FFT for arc-fault detection in PSoC™ 4

About this document

Scope and purpose

A two-channel, 64-point Fast Fourier Transform (FFT) is implemented with continuous data acquisition during a single cycle of the AC line with FFT output on the next cycle of the AC line. An archived project is included in the download. This app note describes the associated project and code in detail.

Intended audience

This application note is intended for FAEs and customers who need to create a two-channel FFT for analyzing the harmonics of the line frequency, looking for signatures that are representative of series arcing.

Table of contents

About this document **1**
Table of contents **2**
1 Introduction **3**
1.1 FFT outline 3
2 PSoC™ Creator project **5**
2.1 Input buffers 5
2.2 ADC 6
2.3 PWM trigger 7
2.4 UART 7
3 Code **8**
3.1 *main.c* 9
3.1.1 Interrupt 10
3.2 *fft.c* 10
3.3 Monitoring 11
4 Conclusion **13**
Appendices **14**
A Appendix 1: Kit realities 14
B Appendix 2: Data display TeraTerm to Excel 14
C Appendix 3: Anti-aliasing 16
D Appendix 4: Expanding frequency range 18
E Appendix 5: Quick root 18
References **19**
Revision history **20**
Disclaimer **21**

1 Introduction

Arc fault is a well-known failure mechanism in residential and commercial AC wiring systems. Arc can occur when insulation is broken down and parallel conductors are shorted, this is called “parallel arc fault”. Arc faults more commonly occur when the conduction path develops a gap, called a “series arc fault”. As the voltage rises from the zero-voltage crossing, the voltage can be large enough to jump the gap, creating an arc, which can be dangerous. Arcing results in high-intensity heating and can ignite surrounding materials.

NEC 210.12(A) of the 2017 National Electrical Code (NEC) requires Arc-Fault Circuit Interrupter (AFCI) protection for branch circuits supplying all newly installed 120 volt, 15 and 20 amp outlets or devices in areas of a dwelling.

Arc faults can be observed by monitoring the current or line voltage; current monitoring is more common. The signature of the arc fault depends on the type of the nominal load connected.

A typical example is a household toaster, dominantly a resistive load because of the heating element.

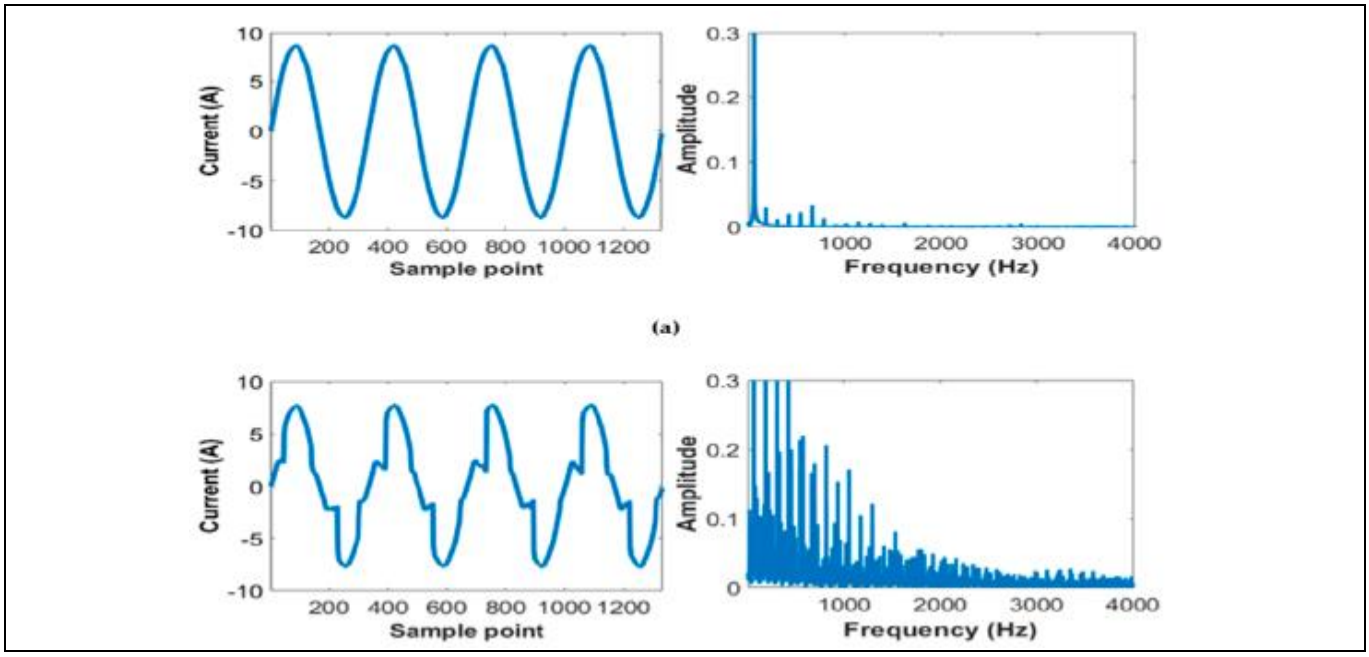


Figure 1 Normal and arc fault current waveform, resistive load

Infineon’s PSoC™ microcontrollers are particularly well-suited to this application because of high quality analog front-end, high-speed 12-bit ADC, and high-speed 32-bit Cortex®-M0 processor.

1.1 FFT outline

A Fast Fourier Transform (FFT) provides a frequency-based analysis of a measured signal with a number of equally spaced frequency bins. Each bin is the width of the length of the sample, in this case 60 Hz. The frequency range is half of the sample rate, in this case 128 samples per 60 Hz cycle (7680 Hz) resulting in 64 bins from 0 to 3680 Hz.

An ADC acquires the values on each of two channels and stores them into buffers. In the same interrupt, two 64-point FFTs buffered during the previous cycle are delivered to the UART, while the main line code simultaneously calculates the FFTs on the two buffered ADC channels and pushes them to another pair of buffers.

Introduction

A Google search reveals a number of FFT implementations in Infineon (formerly Cypress) application notes and independent journal articles. They use PSoC™ 1 (CY8C29466) and PSoC™ 5 LP (CY8C5868, using CYKIT-059). These earlier implementations were not optimized to provide cycle-by-cycle FFTs on 60 Hz waveforms. The intent of this exercise is to provide a high-speed, low-cost FFT capable of completion within a single cycle of the 60 Hz line built using PSoC™ 4.

2 PSoC™ Creator project

This relatively simple design was created in CY8C4247AZI-M485 using PSoC™ Creator 4.2. It does not stress the analog and digital hardware capabilities of the chip.

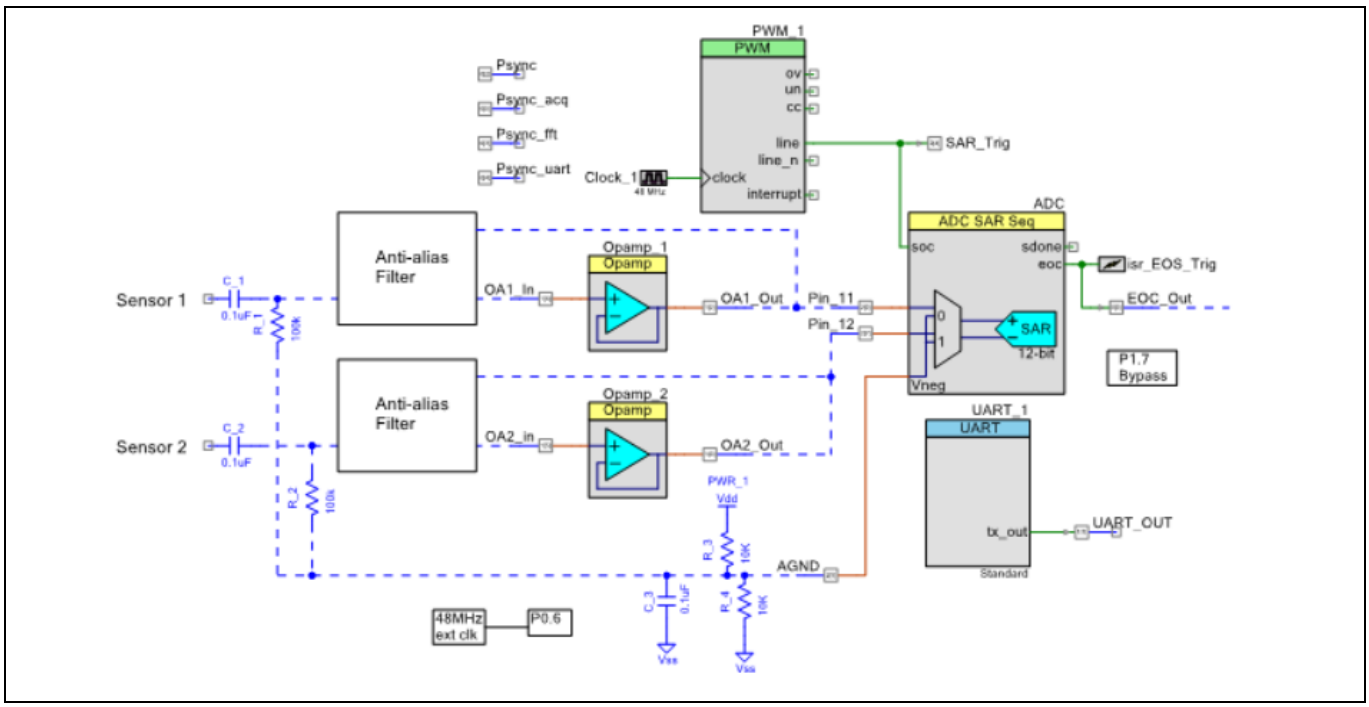


Figure 2 PSoC™ Creator project

2.1 Input buffers

Each of the two input buffers are connected to current sensors (typically Hall-effect sensor or current transformer), with signals capacitively coupled and biased to analog ground (AGND) at mid supply. AGND is generated with a bypassed resistive divider from Vdd. The outputs of the buffers can be routed directly to the ADC inputs, but use of external pin connections enables later installation of anti-alias filters.

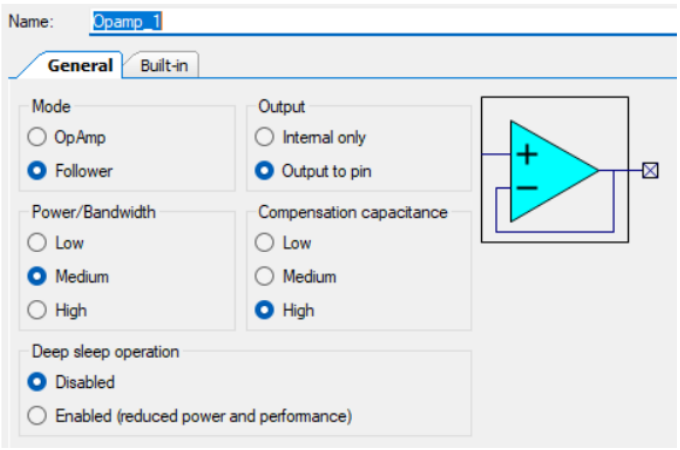


Figure 3 Input buffer configuration

PSoC™ Creator project

The signals are relatively slow (nominal 7.8 kHz max) so Power/Bandwidth is set to **Medium**. The opamp buffer outputs are connected to pins (GPIO) which are externally routed to ADC inputs. External routing is used so that the buffer can be used as part of a signal-conditioning filter.

2.2 ADC

The ADC is a 12-bit SAR with differential input on each of two channels. The negative input is external, referenced to AGND. The ADC output range is signed, +2047/-2048 counts (+/- 11 bits on +/- 2.5 V) delivered as 2's complement.

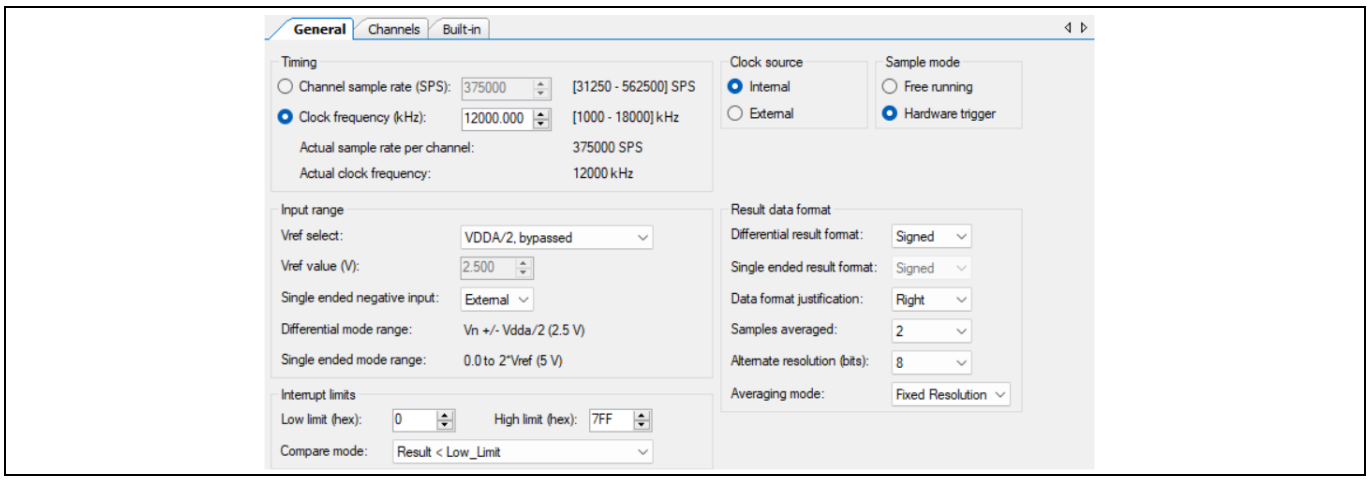


Figure 4 ADC General configuration

The clock is internal at 12.0 MHz. The start of conversion is external, derived from a PWM as detailed in the following section.

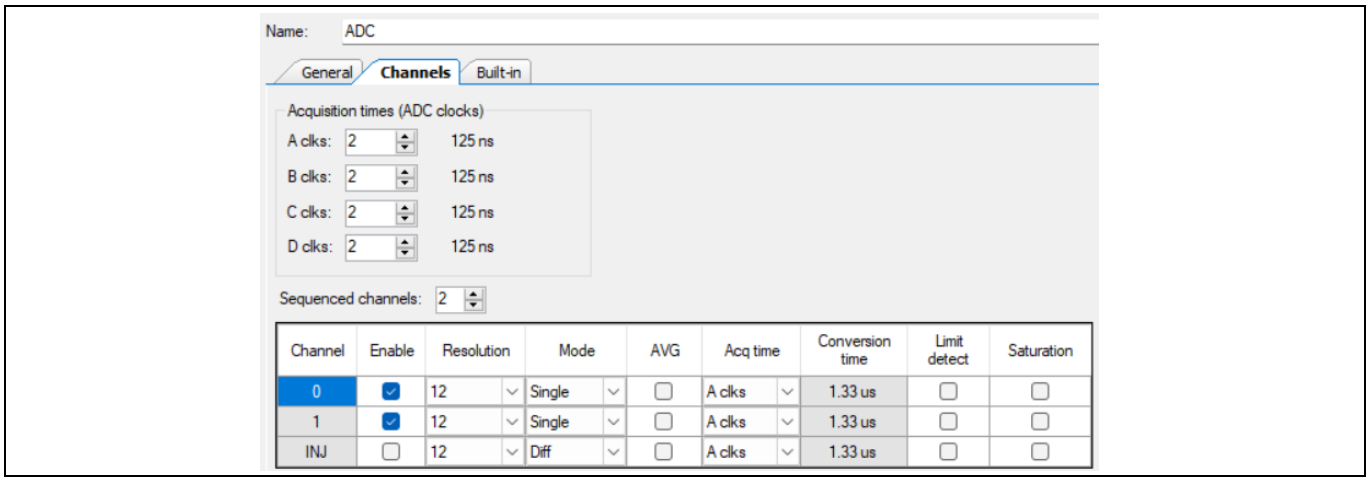


Figure 5 ADC Channels configuration

The total conversion time is 2.66 µs per sample pair, set by the ADC clock and the acquisition time. The acquisition time is determined by settling at the ADC input. This acquisition time constant is the product of the input capacitance of the ADC (approximately 10 pF) and the series charging resistance through the input multiplexer, on-chip routing, and the output impedance of the buffer. The input to the ADC must settle to less than a single bit value during the acquisition time; this takes 9 time constants. The 125 ns acquisition time allows more than enough time to meet this requirement. The two channels are measured sequentially; there is no need for simultaneous sampling because the two input signals are independent measurements.

PSoC™ Creator project

The End of Convert (EOC) out goes high for one ADC clock cycle after both samples have been completed. This triggers the interrupt. This signal is also routed to a GPIO for timing observation.

While not explicitly stated on the generated PSoC™ Creator schematic, the ADC reference is bypassed by a 1.0 µF capacitor to enable fast and accurate conversions.

2.3 PWM trigger

Start of conversion is triggered by the rising edge of the line output of PWM_1. PWM_1 divides the 48 MHz clock by 6250 (period value +1) to yield the ADC sample rate of 7680 (60 Hz * 128) sample pairs per second. The PWM compare value is set arbitrarily to 100 yielding a 2.2 µs pulse. This is long enough to be easily seen on an oscilloscope and has no impact on the trigger time. The PWM runs continually once started. PWM interrupts are not used.

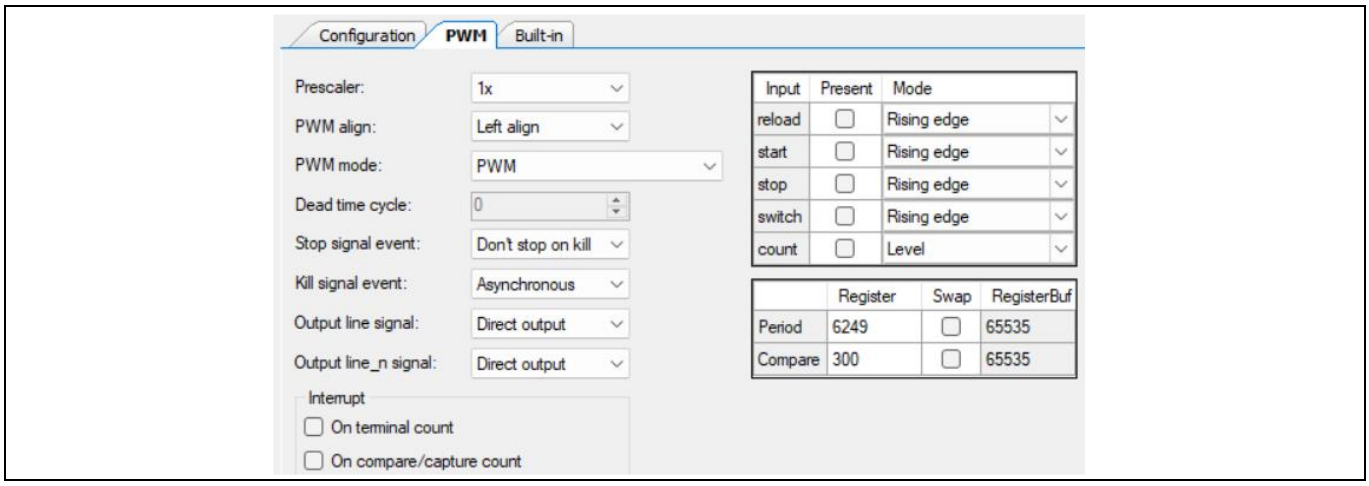


Figure 6 PWM configuration for ADC sampling

2.4 UART

The UART is configured as TX only in the simplest possible form: 8 data bits, no parity, 1 stop bit.

The default Over-sample option results in the closes actual baud rate to the design value. The designed baud rate is 230400 baud; the realized baud rate is 230769. This is 0.16% from nominal, close enough for any receiving terminal.

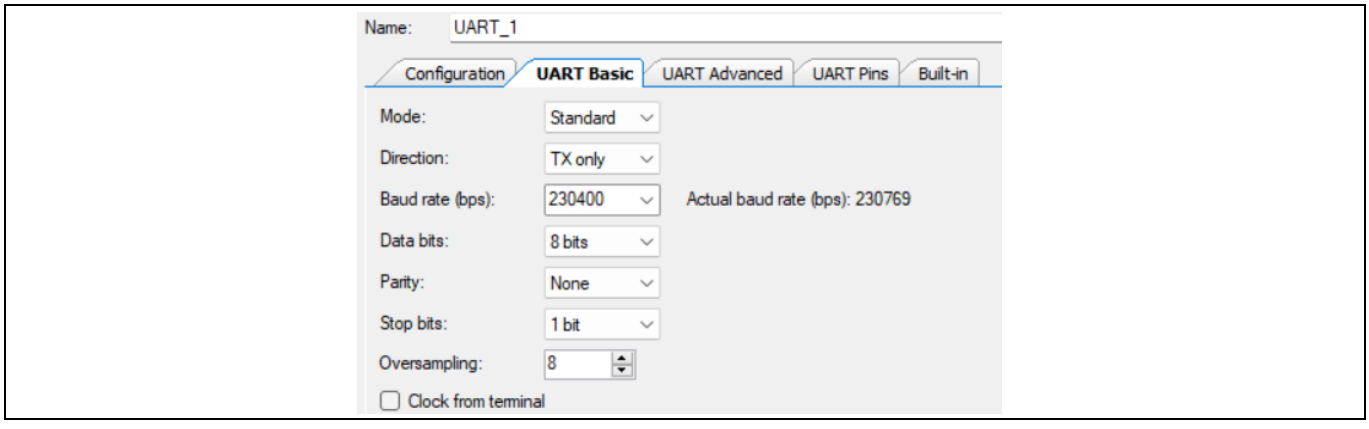


Figure 7 UART Basic configuration

3 Code

The system acquires ADC values from two channels sequentially (and almost immediately). ADC data is pushed to a set of buffers. At the end of cycle acquisition (the same point in the 60 Hz sine wave) the buffered ADC values are loaded into the FFT routine. The magnitude at each frequency point is calculated for arc-fault detection using harmonic analysis. The data is delivered to a UART for examination, but the arc-fault detection is not expected to output cycle by cycle FFT data.

The code is written in C. The code from which the FFT routine was adapted reused the variable (same input and output on function) to save memory space. With the CY8C4247, RAM is not an issue, so this part of the code was copied directly to eliminate the risk of transcription errors.

The FFT function is called twice, once for each channel. The input and output variable arrays for these routines have distinct names rather than using a two-dimensional array (e.g., array[n][p]) in order to save a small amount of memory access time. The entire code was written to optimize for execution time. FFT calculations are mathematically complicated. Effort was made toward efficiency in order to allow time for the algorithm that actually determines the presence of arc-fault from the acquired data.

The following flow charts outline the operation of the code. The complete code is listed in the project archived with this document. Snippets and comments in this note are intended to be a guide to the user in understanding the complete code.

Code

3.1 *main.c*

The code starts with the usual set of #defines, variable definitions, interrupt enable and start commands.

128 samples are acquired by the ADC for each channel. When the sample buffers are full, the interrupt sets `endloop` to 1.

When `endloop = 1`, data handling and FFT calculation run in *main.c* except as interrupted to gather new data and transmit FFT values via UART.

First, `endloop` is set to 0. The root value output array values from fft.c are transferred to a pair of buffers .

`realNum0` and `realNum1` arrays are set equal to the buffer values from the ADC acquisition in the interrupt routine; the imaginary number arrays are set to zero.

The FFT function is called for each pair of arrays: `realNum0,1` and `imagNum0,1`, returning FFT values in the same arrays.

For each real and imaginary pair in the two arrays, the magnitude is calculated in the main loop, then the square root is calculated, resulting in arrays `iroot0[..]` and `iroot1[..]`. These will be buffered the next time around, when `endloop` is found to be equal to 1 again.

The square root routine is a standard one, readily found in a Google search. An alternative is to use an approximation to the square root found in Appendix 7. This reduces the square root time from 1.4 ms to 290 µs at the cost of +/- 5% error.

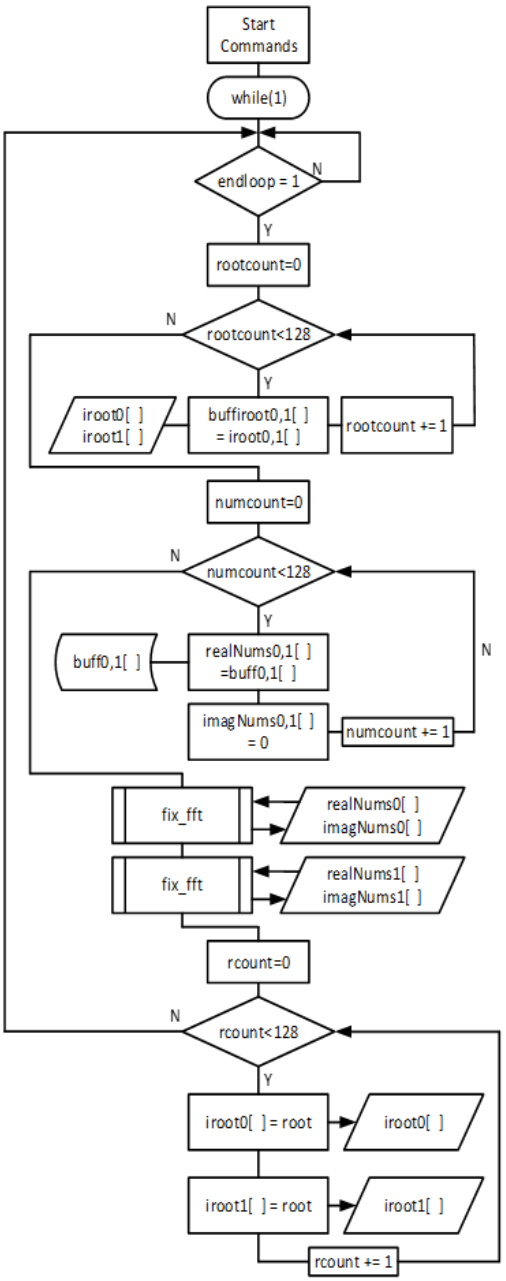


Figure 8 *main.c* flow chart

3.1.1 Interrupt

The interrupt handler is contained within *main.c*.

The interrupt routine has four functions:
1. Keep track of sample count
2. Download ADC results into a pair of buffer arrays
3. Format the FFTs' root data arrays using sprintf
4. Send the FFTs root data arrays to the UART.

One pair of input values are downloaded and one FFT value is transmitted on each interrupt. The FFT output is for half the number of input samples, so FFT0 and FFT1 arrays are transmitted sequentially.

First, the interrupt is cleared:

ADC values are read into buffers: buff0[..] and buff1[..]

When `icount` has recycled and equal to 0, FFT0's header is sent to the UART. The first 64 values of the buffered square root output (for the first channel) are formatted and sent to the UART, one value per interrupt cycle. This uses the same index as the ADC sample.

When `icount = 64`, FFT1's header is sent to the UART. The first 64 values of the buffered square root output (for the second channel) are formatted and sent to the UART, using the same index as the ADC sample, offset by 64.

When `icount` reachs 127 (the last sample), `icount` is set to 0 and `endloop` is set to 1, signaling main to move the ADC buffers and perform FFTs for delivery by the interrupt on the next cycle.

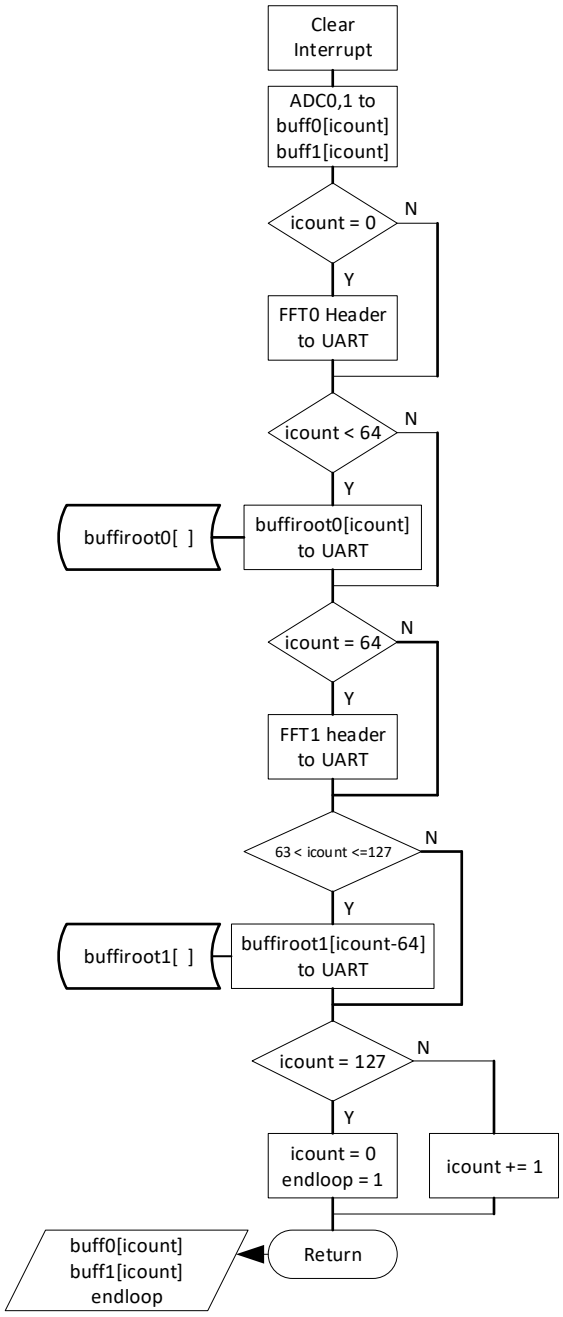


Figure 9 Interrupt flow chart

3.2 *fft.c*

This part of the software was adapted from open-source work. The original code was written in a way to minimize RAM consumption due to the amount of array usage. The code displays 32-bins of audio FFT data at 10 frames per second, nowhere near fast enough for cycle by cycle analysis.

This PSoC™ version must have twice the number of bins (64 outputs on 128 samples per channel) and must be much faster because the FFT is computed on every line cycle – not averaged. Using a 32-bit processor at 48 MHz and minor modifications to the original code, the PSoC™ FFT for a single channel is executed in about 4.3 ms for the complex (i.e., real and imaginary number arrays) FFT result. It takes another (average) 1.4 ms to compute

the square root of (real^2 + imag^2) to get magnitude. The result is a pair of 64-bin FFTs (60 Hz bins 0 to 3.84 kHz) in 11.4 ms, 17 times faster (i.e., PSoC™ >> PIC) for twice the FFT size and twice as many FFTs. This still leaves adequate time for making the decision on arc-fault within a single AC line cycle. The actual FFT computation time is probably considerably faster, but it is continuously interrupted to acquire ADC data and format output data to the UART.

fix_fft is called from *main.c*. Psync_fft is toggled high before the function call, then low afterward to demonstrate the calculation time.

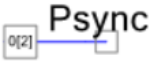
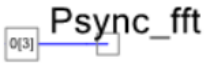
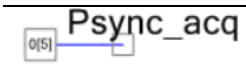
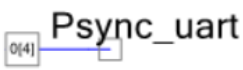
The FFT is executed in a series of non-obvious loops. The FFT routine (roughly) cross-multiples input samples with a sync'd value from a sine table, adding results into real and imaginary parts. The Sine Table is contained within fft.c. The table covers ¾ of a cycle with ¾ of 1024 values and 16-bit precision. There is a slight improvement in execution using the sine table in RAM (in *fft.c*) compared to having the sine table in a *.h* file.

The same function is called for both FFTs, array of `realNums0[..]` and `imagNums0[..]` and array of `realNums1[..]` and `imagNums1[..]`.

3.3 Monitoring

An array of synchronization outputs is provided. These are controlled in software in order to monitor and measure the amount of time the devices spends in each part of the code.

Table 1 Synchronization outputs

Sync outputs	Description
	Toggled high at the start of the 128-sample loop (60 Hz). Remains high while FFT output data is fed to buffer, t = 292 µs
	Occurs once per 60 Hz cycle in *main.c* Toggled high while interrupt-generated data is fed to buffer, t = 420 µs Toggled high while first FFT executed, t = 4.3 ms Toggled high while second FFT executed, t = 4.3 ms Toggled high while first FFT sqrt is executed, t = 1.8 ms Toggled high while second FFT sqrt is executed, t = 1.0 ms
	Toggled high during ADC acquisition, t = 4.0 µs, occurs with period of 130.2 µs = 1/sample rate
	Toggled high in interrupt at start of data formatting for each sample, remains high while data for that previously buffered sample is written to UART, toggles low at end of UART write, t = 35 µs average
	Raw UART data time varies depending on length of data byte.

A single cycle (16.67 ms) shows the overall timing. The expanded view shows the busy section at the start of the cycle.

Code

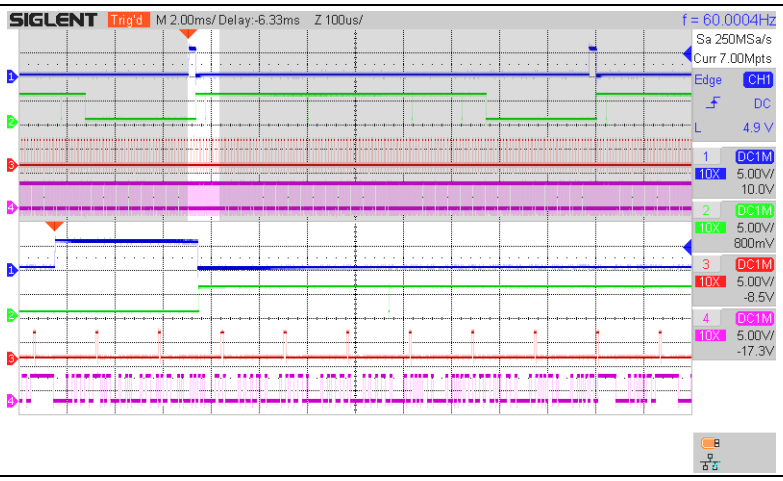


Figure 10 Sample and UART timing

- Trace 1: Psync: Moves iroot values to buffer, includes time for acq' interrupts
- Trace 2: Psync_fft: First 396 µsec moves buff[i] data to fft input buffer
- Trace 3: Psync_uart: Moves buffered sqrt values to UART
- Trace 4: UART_OUT: Busy at the start, sending CRLF; more dead time when UART buffer catches up

4 Conclusion

The FFT data streaming to the UART to USB is available now. The FFT calculation for both channels takes a total of 11.7 ms. This includes interrupts for transmitting data to UART while the FFTs are being calculated. In the time after FFT completion (16.67 ms – 11.7 ms), writing to the UART consumes 35 us out of 130 us (1/7680) or 27%. This leaves 3.6 ms per cycle to evaluate the data to determine the presence of an arc fault.

There is significant diversity in methods of arc fault detection The presumption in this effort is that the bulk of the arc signature will be harmonics of the line. Other methods for arc fault detection include looking at higher frequency current spectra around the zero crossing (typically measured in the neutral). Further development will be done on these diagnostic techniques.

Appendices

A Appendix 1: Kit realities

This project was originally written for the PSoC™ 4100PS, CY8C4145LQI-PS433 in CYKIT-147. When the number of FFT bins was doubled to extend the frequency range, the available RAM was more than used up, so the project was ported to PSoC™ 4M, CY8C4247AZI-M485 in CYKIT-043, which has the same hardware resources (except the Universal Analog Block) but four times the RAM and Flash.

Kits -147 and -043 share a common problem. The UART to USB function is contained in the Mini-Prog3 (with CY8C5838LTI-LP039) on the board to control programming and interface to USB. The Mini-Prog3 uses the on-chip internal main oscillator (IMO) to control data speed. At higher rates the IMO is not sufficiently accurate and the UART loses sync with the external terminal.

This problem is resolved by removing the UART to USB tab from a CYKIT-049 and wiring it to the power and Tx pins from the CYKIT-043. This is proven to support data rates up to 430.8 kBaud.

B Appendix 2: Data display TeraTerm to Excel

1. Connect a signal generator to the two inputs, set for 3.0 Vpp at 360 Hz.
2. Connect the UART to USB on the KitProg or other connection. In this case, a serial port to USB on COM26 is used. This may change in your system.

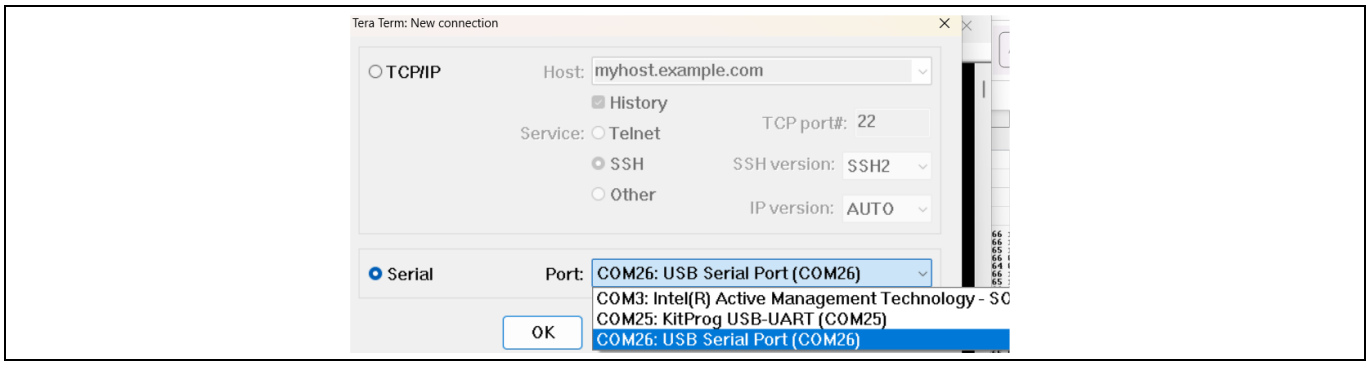


Figure 11 Tera Term Port selection

3. Configure the serial port for 230400 baud, 8 bits, no parity, 1 stop bit.

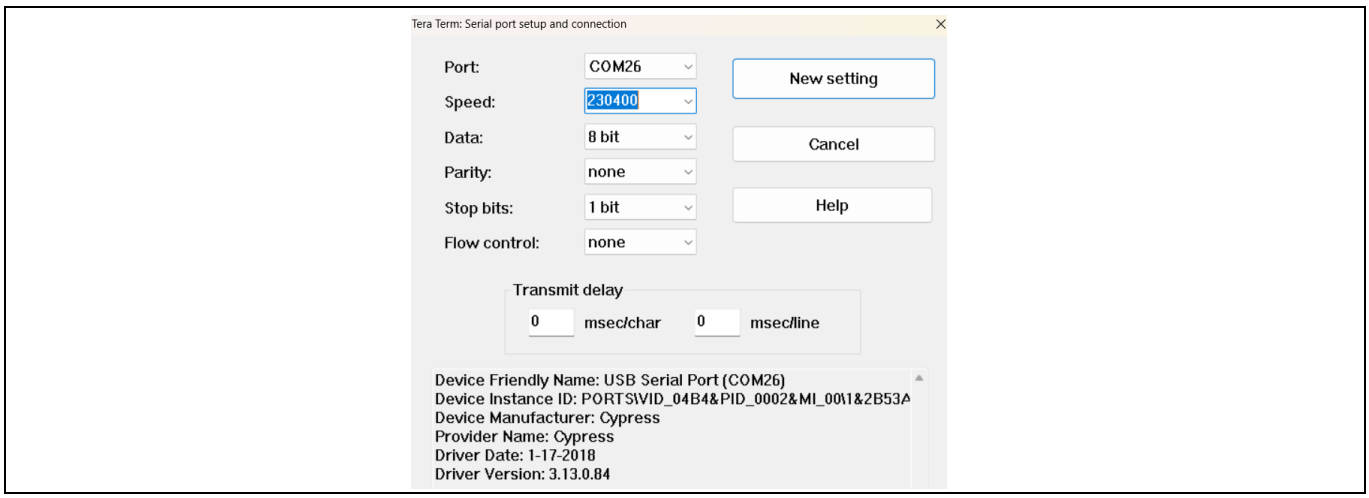


Figure 12 Tera Term Port configuration

Appendices

The data will now stream to the PC, alternating lines between FFT0 and FFT1.
The FFT bins are numbers 0 through 63, for center frequencies 0 + 60 Hz*bin#.
A sine wave at 360 Hz, 3.0 Vpp from a signal generator is applied to both inputs. FFT0 is direct, FFT1 has an anti-aliasing filter. The peak signal is expected in bin 9. The signal levels may be different because of the ripple in the anti-aliasing filter.

```
FFT0 5 4 3 3 3 3 2 3 553 3 3 4 4 4 4 4 3 4 3 3 3 3 2 3 3 3 3 4 4 4 4 4 4 4 3 3 3 0 2 3 3 3 3 4 4 4 4 4 4 4 3 3 3 3 2 3 2 3 3 4 4 4 4 4
FFT1 4 4 3 3 3 3 2 3 504 3 3 4 4 4 4 4 5 4 3 3 3 3 2 3 3 3 3 4 4 4 4 4 4 4 4 3 3 3 3 2 2 3 3 3 3 3 3 3 3 3 3 4 4 4 4 4
FFT0 6 4 3 3 3 3 2 2 553 4 3 4 4 4 4 4 3 4 3 2 3 2 2 3 3 3 3 3 4 4 4 4 4 4 4 3 3 3 3 2 2 2 3 3 3 4 4 4 4 4 4 4 3 3 3 2 2 2 3 3 3 4 4 4 4
FFT1 4 4 3 3 3 3 2 3 504 3 3 4 4 4 4 4 5 4 3 3 3 3 2 3 3 3 3 4 4 4 4 4 4 4 4 3 3 3 3 2 2 3 3 3 3 3 3 3 3 3 3 4 4 4 4 4
FFT0 6 4 3 3 3 3 2 3 552 3 3 4 4 4 4 4 3 4 3 3 3 3 2 3 3 3 3 4 4 4 4 4 4 4 4 3 3 3 0 2 3 2 3 3 4 4 4 4 4 5 4 3 3 3 3 2 2 3 3 4 4 4 4 4
FFT1 4 4 3 3 3 3 2 3 504 3 3 4 4 4 4 4 3 4 3 3 3 3 2 3 4 3 3 4 4 4 4 4 4 3 4 3 3 3 3 2 3 3 3 3 4 4 4 4 4 4 4 3 3 3 2 3 3 3 3 4 4 4 4 4
FFT0 5 4 3 3 3 3 2 3 553 3 3 4 4 4 4 4 3 4 3 3 3 3 2 3 5 3 3 4 4 4 4 4 4 3 4 3 3 3 0 2 3 3 3 3 4 4 4 4 4 5 4 3 3 3 3 2 3 3 3 4 4 4 4 4
FFT1 4 4 3 3 3 3 2 3 503 3 3 4 4 4 4 4 4 4 3 3 3 3 2 3 3 3 3 4 4 4 4 4 4 3 4 3 3 3 3 2 3 3 3 3 4 4 4 4 4 4 4 3 3 3 2 3 3 3 3 4 4 4 4 4
FFT0 5 4 3 3 3 3 2 3 552 3 3 4 4 4 4 4 4 4 3 3 3 3 2 3 4 3 3 4 4 4 4 4 4 3 4 3 3 3 3 0 2 3 2 3 3 4 4 4 4 4 5 4 3 3 3 3 2 3 2 3 3 4 4 4 3 4
FFT1 4 4 3 3 3 3 2 3 503 3 3 4 4 4 4 4 3 4 3 3 3 3 2 3 3 3 3 4 4 4 4 4 4 3 4 3 3 3 3 2 3 3 3 3 4 4 4 4 4 4 4 3 3 3 2 3 2 3 3 4 4 4 4 4
FFT0 5 4 3 3 3 3 2 3 553 3 3 4 4 4 4 4 3 4 3 3 3 3 2 3 3 3 3 4 4 4 4 4 4 3 4 3 3 3 3 0 2 3 3 3 3 4 4 4 4 4 4 4 3 3 3 2 3 2 3 3 4 4 4 4 4
FFT1 4 4 3 3 3 3 2 3 503 3 3 4 4 4 4 4 5 4 3 3 3 3 2 3 3 3 3 4 4 4 4 4 4 3 4 3 3 3 3 2 3 3 3 3 4 4 4 4 4 4 4 3 3 3 2 3 3 3 3 4 4 4 4 4
FFT0 5 4 3 3 3 3 2 3 553 3 3 4 4 4 4 4 3 4 3 3 3 3 2 3 3 3 3 4 4 4 4 4 4 3 4 3 3 3 3 0 2 3 3 3 3 4 4 4 4 4 5 4 3 3 3 3 2 3 2 3 3 4 4 4 4 4
FFT1 4 4 3 3 3 3 2 3 503 3 3 4 4 4 4 4 6 4 3 3 3 3 2 3 3 3 3 4 4 4 4 4 4 4 4 3 3 3 3 2 3 3 3 3 4 4 4 4 4 4 4 3 3 3 2 3 4 3 3 4 4 4 4 4
FFT0 6 4 3 3 3 3 2 3 553 3 3 4 4 4 4 4 3 4 3 3 3 3 2 3 3 3 3 4 4 4 4 4 4 3 4 3 3 3 3 0 2 3 3 3 3 4 4 4 4 4 4 4 3 3 3 2 3 3 3 3 4 4 4 4 4
FFT1 4 4 3 3 3 3 2 3 503 3 3 4 4 4 4 4 4 4 3 3 3 3 2 3 3 3 3 4 4 4 4 4 4 4 4 3 3 3 3 2 3 3 3 3 4 4 4 4 4 4 4 3 3 3 2 3 3 3 3 4 4 4 4 4
FFT0 6 4 3 3 3 2 2 3 553 3 3 3 4 4 4 4 4 4 4 3 3 3 3 2 3 3 3 3 4 4 4 4 4 4 4 3 3 3 0 2 3 3 3 3 4 4 4 4 4 4 4 3 3 3 2 3 4 3 3 4 4 4 4 4
FFT1 4 4 3 3 3 3 2 3 503 3 3 4 4 4 4 4 4 4 3 3 3 3 2 3 3 3 3 4 4 4 4 4 4 3 4 3 3 3 3 2 3 3 3 3 4 4 4 4 4 4 4 3 3 3 2 3 3 3 3 4 4 4 4 4
```

Figure 13 Two channel Tera Term output

To get data ready to plot, freeze the Tera Term display; disconnecting the USB cable is as good a way as any. Pick a line and copy to a new Excel spreadsheet. The data will arrive all in one cell. Go to Data, select text to columns, select delimited by space. The data is now in a single row, one value per cell. Add a line above, starting at zero and indexing by 60 (Hz) for each horizontal cell.

Plotting sine wave spectrum as a bar graph doesn't show much, and the Excel bar graph doesn't do an easy job of specifying the X-axis. A better alternative is the scatter plot, which shows the frequency on the X-axis. The vertical scale was changed to dB, 20*log(count value plus a small amount to keep excel from complaining about log of zero) for better resolution of smaller signals. In this case, the input sine wave is quite clean and the FFT is calculated with less than 0.2% error (1 count in 566).

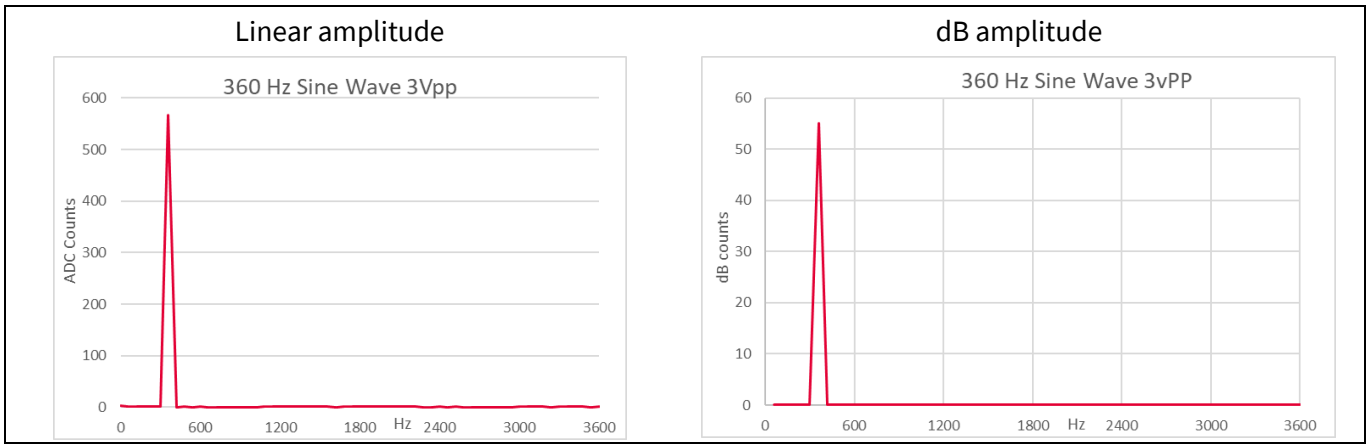


Figure 14 360 Hz sine wave FFT

The square wave harmonics fall off as 1/n, exactly as expected. This being a square wave, only the odd harmonics show up. However, the higher harmonics (above $f_{SAMPLE}/2$) show up as they are folded back in around the sample rate. This can be eliminated to some degree with an anti-aliasing filter.
This was a lot juggling just to get a plot, a real-time display would be better. Subsequent updates to this application note may include converting the data stream to a plot using Python.

C Appendix 3: Anti-aliasing

The nature of analog to digital conversion is that there is sampling of other signals, not just the desired one. The image of the desired waveform appears as waveforms around the sample frequency. Thus, signals above and below the sample rate can show up in the nominal signal band. These can, to some degree, be filtered out. A three-pole Chebyshev filter only takes a few more low-cost parts and uses the same opamp that is already buffering the input to the ADC.

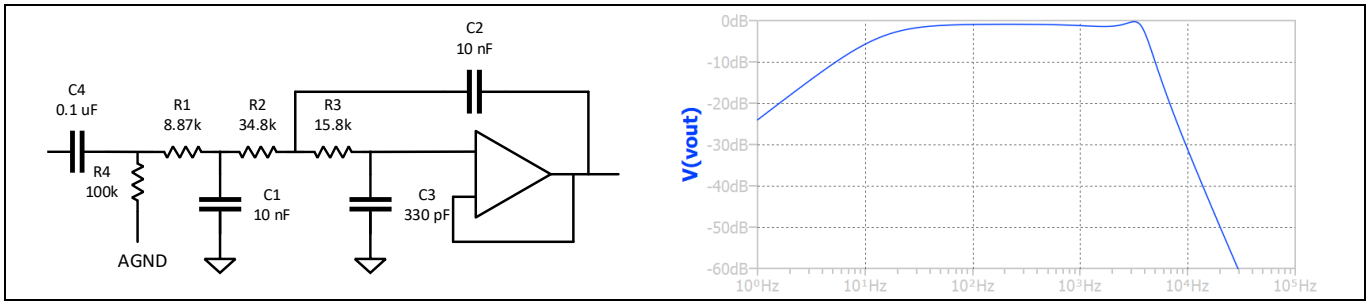
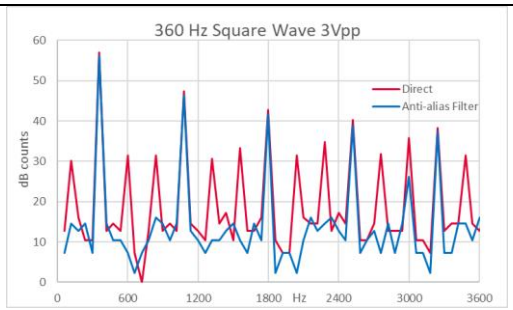
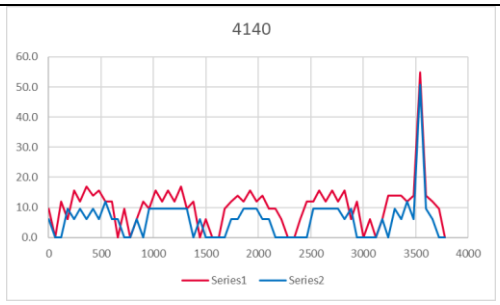
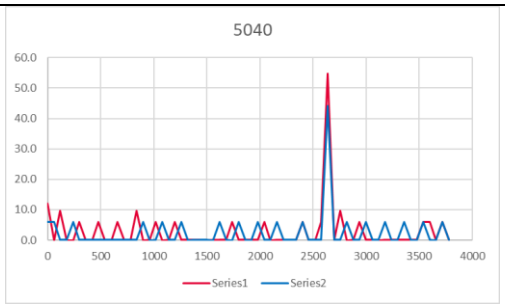
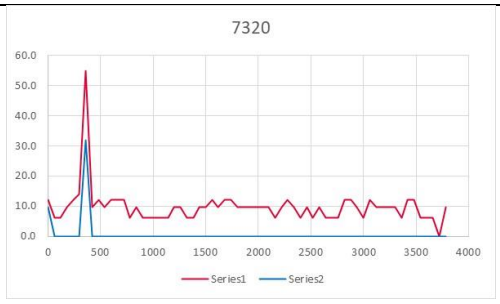
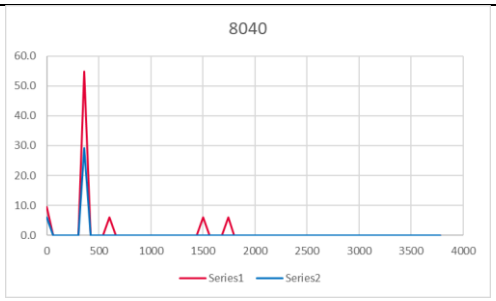


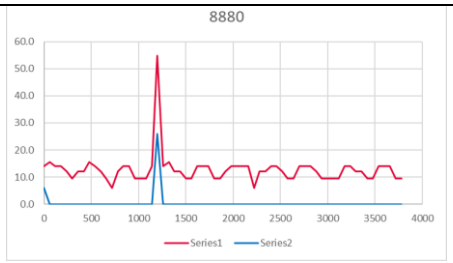
Figure 15 Anti-aliasing filter schematic and response

The level at the sample rate is reduced by ~24 dB. This is not a “brick wall,” but it is a help. A few examples show the aliases and the improvements with the filter. The same signal is fed to both channels, one with an anti-aliasing filter, one without.

Appendices

One channel (in red) shows the direct input at 360 Hz. The second channel (in Blue) show the behavior of the anti-aliasing filter. The harmonics above $f_{SAMPLE}/2$ fold back into the band of interest. The sum of these harmonics is approximately equal to 10% of the total energy. The anti-aliasing filter reduces these levels.	
A series of out-of-band signals demonstrates fold-over around the sample rate into the $f_{SAMPLE}/2$. A signal at 4140 Hz is results in an alias at 7680 – 4140 = 3540 Hz The direct channel is the same as it would be at 3540 Hz. Filter reduces the alias by 5 dB.	
A signal at 5040 Hz yields an alias at 7680 – 5040 = 2640 Hz. The filter reduces the signal at 5040 Hz folded to 2640 Hz by about 11 dB as seen in the filter response graph above. This reduces the alias by the same amount.	
A signal at 7320 Hz is very close to the sample rate, resulting in alias at 7680 – 7320 = 360 Hz. This alias is now 23 dB in the filtered channel. You can clearly see the alias “walking” down as the input gets closer to the sample frequency.	
Signals above the sample rate alias the same way as signals below the sample rate. A signal at 8040 Hz has an alias at 8050-7680 = 360 Hz. The continued roll-off of the filter results in suppression of this alias by 26 dB.	

Appendices

The final example at 8880 Hz is 1200 Hz about the filter's corner frequency. This results in an alias at 1200 Hz that is 28 dB below the level of an unfiltered signal.	

Complex digital filters do not help in this application, because the aliased noise is already in the pass band. Digital filters work only when the signals above $f_{SAMPLE}/2$ have been removed.

D Appendix 4: Expanding frequency range

The frequency range of an FFT is half of the sample rate. To double the frequency range (from 3840 Hz to 7680 Hz, it is sufficient to simply double the sample rate. A two-channel implementation (like this one or for a stereo audio application) would first set the sample rate to twice the desired frequency range (e.g., 40 ksps for 20 kHz audio). This results in FFT bins of 20 kHz/64 = 156 Hz. This is too coarse for the low frequency end of audio, so the number of samples should be at least doubled. Acquiring the data is not a problem, the ADC is far faster than required, so the sample rate can be easily increased significantly.

The software processing may not be fast enough to accomplish FFT output in the time it takes to get the data. Thus, the method would be to get the samples, put the sampling on hold, then calculate the FFT and ship the data. 128 samples are converted to FFT in 4.3 ms. The calculation time increases as n(log2(n), so 256 samples would take 4.3 ms * 2 *log2(7)/log2(6) = 9.3 ms (too much for 60 Hz update on two channels. This can be done with a few minor adjustments to the code.

Alternatively, doubling the number of samples (over two cycles) would provide 30 Hz bin resolution instead of 60 Hz. This rate is still far faster than required to meet regulatory requirements.

E Appendix 5: Quick root

Real and imaginary parts of the FFT are converted to magnitude by sqrt(real^2+imag^2). This takes an average of 1.4 ms. If time is absolutely critical and some error can be tolerated, there is a quicker way to get the root. A simple approximation to the square root is:

$$root(A,B) = MAX(A,B) + 0.41 * MIN(A,B)$$

This can be installed as a macro in a #define. It has been demonstrated to execute in 290 µs, this is a considerable saving compared to 1.4 ms for the canonical approach. Is it worth it?

The error can be +/-5% from sqrt(real^2+imag^2). If this was for an audio system, that would be +/-0.5 dB, not worth worrying about. It may, however, inflate the error on individual harmonics to several percent of scale, a problem if the arc-fault signals are small compared to the fundamental at 60 Hz.

References

[1] Liu, Y.-J.; Chen, I.-I., Fu, W.-C.; Lee, Y.-D.; Cheng, C.-C.; Chen, Y.-F.

A Hybrid Approach to Low-Voltage AC Series Arc Fault Detection, Energies 2023,16, 1256.

[2] Simon Inns, *Real-Time Audio Spectrum Analyser*, waitingforfriday.com

Revision history

Document revision	Date	Description of changes
**	2024-09-09	Initial release

Trademarks
All referenced product or service names and trademarks are the property of their respective owners.

Edition 2024-09-09

Published by

Infineon Technologies AG

81726 Munich, Germany

© 2024 Infineon Technologies AG. All Rights Reserved.

Do you have a question about this document?

Email: erratum@infineon.com

Document reference

002-40504 Rev. **

Important notice

The information contained in this application note is given as a hint for the implementation of the product only and shall in no event be regarded as a description or warranty of a certain functionality, condition or quality of the product. Before implementation of the product, the recipient of this application note must verify any function and other technical information given herein in the real application. Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind (including without limitation warranties of non-infringement of intellectual property rights of any third party) with respect to any and all information given in this application note.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer's technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

Warnings

Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.