In these measurements we are not concerned with ground or Vcc measurements, just with measuring the pin input voltage. Compared to the earlier work described this gives us an immediate factor of 3 speed-up in our sampling rates.

We connect a function generator to pin 16 and select one of the above three ADC modes to acquire a waveform. My vi incorporates a Levenberg-Marquardt algorithm (a built-in LabVIEW function) to perform non-linear least squares fitting of the recorded waveform to A sin(bt+c) + d – where A is the amplitude, b is the frequency, c the phase and d the offset. An initial guess for these parameters is pre-calculated using the ADC_cycles parameter specified by the user and the nominal input frequency. The vi then displays the acquired waveform (in yellow), the fitted waveform (in red) and a residuals plot showing the difference : (acquired -fitted, in green).

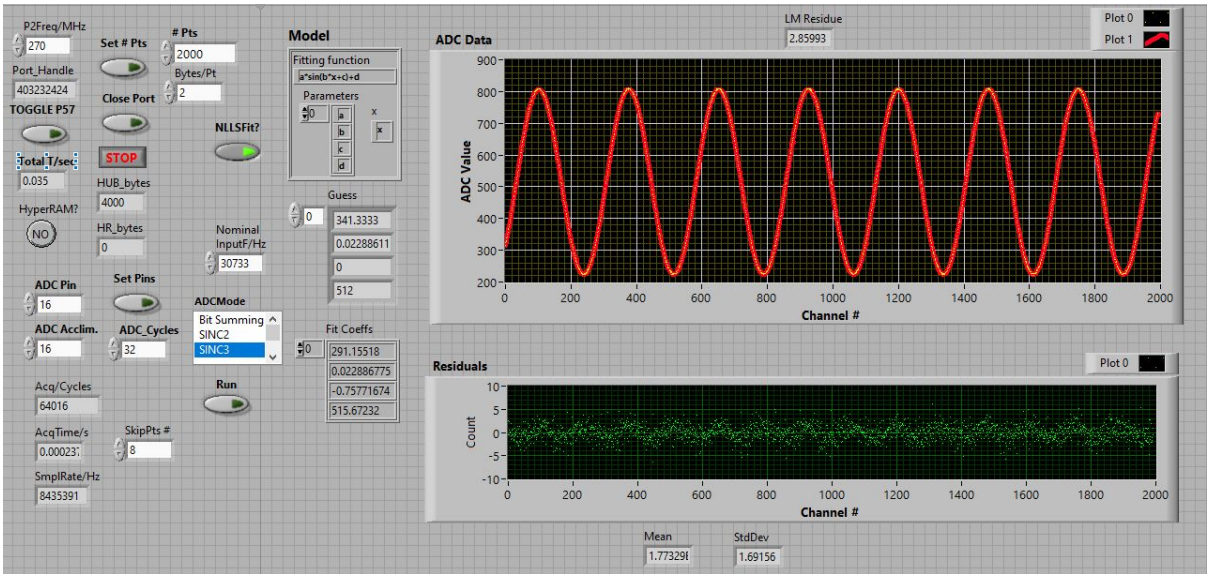


Fig. 7 A LabVIEW vi developed to experiment with the P2's ADC SINCn modes

In SINC3 mode the ADC_cycles parameter is limited to a maximum value of 512, which yields an ENOB of 18 bits, but with my code (which uses the WRWORD instruction to store acquired points into a HUB buffer) I restrict to ADC_cycles to 256 or less. The plots in Fig. 7 show the results of measuring a 30733 Hz sinewave being generated by the function generator shown in Fig. 8, with ADC_cycles=32. The acquisition here is 2000 points long. Note the excellent fit and residuals – all the points in the latter trace fall within a +-2 count band of zero. A small oscillation is seen in the residuals trace showing that the performance of the ADC is slightly worse near GND and Vcc than at mid-range, but this effect is minimal. Also note the sampling rate reported here – 8.44 MHz – impressive !

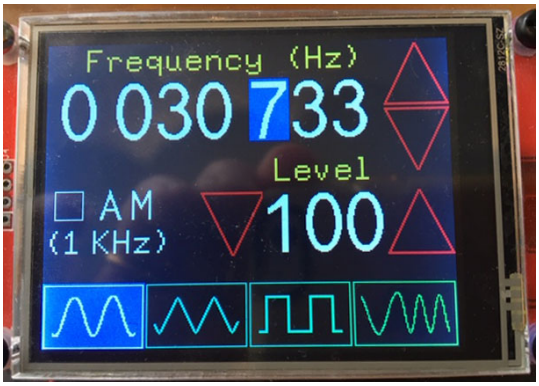
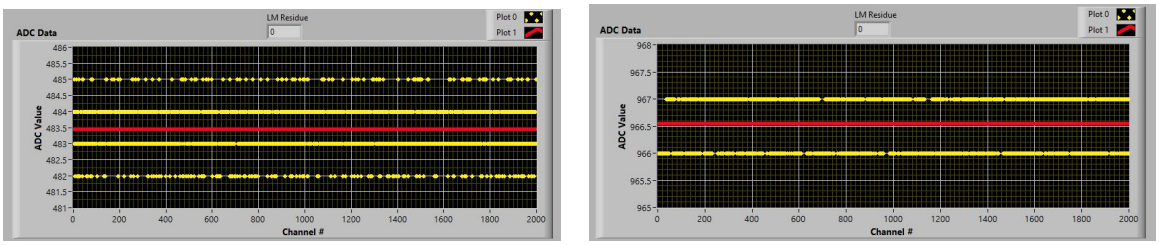


Fig. 8 Function generator generating a 30733 Hz sine wave

In Figs 9a and 9b we show a DC measurement using bit summing (left) and SINC2 modes (right), each recorded using ADC_cycles = 1024. The extra bit of resolution is clearly apparent in the SINC2 dataset (n.b. the sampling rate here is 263.7 kHz).



Figs. 9a and 9b Bit summing (left) and SINC2 (right(measurements of the same DC level.

HyperRAM

As a final note, the P2ASM code and the LabVIEW vi's described here allow the option of saving the acquired data on-the-fly into HyperRAM as opposed to using hubRAM, using the HyperRAM? button. In this case a Parallax HyperRAM/HyperFlash PCB is connected at base pin 32 on the P2-EVALB. For word length (2 byte) acquisitions and at 3 measurements per point the P2's hubRAM size limits a single experiment to ~120k samples. Using HyperRAM (with 8 MB capacity) a single experiment can acquire > 1.3 million points, which at 10 kHz would mean more than 2 minutes of 12 bit data acquisition.

Streamer, Smart Pins and Cordic Operations

Now switching gears, let's look at some other neat things we can do with a P2. We start by looking at the vi shown in Fig. 10.

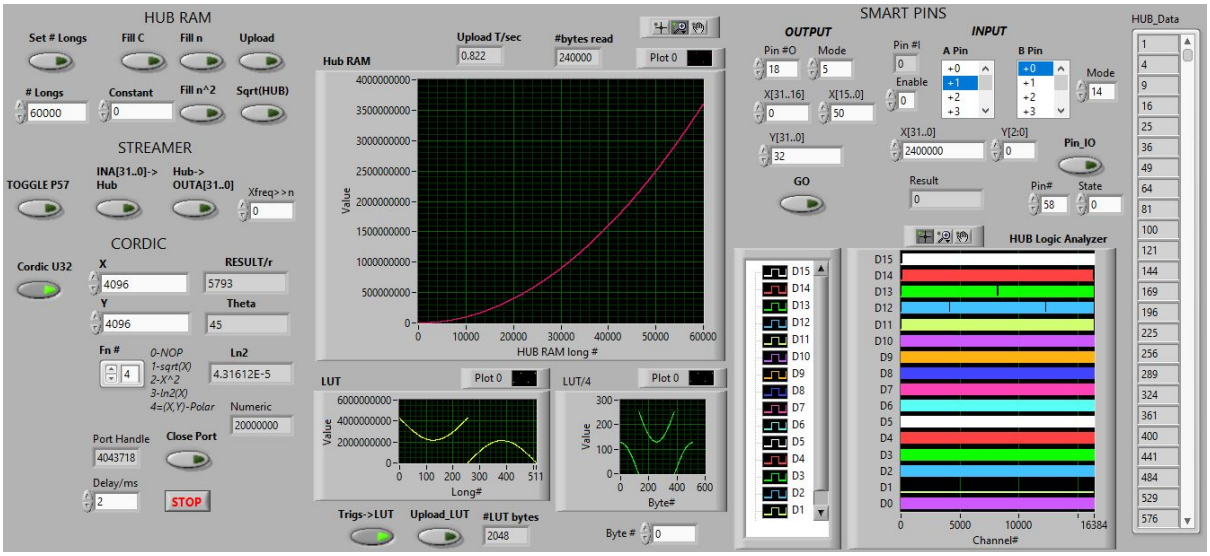


Fig. 10 The vi developed to exercise the P2's streamer and smart pin functionality.

This vi (and its associated P2ASM code) are designed to exercise the P2's smart pins and streamer; the latter moving 32 bit longs between hubRAM and the P2's I/O pins. Later in this document we will look at more specialized streamer modes that involve analog functions.

At the upper left of screen there are some buttons that are used to set the contents of hubRAM. Here we have set 60k longs in hubRAM to n^2 (n=1,2,3,…) and the result can be seen both in graphical and tabular form (the hub RAM graph window and HUB_Data array table) – after calculation on the P2 the data is uploaded to the host PC.

In addition, the 16 LSB's of each hubRAM long are displayed in a logic analyzer-style format at the bottom right of screen. Fig. 11 below shows an expansion of this region where the bit patterns clearly display the n^2 functionality.

Below the HUB RAM setting area there are two additional sets of controls – STREAMER and CORDIC. The CORDIC region allows some of the P2's CORDIC functions to be exercised. In the example shown, an X,Y pair of (4096,4096) is converted into polar coordinates – giving a hypotenuse value (r) of 5293 and an angle θ of 45^0. The CORDIC functionality built into the P2 is extremely powerful and what is shown here just barely scratches the surface, only being used for some very basic tests.

In the STREAMER area of screen, the two buttons allow data to be sent from hubRAM to the P2's pins (P31…0) (via the HUB->OutA[31..0] button) and in the reverse direction via the INA[31..0]->HUB button. In the latter case the logic analyzer screen can be used to visualize the pin states that have been acquired. In addition, the rate of streamer data transfers can be controlled via the XFREQ>>n parameter. When n is 0, a new long is output/input on each

clock cycle. Alternatively, the data rate can then be conveniently divided down by 2^n, as required by the user.

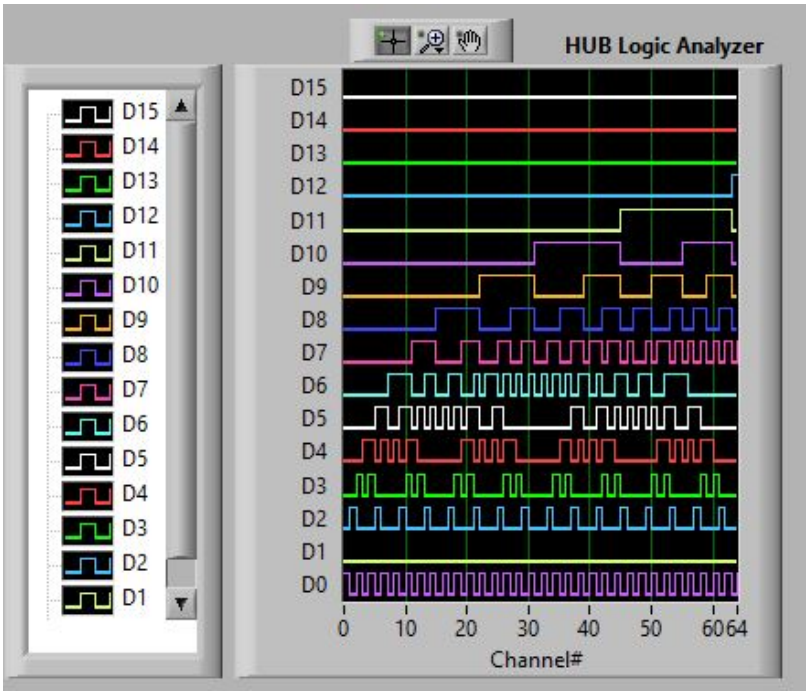


Fig. 11 Expansion of the vi's logic analyzer display area

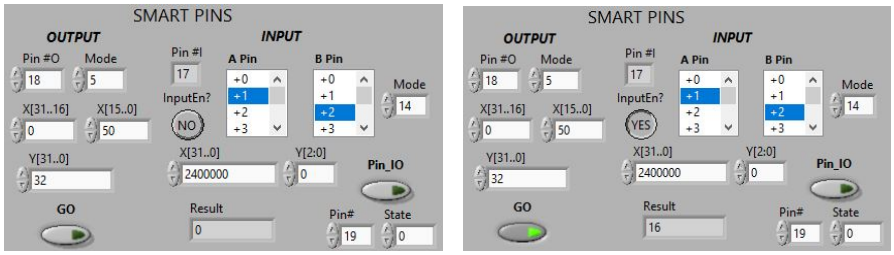
Earlier on we saw that the P2's smart pins can be configured for analog operations, but now we examine their use for digital I/O. The underlying philosophy of a P2 smart pin is that a complex task can be handed off to the smart pin which then handles that task autonomously. A smart pin's action is set by a mode value and these values can be grouped according to whether the corresponding action is an output or an input. A summary listing many of the smart pin modes is shown for reference in Fig. 12.

Output Modes :
Mode 4 = Single pulse generation; width set by Y[31..0] and X[15..0]
Mode 5 = Pulse train (transition output); Y[31..0] sets #transitions, X[15..0] width
Mode 6 = NCO freq (sq waves); Y[31..0] controls NCO rollover , X[15..0] affects high/low period
If the NCO rollover is configured to occur every n clock ticks, the high/low period will be n * X[15..0]
Mode 7 = NCO saw (inverted comb); Y[31..0] controls NCO rollover, X[15..0] sets the comb's "teeth" width
If the NCO rollover is configured to occur every n clock ticks, the spacing between leading edges of consecutive teeth will be 2^(n+1)* X[15..0] ticks
Mode 8 = PWM triangle mode; Ratio of Y[15..0] to X[31..16] sets duty cycle - H then L. Total # of clock ticks in each pwm cycle is then set by X[15..0} and is given by 2*X[31..16] * X[[15..0] ticks
Mode 9 = PWM saw mode; Y[15..0] * X[15..0] sets the high time in clock ticks. The total number of ticks in each pwm cycle is then set by X[31..16] and is X[31..16]* X[15..0]

Input Modes:
Mode 12 = Count A input positive edges when B input is high
Mode 13 = Accumulate A input positive edges; on B high increment; on B low decrement
Mode 14 = Count A input positive edges
Mode 15 = Count A input highs
Mode 16 = After state change, prior state's duration count
Mode 17 = Time last A input high state during measurement period
Mode 18 = Time X A-input highs, rises, edges (Y=0,1,2)
Mode 19 = Time X periods, Y determines the A/B rise/edge sensivity 0=r/r, 1=r/e, 2=e/r, 3=e/e

Fig. 12 A listing of some smart pin modes for digital I/O with a brief description of each

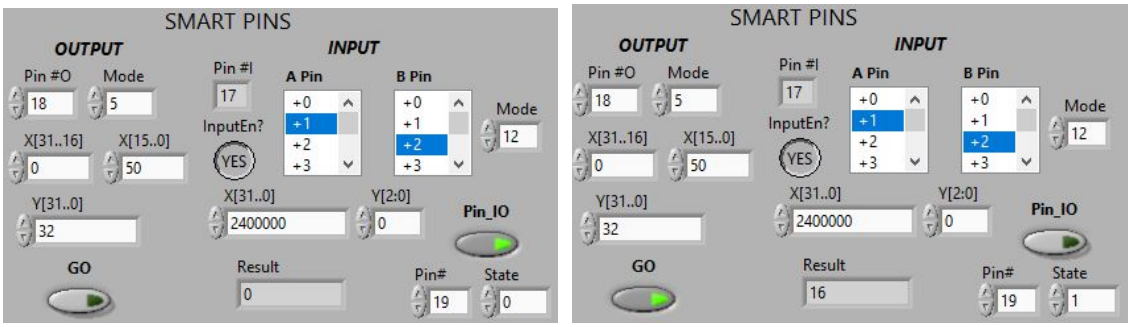
As an example, let's suppose we want to use a smart pin to generate a precise # of output pulses each having a certain pulse width. For this we need to use mode 5. If we wanted 16 pulses we would set Y[31...0] to 32 (32 transitions L-H and H-L = 16 pulses) and set our width parameter in X[15..0]. Our vi allows us to enter this information (and a pin number – Pin #O) in the OUTPUT area of screen – see Fig. 13a below.



Figs. 13a and 13b Expansion of the smart pin control area of the screen – see text for details.

Using a scope to monitor pin 18 and pressing the GO button one observes exactly 16 pulses, as requested. But a second pin could be used (this time serving as an input) to count the number of pulses being generated. For this pin we would want to use mode 14 – see Fig. 13b. By now setting the InputEn? button to "YES" we can interrogate a smart pin output on an adjacent smart pin input pin. The vi here automatically allocates our Pin #I to Pin #O – 1. By then setting the input smart pin's A Pin menu item to +1 that input smart pin will then take its input from the output smart pin selected before (i.e. pin #18). When the GO button is pressed, 16 pulses will appear on pin #18 - and in addition, the Result field now becomes 16 too. Note that when performing input operations, the counting action is performed over the # of clock cycles specified in the X[31..0] field. With a value here of 2.4×10^6 and a 270 MHz clock this corresponds to a period of 8.9 ms.

But we are not done, as there are yet further possibilities, as we see in the next example. Notice that input mode 12 counts positive edges on pin A but only while the B input is high. If we configure the input smart pin to mode 12 and set its B input menu item to +2 its B pin now becomes pin 19. The vi allows the state of any pin to be set or cleared (via pressing the Pin_IO button) and so we can run the vi twice, first clearing pin 19 and then setting it. Fig. 14 shows the results – a count of 0 in the first case (in the Result field) and a count of 16 in the second – exactly as one would expect !



Figs. 14a and 14b Results of using smart pin mode 5 for output to generate a pulse train (16 pulses) in conjunction with mode 12 for input (and using both A and B inputs) to perform

pulse counting. On the left the B pin is set low and the returned count is 0. When the B pin is set high the count is 16.

As can be seen in the above examples, my vi is crafted so that any pin(s) on the P2 can be easily configured to exercise/experiment with all of the different smart pin modes, whose action(s) can be conveniently monitored using an oscilloscope.

There are a number of other very useful smart pin modes that I've not mentioned. For example I've been using the synchronous serial transmit and receive modes (#'s 28 and 29) to implement a high speed SPI link between the P2 and an Espressif ESP32, where the latter generates the clock, chip select and data signals (serving as an SPI master) and with the P2 responding as an SPI slave. For more information on these and other smart pin modes I recommend consulting the P2 documentation.

Specialized Analog Operations using the Streamer – DDS and Goerztzel Modes

The P2 provides unique capabilities for DDS waveform generation and using its Goertzel mode, one can also track an input having the same frequency. In addition, the Goertzel mode can be used to measure the power at a target frequency in an input signal.

Quoting an explanation from Chip Gracey, the P2's lookup table (LUT) plays a central role in Goertzel operations. All four bytes (a long) read from the LUT are written to the DACs, according to %dddd – see the P2 documentation. The upper two bytes from the LUT are multiplied by the ADC input, such that 0/1 is treated as -1/+1, and the separate products are accumulated into X and Y registers, readable via GETXACC. These sums of products, after NCO rollovers, form (X,Y) coordinates, which can be converted to polar coordinates via the QVECTOR instruction, so that power and phase measurements are obtained, representing the ADC bit stream's correlation to sine/cosine patterns in the top two LUT bytes.

To investigate these capabilities further we will refer to the LabVIEW vi shown in Fig. 15. There are two controls on this screen, one called MODE (with several menu items) and another, a button, labelled Streamer->DAC. We'll start with a description of the latter as it is the simplest to understand.

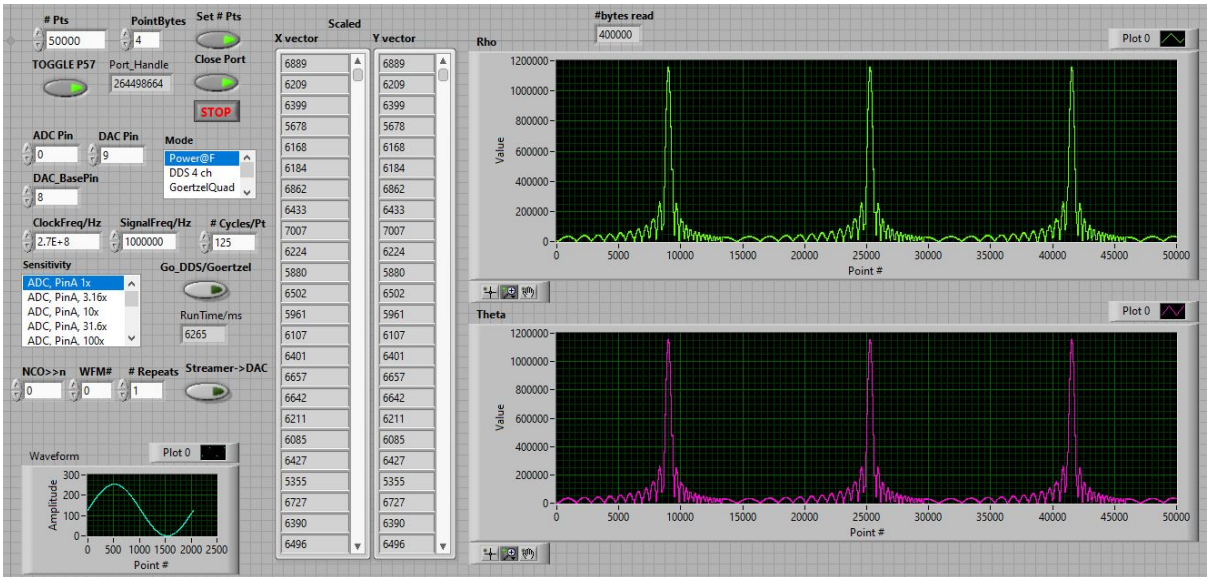


Fig. 15 A LabVIEW vi developed to explore the P2's DDS and Goertzel capabilities

To the left of the latter button are several fields, NCO>>n, WFM# and # Repeats. Here, the user can choose from a number of different waveforms (WFM# - currently these are fixed length, 2048 byte buffers) that are pre-calculated in LabVIEW. When the Streamer->DAC button is pressed, the selected waveform is downloaded from LabVIEW to P2 hubRAM, and the streamer then outputs 32x64 byte blocks of this hubRAM to a DAC (on DAC Pin); this is then repeated for # Repeats cycles. For example, the scope trace in Fig. 16 shows an output of waveform #1 (a sine wave multiplied by a decaying exponential function) repeated 6 times with an NCO>>n setting of 0 giving one point every clock cycle. With n=1, output is every 2nd clock, with n=2, every 4th and so on.

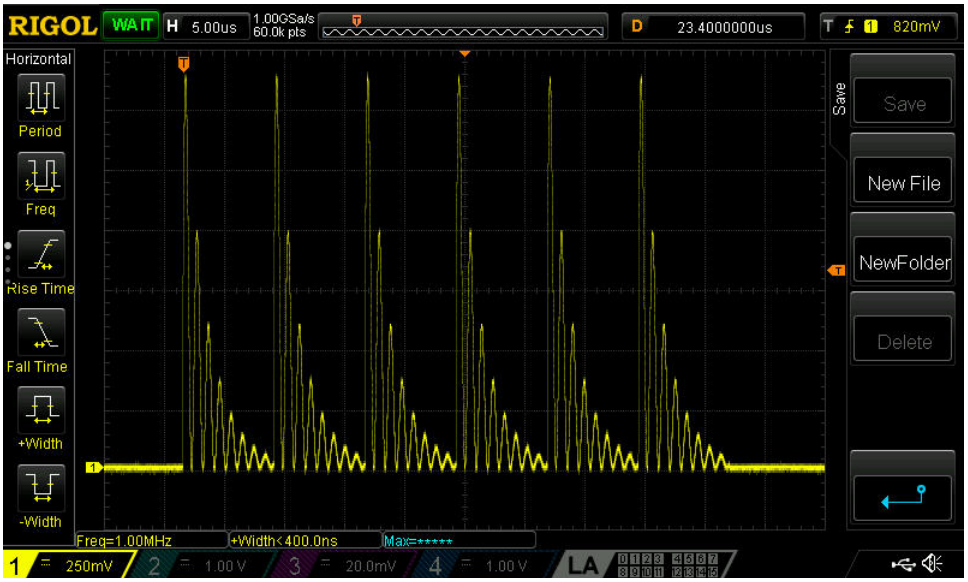


Fig. 16 Waveform generation – here a "chirp" waveform calculated in a LabVIEW vi is downloaded to the P2 and output to a DAC pin using the streamer, with a total of 6 repeated cycles.

Waveforms generated in this manner are limited only by the size of hubRAM. Complex waveforms can be calculated in LabVIEW and downloaded or alternatively, the P2's Cordic functions could be used to calculate them directly in the P2.

Next we'll look at options available in the MODE menu. The first one allows us to measure the power at a specified frequency (Signal Frequency/Hz field) when an input signal is connected to a user-specified analog input pin (ADC Pin). We also need to specify a number of cycles for the measurement (#cycles/Pt). If we are probing for a 1 MHz signal and cycles is set to 125, each point will take 125 usec. This mode of operation is selected by choosing Power@F, then pressing the Go button.

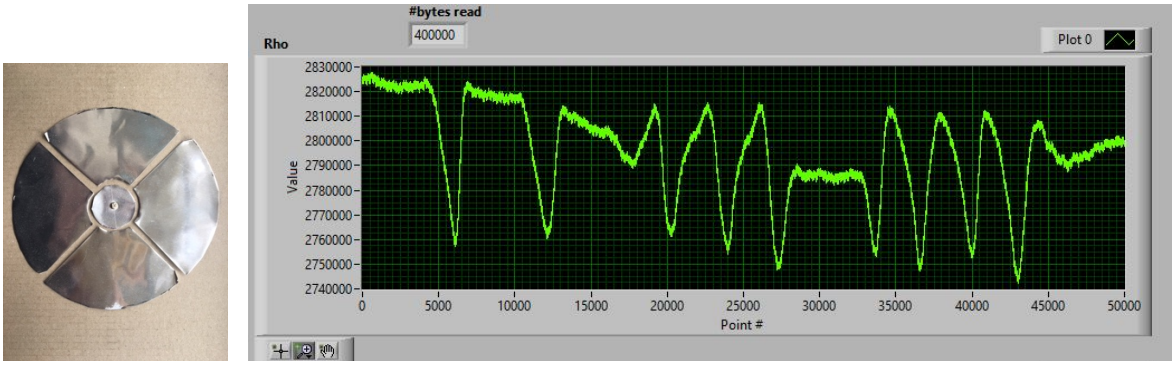
The plots in Fig. 15 show the results obtained when selecting 1000 kHz as the target frequency, then applying a frequency sweep covering 900 kHz to 1100 KHz and lasting 2 seconds. Here 50000 points have been acquired in a total run time of ~6.2 seconds. Over this

acquisition period the sweeper makes three complete passes and the three repeated traces seen in Fig. 15 show the response of the input smart pin - peaking each time the sweep passes through exactly 1 MHz. Note the side lobes in the traces here. Chip has pointed out that the Q of this measurement system can be improved by using the SINC2 mode.

The next item in our Mode menu is DDS 4ch – this continuously outputs 4 x 8 bit waveforms that are pre-calculated in P2's lutRAM space onto 4 consecutive DAC pins beginning at DAC_BasePin. A scope trace is shown below in Fig. 17.

Fig. 17 A scope trace acquired while the P2 is running in 4 channel DDS mode, this time outputting waveforms pre-calculated in the P2's LUT.

Finally, we have the Goertzel quad mode that brings together a number of the aforementioned features. Here, a set of 4 DAC's operate in quadrature at the user's nominated frequency. In a gesture recognition application (first demonstrated by Chip Gracey) these DAC's are connected to a set of 4 outer electrodes and a fifth inner electrode is connected to a pin operating as an ADC channel. Fig. 18a shows my home brew electrode assembly - fashioned from aluminium flashing material. Now, the electrode to which the ADC pin is connected serves as an antenna, tuned to the DAC frequency, and as a finger is moved above and around the electrode assembly, as well as closer and further away the coupling of energy into the antenna is affected.



Figs. 18a and 18b Home brew electrode set for Goertzel quad experiments (left) and data trace as my hand/finger are moved above and around the electrodes (right).

The ADC power trace then provides a visual representation of the motion detected. Fig. 18b show some typical results. Here we have 50000 points accumulated over 15 seconds, with F = 1 MHz and 300 cycles/point selected for the measurements. A Sensitivity selection of ADC, PinA, 10x was made here to improve the quality of the measurements. For this test my extended right hand was lowered over the electrodes, then raised; this was done twice. My finger was then moved in a circular orbit a few cm above the electrodes for 3 orbits, first in a counter-clockwise direction, followed by a brief pause and then repeating this in the reverse (clockwise) direction. Each of these movements can be recognized in the above trace, even down to finer details (notice the shape of the peaks during the finger orbits allows the direction of motion to be distinguished). My current setup is very crude (with quite long clip leads from the electrodes to P2's pins and this could be greatly improved upon with a PCB design.

Scope mode

One of the very exciting features of the P2 is its provision of a 4 channel "scope" mode. Here, 8 bit data from a group of 4 ADC pins can be streamed directly into hubRAM at speeds as high as one set of 4 samples every clock. As in previous examples, the streamer data rate can be adjusted by changing the NCO>>n parameter. Fig. 19 shows a vi developed to experiment with P2's scope mode. Here a function generator producing a 600 kHz square wave is connected at CH0 (here on ADC pin 8), with CH1 connected to ground, CH2 to Vcc (3.3V) and CH3 floating. With an NCO>>n setting of 1, these channels are sampled every 7.4 ns.

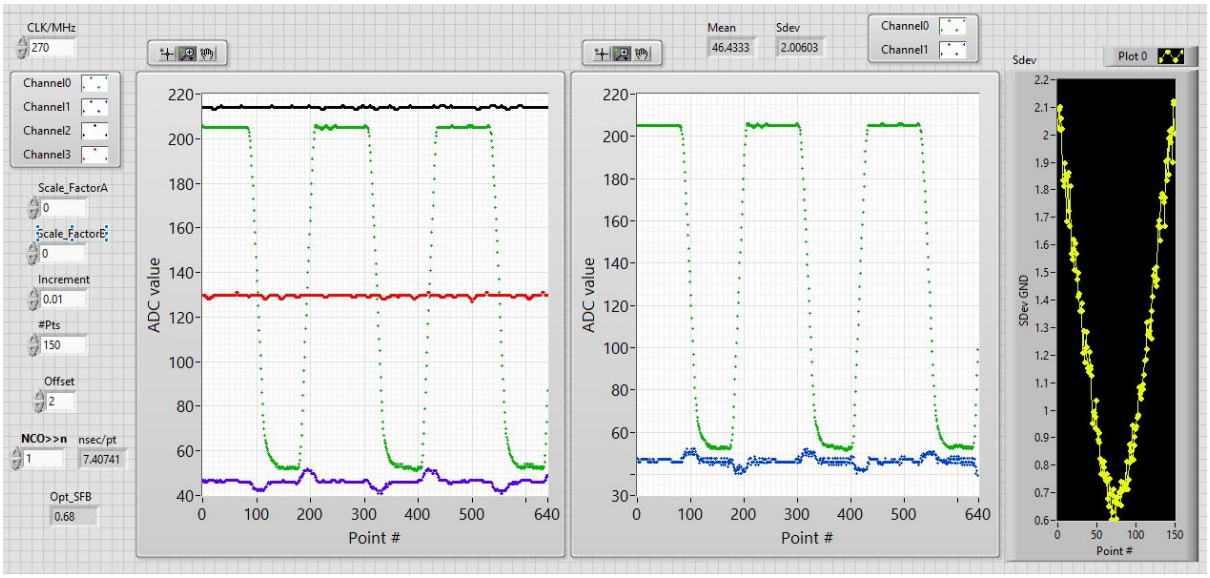


Fig. 19 The P2 scope mode vi. Here, 4x8 bit channels are recorded at 135 MHz. Operation at the full P2 clock speed (here 270 Mhz) is possible.

Looking at the CH1 ground (purple) trace in the left-most graph we can see that there is some interference when CH0's dV/dt is large; i.e. when CH0 is changing rapidly. It has subsequently been realized that as a result of the P2 chip layout there is coupling between adjacent pins - and fixing this problem will require a new iteration of the P2 (P2 revC). For now, the workaround is to ensure that an adjacent channel is idle during a measurement, or alternatively that the signal of interest is connected to both channels (here CH0 and CH1).

Using the vi presented in Fig. 19 we can test a software algorithm to improve the data quality that has been discussed on the Parallax forums. The approach is to calculate differences between adjacent samples for each channel (these are proportional to dV/dt) and then subtract some fraction of those differences (a scaling factor) from each other channel. The vi in Fig. 19 allows the optimum scaling factor to be determined. It includes an offset field that controls which points are used in the difference calculation. The scaling factor to then be applied at point #n is calculated using the difference between points n-offset and n-offset-1.

While measuring, the vi tests a range of scaling factors (using the #Pts and Increment values) while determining the standard deviation in the ground signal, plotting this in the yellow trace at right of screen. The scaling factor at which the standard deviation is minimized becomes the optimum value. In this case (with offset = 2 and NCO>>n = 1) the correction formula for

CH1 becomes CH1(corrected) = CH1(measured)-0.68*ΔCH0(measured:n-2,n-3). Fig. 20 shows the result of applying this correction to CH1 (the ground signal) as captured from the middle plot in Fig. 19. The standard deviation in the ground signal is now reduced from 2.1 ADC counts to 0.7 when this correction is applied – a 3-fold improvement.

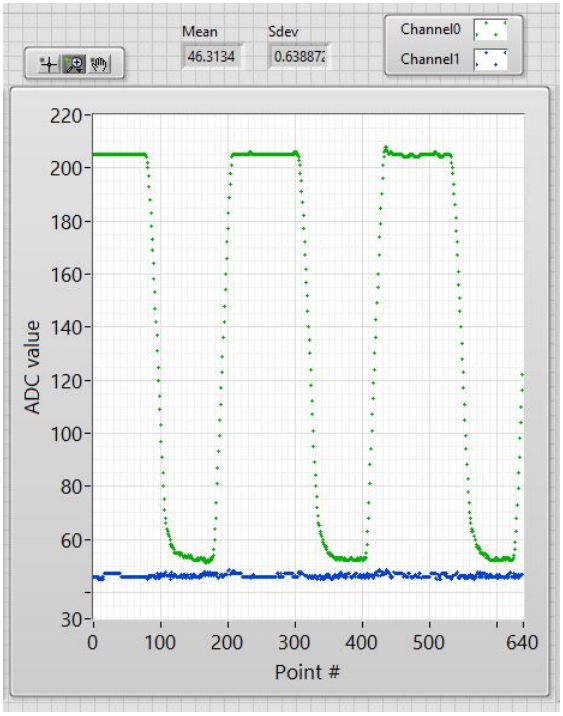


Fig. 20 Data illustrating clean-up of a ground signal (blue trace) using a software algorithm (see text for details). The green trace is a square wave applied to an adjacent pin.

Although this method is effective it does depend on the sampling rate, the offset and on the precise nature of the signals on adjacent channels and consequently a P2 trace layout rework is essential to effectively eliminate this problem. Even now one can see the huge potential of P2's scope mode, which is going to be an outstanding distinguishing feature of the P2 chip.

USB Capability

The P2 chip provides a smart pin mode (#27) in which two adjacent pins (numbered even/odd) are configured to become a USB pair, capable of operating at a 12 MHz baud rate. I have not personally used this mode but a contributor on the Parallax forums, garryj has developed some great code to allow direct connection of a mouse and keyboard to the P2.

Video Output

A large code base has been developed already to allow the P2 to display output to various video devices including VGA, HDMI and NTSC. Frame buffers using HyperRAM have also been successfully implemented by rogloh, ozpropdev and Rayman.

Having access to LabVIEW I have not so far had occasion to investigate all of these capabilities but when video is combined with USB the P2 becomes a completely stand-alone platform for power computing, with applications as diverse as gaming or high performance instrumentation.

Concluding Remarks

For now, that concludes my initial evaluation of the P2. In the work I've done to date I've been particularly struck by how easy it has been to carry out the many different experiments reported here. A large variety of complex tasks can be performed without any additional hardware and usually with very few lines of code. Having the smart pins (with so many different modes) and a streamer capable of running at up to the full clock rate will make many new things possible. The array of analog options is quite exceptional, from very high speed 8 bit acquisitions to lower speed, higher resolution options on every P2 pin.

For mine the P2 is proving both fun and a joy to program and with its dazzling suite of hardware capabilities I'm sure it will be a perfect fit in many of my upcoming scientific instrumentation projects. Chip, Ken and the whole Parallax team can take great credit for a wonderful achievement in having the foresight, patience and dedication to create the P2.