WaveLink: a Function Generator and Oscilloscope That Learns Its Own Behaviour

PH 3032 – Embedded Systems Laboratory. The design story, the problems we met, and the solutions we built.

Author: **Kenura R. Gunarathna** I EIT Student
Project site: wavelink.krag.lk I Report site: report.wavelink.krag.lk

Abstract – This report tells the story of **WaveLink**. WaveLink is a two-channel signal generator and oscilloscope. We built it for the **PH 3032 Embedded Systems Laboratory** course. The report starts with the reason for the project. It then follows each problem we met, in the order we met it. For each problem, the report shows the solution we built. The instrument uses one **ATmega32A** microcontroller at 16 MHz. It creates each waveform with two 16-bit R-2R ladder DACs. Two 74HC595 shift registers drive each DAC. It shapes the waveform with X9C103 digital potentiometers and TL071 operational amplifiers. It can filter the waveform with a CD4051B analogue multiplexer and a Sallen-Key low-pass filter. The instrument reads its own output two ways. The first way is the ATmega32A's internal 10-bit ADC. The second way is a 16-bit successive-approximation (SAR) converter, built from the signal DAC and one LM393 comparator. A Hantek 6022BE/BL USB oscilloscope gives a third, external measurement. The report explains why the commanded settings and the real output voltage did not match, and finds the cause. Both shaping stages stop responding above wiper 30, and the design formula predicts the measured output worse than a constant does. The cause is not the amplifier saturating, as we first assumed. Each potentiometer wiper moves 3.33 times further per commanded step than its datasheet mapping implies, so it reaches its end stop after 30 of the 100 available commands. The part fitted is not the 100-position device the schematic specifies. One constant, added to the original design equation, raises the held-out R^2 from -0.50 to 0.9968 and beats a nonparametric model with no parameter limit. The report also documents the earlier records we deleted and why, and states plainly that the output is currently about three times the commanded amplitude. Last, the report shows how a React web interface uses this model to protect the hardware. The interface keeps the output frequency exact. It also holds the output voltage below the safe limit.

Contents

1 Background: Why We Built WaveLink

A student in an electronics laboratory needs two instruments on almost every bench. The student needs a **function generator**. This instrument creates a test signal. The student also needs an **oscilloscope**. This instrument looks at the result. Each instrument is useful on its own. But a laboratory task almost always needs both together. One instrument makes the signal. The other instrument checks what came out.

Two separate bench instruments have three costs. First, each instrument has its own price. A lab needs two budgets, not one. Second, each instrument takes its own space on the bench. A small workstation gets crowded. Third, a standalone bench instrument hides its own inner working from the student. This third cost matters most for this course. A student can turn its knobs. But the student cannot see the phase accumulator inside the generator. The student also cannot see the analogue-to-digital converter inside the scope.

The **PH 3032 Embedded Systems Laboratory** course asks for a different kind of instrument. The course asks for an instrument that is open, built, and understood, from the microcontroller code up to the analogue circuit. This report describes **WaveLink**. WaveLink puts a signal generator and an oscilloscope on one small board. One ATmega32A microcontroller controls the whole board. Every stage of the signal path stays open to inspection.

The rest of this report does not read like a datasheet. It follows the order in which we met each design problem. For each problem, it shows the solution we chose. Only after that does it move to results and proof.

1.1 What the Finished Instrument Does

The finished board gives the user two independent output channels. Each channel can create a sine, square, triangle, sawtooth, or custom waveform. Each channel can run at any frequency from 10 Hz to 20 kHz. The user sets the output amplitude. The user can also route the signal through a filter, for a cleaner high-frequency output. The board also measures its own output. It can send that output to a full external oscilloscope, for a second, independent check. A web page, built with React, shows live data. The web page also stops the user from setting the hardware to an unsafe state.

1.2 A Map of This Report

This report has four parts, after this background. Section 2 walks through the analogue hardware, one stage at a time. Every schematic in this report lives in that one part, next to the stage it draws. Section 3 is the turning point. It shows what happened when we tested that hardware against the simple equations that describe it, and it identifies the one constant that reconciles them. Section 4 walks through the firmware and the software, one technique at a time. This includes the fix for the problem Section 3 raises: a measured, fitted model of the real hardware. Section 5 proves how well that model works. Section 6 closes the report.

Circuit Stage	Component	Component Specifications / Value
R-2R Ladder DAC	Discrete array + 2×74HC595	$R = 10\ \text{k}\Omega$, $2R = 20\ \text{k}\Omega$ (1% tolerance), 16-bit, SPI @ 8 MHz
Voltage Reference	TL431LP	Precision shunt reference for the DAC ladder
Attenuator Stage	X9C103 Digital Pot	100 steps, $R_{\text{total}} = 10\ \text{k}\Omega$, $R_{\text{in}} = 3.3\ \text{k}\Omega$
Op-Amp Gain Stage	TL071 (JFET-input)	Active inverting stage, $R_{\text{FB}} = 10\ \text{k}\Omega$; 18 instances on the board
Sallen-Key Filter	2-stage active low-pass (TL071)	$R = 10\ \text{k}\Omega$ throughout; $C = 1\text{nF}/1\text{nF}$ and $2.7\text{nF}/470\text{pF}$
Analogue Mux	CD4051B IC	8-channel analogue switch (`x0` direct path / `x1` filtered path / `x2` scope return)
SAR Comparator	LM393 (scope) / NE5532 (buffer)	Open-collector output, 5 V pull-up, feeds pin PD2
Scope Input Divider	$900\ \text{k}\Omega/100\ \text{k}\Omega$ + 1N4148 clamps	1:10 ratio; clamps the input between -0.7 V and $+5.7$ V
Output Switching	Relays K1 to K5	Channel select, output enable, and scope-path routing

Table 1: Every component in the signal-shaping and switching chain.

2 Circuit Analysis: the Analogue Hardware

This part is a tour of the board's analogue circuit, stage by stage. Each stage gets its own short section: what problem that stage solves, how it solves it, and its own schematic. Nothing in this part is code. The firmware that drives these stages, and the software built on top of them, has its own part, Section 4.

2.1 The Signal Path at a Glance

Table 1 lists every component in the signal-shaping and switching chain the rest of this part walks through, stage by stage.

2.2 The Signal DAC: an R-2R Ladder

The ATmega32A has no built-in DAC. It only has digital output pins. Our first hardware problem was to turn a 16-bit digital sample into a real, analogue voltage, at a fresh sample every tick of the firmware's timer. Section 4 explains that timer and the sample values it sends; this section covers only the analogue ladder those samples drive.

We built this DAC from discrete resistors, in an **R-2R ladder**. It uses $R = 10\ \text{k}\Omega$ and $2R = 20\ \text{k}\Omega$ resistors, at 1 percent tolerance. Two 74HC595 shift registers clock the 16-bit sample into the ladder. They clock it over the chip's hardware SPI bus, at 8 MHz. One full sample takes about 2 microseconds to load. The ladder converts the digital code D into a voltage by Equation 1:

$$V_{\text{DAC}(D)} = V_{\text{ref}} \cdot \left(\frac{D}{65535} \right), \quad D \in [0, 65535] \tag{1}$$

But that waveform comes out of the ladder at one fixed, full-scale swing. A real function generator must give the user an adjustable output level. The rest of this part covers how we added that control, in two more analogue stages, without a second microcontroller or an expensive digital-to-analogue level shifter.

2.2.1 The Ladder's Own Resolution Limit

The ladder's resolution is bounded by **resistor tolerance**, not by the 16-bit code width. At 1 percent resistor tolerance, only about 6 to 7 bits of the 16-bit code give a clean, dependable level. So the ladder gives roughly 64 to 128 dependable amplitude steps. The extra digital depth is still useful. It lets the firmware place the signal at a fine DC offset. It also lets the firmware interpolate between the clean steps. But it cannot add new clean steps on its own.

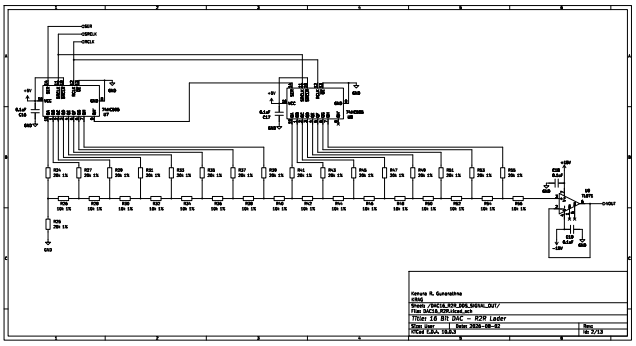

Figure 1: Schematic of the channel-1 signal DAC. It shows the 74HC595 shift-register chain and the R-2R resistor ladder. Together they turn a 16-bit sample into an analogue voltage. Board 1, sheet `DAC16_R2R_DDS_SIGNAL_OUT`.

2.3 The Attenuator Stage

The first shaping stage is a voltage divider. It uses an X9C103 digital potentiometer in place of a manual knob. This chip gives 100 discrete resistance steps under digital control. We call the chosen step the **attenuator wiper setting**, w_a. This setting can take any whole value from 0 to 99. The chip's resistance at that setting follows Equation 2:

$$R_{\text{pot}(w_a)} = 10\ \text{k}\Omega \cdot \frac{99 - w_a}{99} \tag{2}$$

This resistance forms a divider with a fixed $3.3\ \text{k}\Omega$ input resistance. The fraction of the signal that reaches the next stage is:

$$\begin{aligned} A_{\text{atten}(w_a)} &= \frac{R_{\text{in}}}{R_{\text{pot}(w_a)} + R_{\text{in}}} \\ &= \frac{3300}{10000 \cdot \frac{99 - w_a}{99} + 3300} \end{aligned} \tag{3}$$

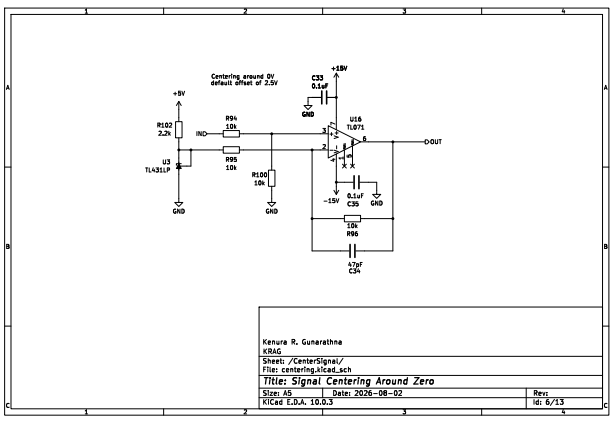

Figure 2: Schematic of the channel-1 attenuator and DC-centring stage that follows the signal DAC. Board 1, sheet `CenterSignal`.

2.4 The Gain Stage

The attenuated signal then feeds a TL071 operational amplifier. This amplifier is wired as an active inverting stage. A second X9C103 chip sets its feedback resistance, at wiper setting w_g. This feedback resistance sets the stage gain:

$$A_{\text{gain}(w_g)} = \frac{R_{\text{FB}}}{R_{\text{pot}(w_g)} + R_{\text{in}}} = \frac{10000}{10000 \cdot \frac{99 - w_g}{99} + 3300} \tag{4}$$

Put together, the two stages give the peak-to-peak output voltage in Equation 5, before either stage clips:

$$V_{\text{pp}(w_a, w_g)} = V_{\text{DAC,swing}} \cdot A_{\text{atten}(w_a)} \cdot A_{\text{gain}(w_g)} \tag{5}$$

The amplifier's supply rails allow only about 3.60 V peak-to-peak, under single-supply operation. If a commanded w_a and w_g ask for more voltage than this, the amplifier clips the top and bottom of the wave flat. Section 3 returns to this limit. It explains why Equation 5, on its own, is not accurate enough to prevent it.

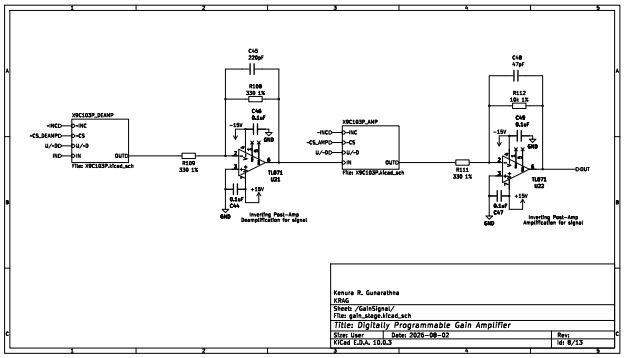

Figure 3: Schematic of the channel-1 gain stage: the TL071 inverting amplifier and its X9C103 feedback potentiometer. Board 1, sheet `GainSignal`.

2.5 The Output Multiplexer

A fixed filter in the signal path has one cost: it always removes some high-frequency bandwidth. This cost applies even when the user wants a fast, unfiltered signal. We needed the filter to be optional instead.

A CD4051B analogue multiplexer chip picks the channel-1 signal's path to the output. The firmware calls this choice the **mux source**. The firmware defines three routes. `x0`, called **direct**, sends the amplifier output straight to the output connector. This route gives a wide, flat response from 10 Hz to 20 kHz. x1, called **smooth**, sends the signal through an active low-pass filter first, described in the next section. x2 is an internal route. The firmware uses x2 only while the on-board oscilloscope is sampling.

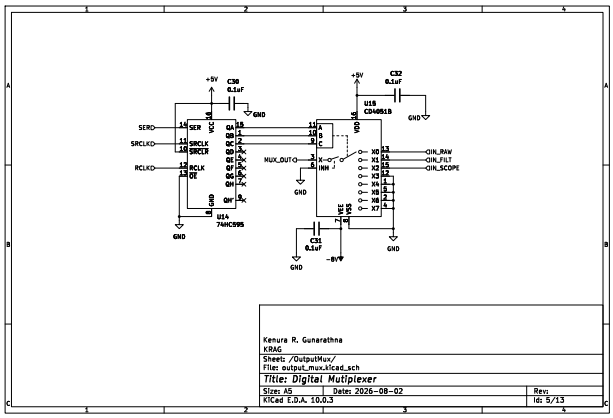

Figure 4: Schematic of the CD4051B analogue multiplexer that selects the direct or filtered output path. Board 1, sheet `OutputMux`.

2.6 The Sallen-Key Filter

The filter on the x1 path is a two-stage Sallen-Key design. Together, its two stages give a fourth-order low-pass response. It uses four $10\ \text{k}\Omega$ resistors. It also uses two capacitor pairs, $1\text{nF}/1\text{nF}$ and $2.7\text{nF}/470\text{pF}$. Take one unity-gain Sallen-Key stage with matched resistors. Its corner frequency ω_0 and quality factor Q follow Equation 6:

$$\omega_0 = \frac{1}{R\sqrt{C_1 C_2}}, \quad Q = \frac{1}{2}\sqrt{C_1/C_2} \tag{6}$$

With the schematic's component values, the equal-capacitor stage gives $f_0 = 15.92$ kHz and $Q = 0.500$. The unequal-capacitor stage gives $f_0 = 14.13$ kHz and $Q = 1.198$. Together they approximate a maximally flat, fourth-order Butterworth response with its corner near 15 kHz.

That is the prediction. Figure 11 measures it, and the fourth-order shape holds while the corner does not: the response is matched by the same form with both corners scaled by $\times\, 0.50$, putting the real corner near 7.5 kHz. A halved corner means the RC product is twice what these values imply, so either the resistors or the capacitors in this filter are about double their marked value. That is worth checking against the fitted parts before the filter's corner is quoted anywhere as 15 kHz. This filter also delays the signal's phase. One stage's phase lag dominates the x1 path's overall phase shift. That phase lag follows Equation 7:

$$\varphi_{\text{stage}(\omega)} = -\arctan\left(\frac{(\omega/(Q\omega_0))}{1-(\omega/\omega_0)^2}\right) \tag{7}$$

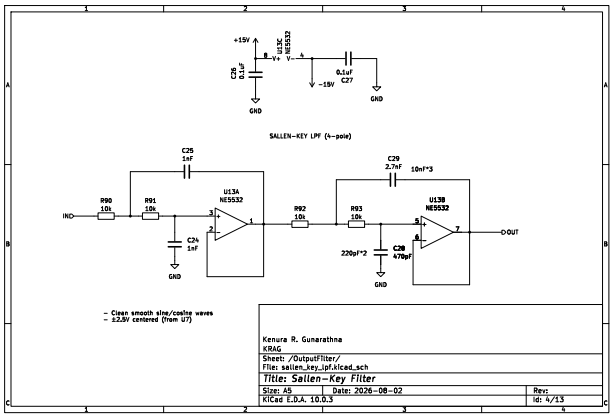

Figure 5: Schematic of the two-stage Sallen-Key low-pass filter on the filtered output path. Board 1, sheet `OutputFilter`.

2.7 Overcurrent Protection

A short circuit or a wiring mistake, at the output connector, can drive a large, damaging current through the amplifier and the output relay. We needed the board to detect this condition on its own, in hardware, before real damage occurs.

Two LM393 comparators watch the output current, one comparator per channel. Comparator U29 watches channel 1. Comparator U30 watches channel 2. Each comparator pulls its own fault line low, the moment the current crosses a set limit. Section 4 covers what the firmware does with that fault line.

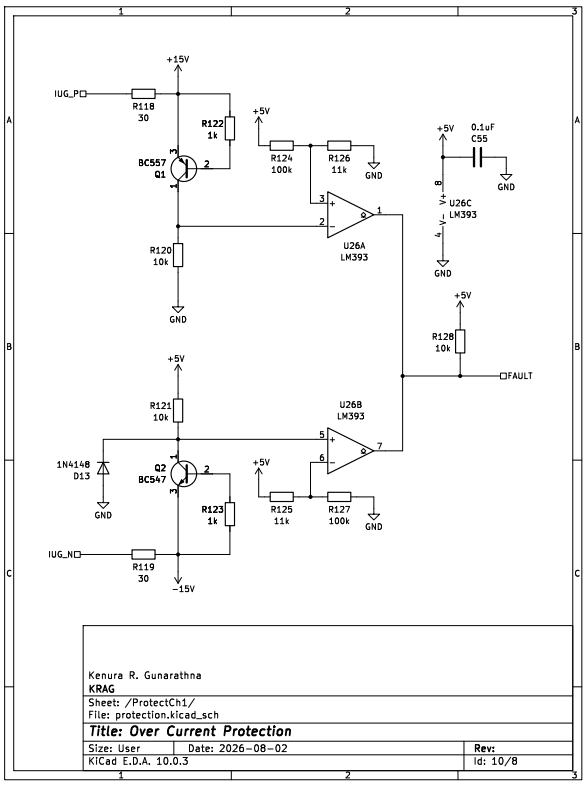

Figure 6: Schematic of the channel-1 overcurrent comparator, U29, and its fault line into the microcontroller. Board 2, sheet `ProtectCh1`.

2.8 Output Relay Switching

Five relays, K1 to K5, do the board's physical switching: channel select, output enable, and scope-path routing. Section 4 describes when the firmware opens each one. This section covers only the relay stage itself.

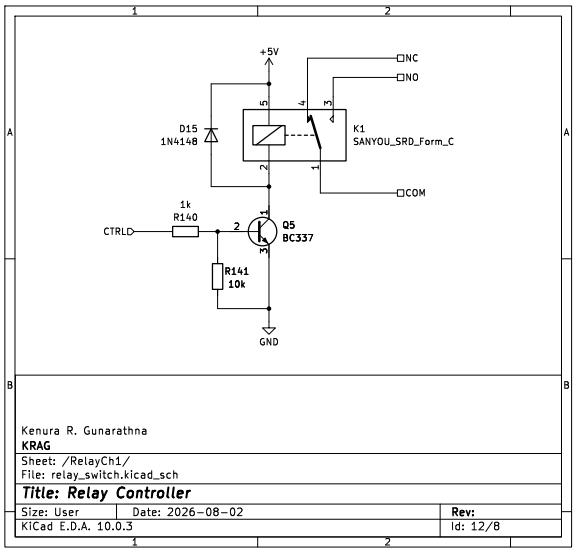

Figure 7: Schematic of one output relay stage, driven from the microcontroller and used to disconnect a channel on a fault. Board 2, sheet `RelayCh1`. Four further relay sheets on board 2, `RelayCh2` to `RelayCh5`, switch the remaining channel, mux, and scope-return paths in the same way.

3 The Deepest Problem: Why the Commanded Values Lied

3.1 What We Expected

Section 2.4 gave us Equation 5. This equation is a clean, closed-form formula for the output voltage. It is built from two wiper settings, w_a and w_g. We expected this equation to predict the real output voltage closely enough for normal laboratory use. It comes directly from the resistor values in the design.

3.2 What We Measured

It did not. We swept the wiper settings across their full range, and measured the real output on the on-board digitiser. The measured voltage did not follow Equation 5 at all: at some settings it was three times higher. Our first instinct was to blame several effects at once - an uneven potentiometer, a saturating amplifier, the filter's roll-off. That instinct was wrong, and the data says so cleanly. There is one cause.

The structure sweeps then told us exactly where the fault was, because they compare each stage against the formula at every wiper setting. Table 2 reads off four of those settings.

Wiper	Formula	Measured	Ratio
0	0.933 V	1.059 V	**1.14**
20	1.100 V	1.895 V	1.72
30	1.208 V	3.635 V	3.01
98	3.648 V	3.652 V	**1.00**

Table 2: The attenuator stage against Equation 5, at four settings.

Both ENDS agree. Only the path between them is wrong. That one observation rules out every explanation that blames the resistor network: a wrong R_{total} or a wrong R_{in} would move the endpoints too, and the endpoints are right to within 14 percent at zero and 0.1 percent at full travel. Fitting R_{total} as a free parameter confirms it, returning $10309\,\Omega$ against a nominal $10\ \text{k}\Omega$ - a 3 percent error, well inside the part's tolerance.

What fits is one number. The wiper moves **3.33 times further per commanded step** than the datasheet mapping assumes, so it arrives at its end stop at command 30 rather than 99:

$$w_{\text{effective}} = \min(99,\, k \cdot w_{\text{commanded}}), \quad k = 3.33 \tag{8}$$

A dedicated sweep at single-step resolution, eight replicates per step, puts that end stop at **wiper 30 on both stages independently**. Below it the response climbs steeply - 91 percent of the plateau at wiper 29 - and above it every remaining step holds within 0.5 percent.

Three independent tests confirm the plateau is not clipping of any kind. First, the shape stays sinusoidal. For a sine the ratio of RMS to peak-to-peak is 0.354, and flattening the tops drives it toward 0.5; measured across all 8,087 grid captures it sits at 0.339, and at the plateau itself, where clipping would show first, it is 0.336 - lower than mid-range, not higher. Second, the plateau does not move with frequency: 15.63 V below 50 Hz against 15.82 V above 2 kHz, a spread inside the measurement noise. Bandwidth or slew limiting would cut it at the top of the range. Third, slew rate is not close to being the limit. A 16 Vpp sine at the instrument's 20 kHz ceiling demands 1.0 V/us, which is 7.7 percent of the TL071's 13 V/us; slew limiting would not begin until about 259 kHz. Refitting Equation 8 inside separate frequency bands closes it, and Figure 8 plots that fit. k moves only between 3.324 and 3.350 from 10 Hz to 5 kHz, a spread of 0.8 percent, and the 10 to 50 Hz band alone - where no frequency-dependent effect can operate - returns 3.343 against the pooled 3.340. The small downward drift across the bands is larger than the fit's own standard error and so is real, but at 0.8 percent it is far too small to be the mechanism behind a factor of 3.3.

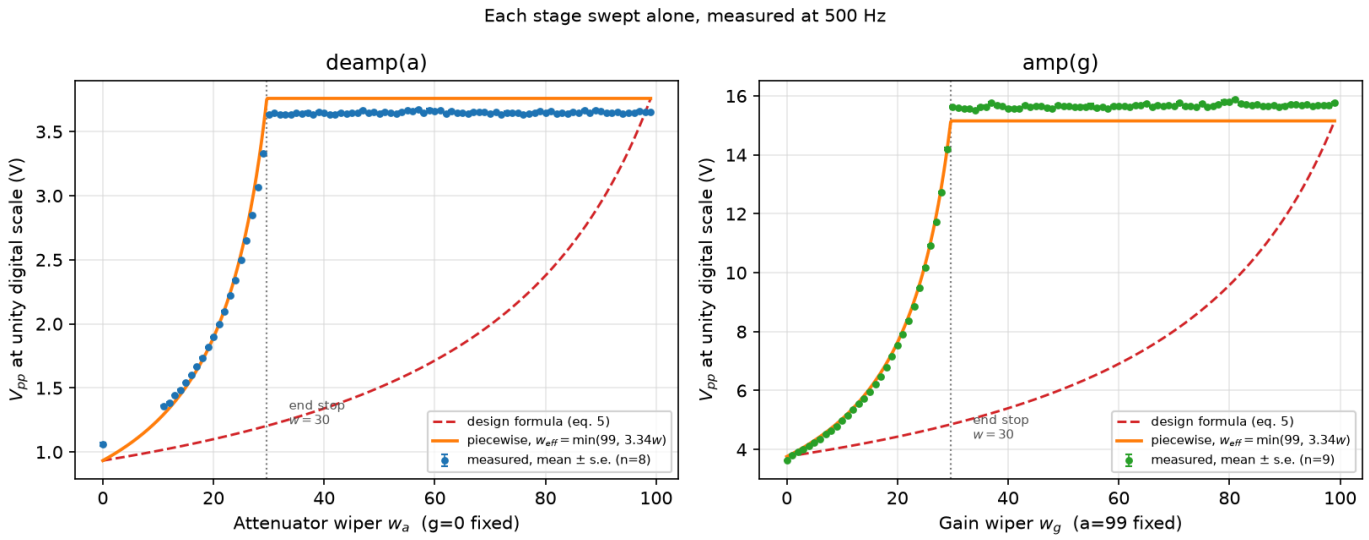

Figure 9: Each analogue stage swept on its own, at 500 Hz with the other wiper held fixed. Markers are measured means with standard-error bars, not joined - there is no measurement between two wiper settings, so no line is drawn through them. Both functions are evaluated on a dense grid instead. Dashed red: the design formula of Equation 3 and Equation 4. Solid orange: the same formula with Equation 8's single piecewise correction, drawn with its true corner rather than smoothed through it. Nothing about the resistor network was re-fitted, and the plateau offsets show that cost. Above the knee the pot is shorted out, so k has no effect there at all and the plateau height is set entirely by the component values: the model sits 3.1 percent high on the attenuator and 3.2 percent low on the gain stage. Opposite signs, so no single scale factor explains both.

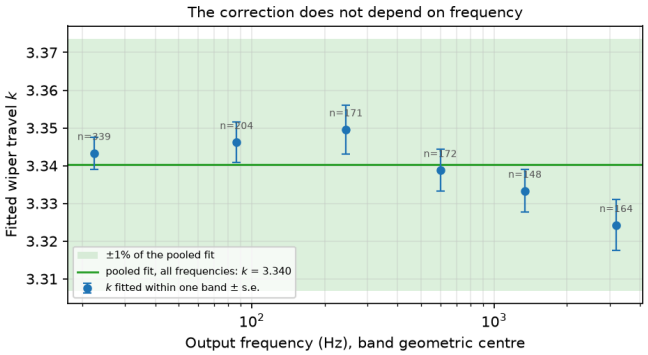

Figure 8: The wiper-travel constant refitted inside each frequency band, on that band's cells only. Error bars are the standard error from each fit's own covariance. A frequency-dependent cause - slew limiting, filter roll-off, or coarse reconstruction - would drag the fitted value down at the top of the range. It does not move outside 1 percent of the pooled fit anywhere across two and a half decades.

This also corrects what the plateau is. It is not the amplifier clipping. It is the potentiometer at the end of its travel, and its value is the formula's own full-travel value: 3.62 V measured against 3.65 V predicted for the attenuator, 15.5 V against 14.7 V for the gain stage. The circuit follows its design equation faithfully. It simply arrives at the end of that equation after 30 commands instead of 99, so 70 of the 100 settings do nothing at all.

An earlier attempt at this diagnosis reached a dead end worth recording. It held the wiper mapping fixed and solved Equation 4 for R_{in} instead, which returns -5457Ω. No resistor is negative. Fixing the wrong parameter turns a mapping error into an impossible component.

These two figures are that evidence, measured directly through the loopback path Section 4's calibration loop uses, on the on-board digitiser rather than the external Hantek dataset.

The flatness above the knee is not a measurement artefact. At every point past the knee in the attenuator sweep, the digitiser's own digital scale sat at its default, unity, value - so the raw ADC reading itself stopped rising, not just a rescaled version of it. In the gain-stage sweep, the digital scale had to shrink as the wiper rose, to keep the capture inside the ADC's window; once that scale settled, the raw reading stopped rising there too, at the same fixed scale, across every remaining wiper step. Both sweeps show the same thing by two different routes: past wiper 30, a further command buys no further output, on either stage.

The output connector confirms it independently. Commanding $w_a = 99$ and $w_g = 33$ is what the design formula asks for to produce 5.0 Vpp. On the oscilloscope, through the BNC output rather than the loopback tap, that setting measures **14.56 Vpp**, and a separate bench sheet from 2026-08-25 recorded 14.60 Vpp for the neighbouring setting $w_g = 34$. Equation 8 predicts 15.15 V, because 3.33×33 exceeds 99 and both pots are therefore at their end stops. Three measurements, two different instruments, one constant.

3.3 The Solution We Chose

We solved this by building a closed measurement loop. The loop sends a command to the board over the Bluetooth link. It waits for the output to settle. It then measures the real result, with the Hantek scope or the on-board digitiser. Last, it stores every measured point in a database. Section 4 describes the mathematical model we fitted to that data, and how we made sure every stored measurement was trustworthy before it ever reached the fit.

4 Programming Techniques: the Firmware and Software

This part covers every technique in the firmware and the calibration software, one at a time. None of it is analogue circuit design; that is Section 2. What follows is code: an algorithm running on the ATmega32A, or a script running on the host computer.

4.1 Direct Digital Synthesis: the Phase Accumulator

The ATmega32A has no built-in function generator either. It only has a fast timer. Our task was to create a smooth sine wave, a square wave, a triangle wave, and a sawtooth wave, at any commanded frequency, using only that timer and the signal DAC from Section 2.2.

We solved this with **Direct Digital Synthesis**, or **DDS**. A DDS core does not draw the waveform in real time. Instead, it looks up the next sample from a stored table. It does this once per tick of a fast internal clock. It then sends that sample to the R-2R ladder DAC.

The DDS update rate comes from a timer, `TIMER2_COMP`. This timer fires at 40 kHz when only one channel is active. It fires at 20 kHz when both channels run at once. A 32-bit **phase accumulator** tracks where the waveform is. It updates on every tick, using Equation 9:

$$\theta[n] = (\theta[n-1] + M) \bmod 2^{32} \tag{9}$$

M is the **phase tuning word**. It is a fixed number. The firmware computes it once, when the user sets a frequency. It tells the accumulator how far to step on every tick:

$$M = \frac{f_{\text{out}} \cdot 2^{32}}{f_s} \tag{10}$$

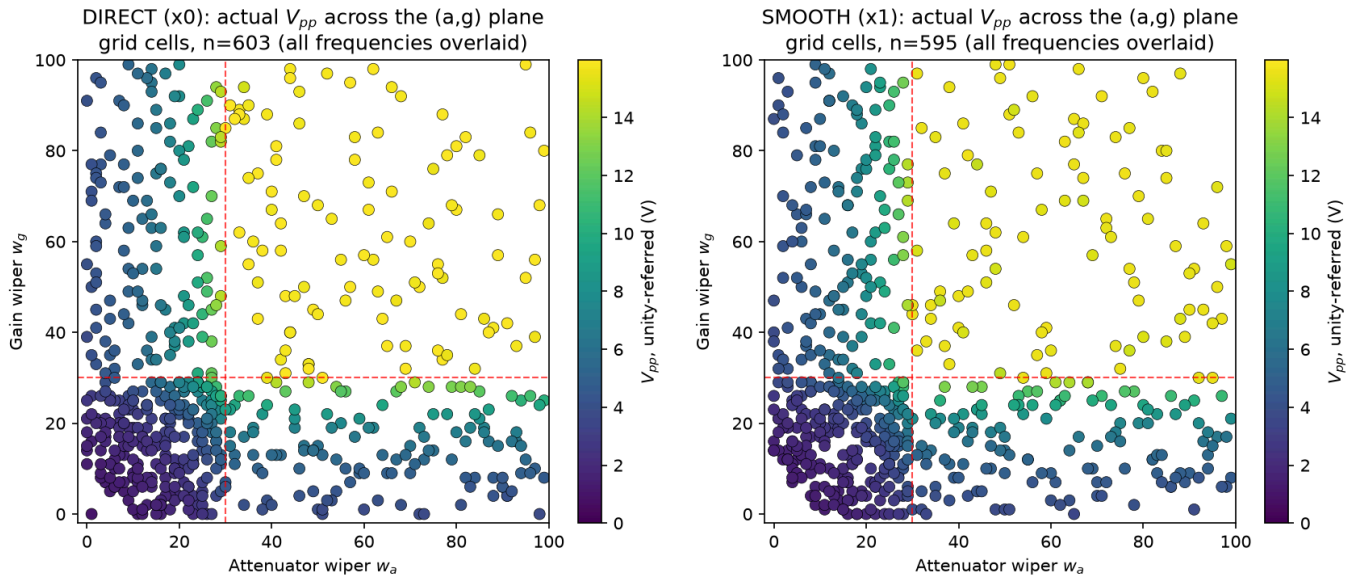

Figure 10: Actual output voltage across the full attenuator/gain plane, both paths, colour-mapped. The top-right quadrant, both wipers at 30 or above, is one uniform plateau covering 16.7% of the sampled cells - both pots parked at their end stops. That fraction is down from 36.5% because the sampler now concentrates draws below the knee, where the response still has shape. All of the instrument's real adjustability lives in the strip below 30 on each axis.

Path	Resolution	Rate	Mechanism
Internal ADC (pin PA0)	10-bit	about 38 kSPS	The ATmega32A's own on-chip ADC. It samples from a true sample-and-hold circuit
External SAR (pin PD2)	16-bit	about 6 kSPS	A binary search: the signal DAC sweeps a trial voltage, and one LM393 comparator reports if the input is above or below it

Table 3: The two on-board measurement algorithms, and the trade-off each one makes.

Path / Wire Format	Sample rate	Usable signal
ADC 10-bit, binary (`ob1`)	**5,359 samples/s**	about 536 Hz
SAR 8-bit tier, binary (`ob1`)	3,551 samples/s	about 355 Hz
ADC 10-bit, text (`ob0`)	1,388 samples/s	about 139 Hz
SAR 16-bit, text (`ob0`)	1,098 samples/s	about 110 Hz

Table 4: Measured data-transfer rate for each digitiser and wire format.

Here f_{out} is the frequency the user asked for. f_s is the tick rate above. The firmware function `WaveGenerator::setFrequency()` computes M as one 64-bit value: `(f * 4294967296) / sampleRate`. The multiply cannot overflow at that width.

This gives a frequency resolution far finer than the ladder's own amplitude resolution (Section 2.2):

$$\Delta f = \frac{f_s}{2^{32}} = \frac{40\ \text{kHz}}{2^{32}} \approx 9.31 \times 10^{-6}\ \text{Hz} \qquad (11)$$

This step is about 9.3 microhertz. Every whole-number frequency from 10 Hz to 20 kHz sits exactly on the digital grid. No rounding happens at all.

4.1.1 Storing Half a Wave, Not the Whole Wave

A full sine table would use twice the memory a half sine table needs. The second half of a sine wave is only a mirror image of the first half. So we store only the first half of the wave. This half runs from 0 degrees to 180 degrees. The table has 4096 entries. Each entry is 16 bits wide. The table lives in flash memory. It costs 8 kB of the chip's 32 kB.

The firmware builds the second half of the wave by reflecting the first half. It reflects about the DAC mid-code, $0x8000$. This gives 8192 real steps across one full cycle. It needs only one table, half the full size. The top 13 bits of the phase accumulator pick a row in the table. One more bit of that address tells the firmware which half of the wave it is in. The firmware reads this bit as a plus-or-minus sign. It does not read this bit as a branch in the code. So every tick of the timer costs the same fixed 15 CPU cycles. This cost is the same for both halves of the wave. At 40 kHz, this DDS core uses about 3.75 percent of the chip's processing time. It never causes timing jitter.

4.2 Reading Our Own Output: Two Digitiser Algorithms

We wanted the board to check its own output. It could not depend only on an external oscilloscope. It also could not use an expensive, dedicated ADC chip.

We built two separate digitiser algorithms from parts already on the board. Each one trades resolution for speed in its own way, as Table 3 shows.

The 16-bit path has no dedicated ADC chip at all. The firmware builds it from parts meant for signal generation instead. It runs a 16-step binary search. On each step, it writes a trial code to the signal R-2R DAC. It then reads the LM393 comparator's output, on pin PD2, to decide the next bit.

4.2.1 A Real Limit We State Plainly

This 16-bit search has one serious weakness: it has no sample-and-hold circuit. Its 16 steps take about 216 microseconds in total. Every step in that search must see the same input voltage. If it does not, the result is wrong. We need the input to stay stable for the whole 216 microseconds. It must stay within about one small step of the search, roughly 19.5 millivolts. This bounds the useful input frequency to about 90 Hz. The internal 10-bit ADC has a true sample-and-hold circuit. It does not share this limit.

Table 4 shows how fast each digitiser can send data over the Bluetooth link, in each output format. It also shows the highest signal frequency each format can still show cleanly. This limit assumes 10 samples per cycle as the target.

4.2.2 A Third, Independent Check

Both on-board algorithms still leave one open question. Each one checks the signal against the board's own reference. Neither can

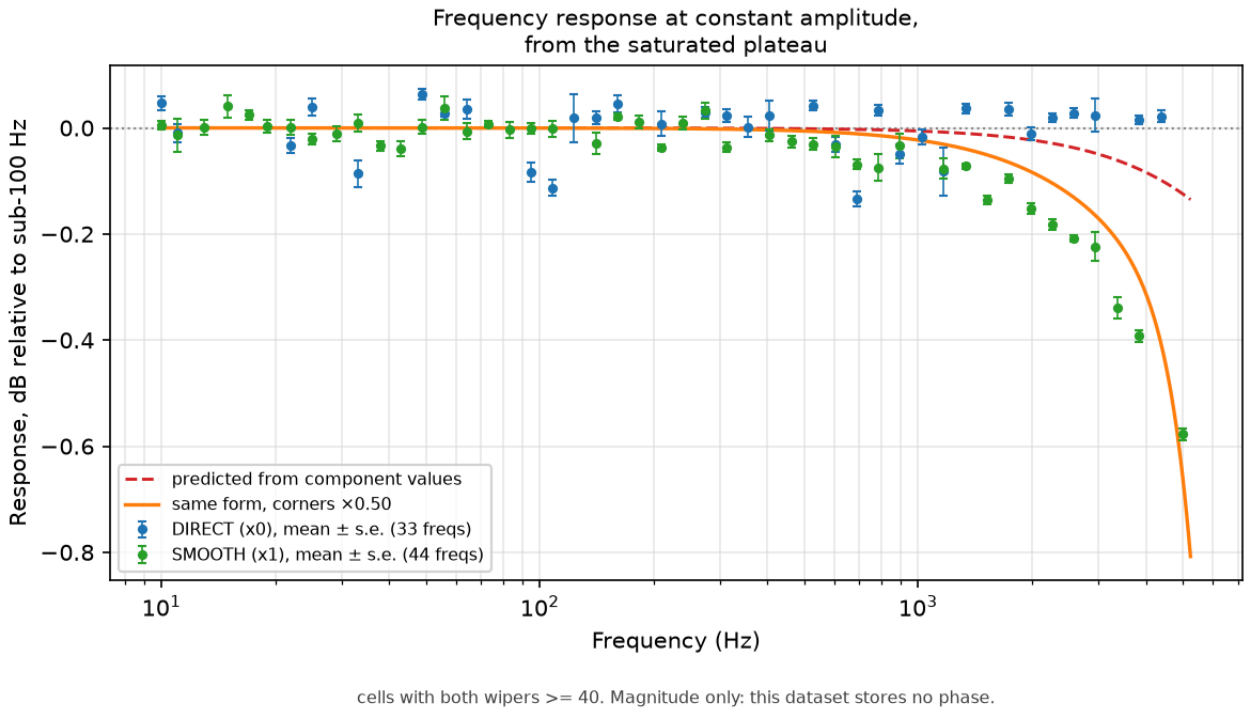

Figure 11: Frequency response of each path, measured from the saturated plateau at constant amplitude. Markers are means with standard-error bars, unjoined. The direct path is flat to within ± 0.1 dB, ending $+0.02$ dB from where it started. The filtered path rolls off by -0.32 dB by 2 kHz. Dashed red is NOT a fit: it is the cascade of the two Sallen-Key sections computed from the schematic's own component values, so the gap between it and the data is a real test of the filter design. Solid orange is the same fourth-order form with both corners scaled by one fitted factor, which lands at $\times 0.50$. Magnitude only.

catch an error in that reference. For a fully independent check, the board can also route its output to a Hantek 6022BE/BL USB oscilloscope, driven by a small program on the host. A calibration run can trigger a real, external capture at any time. Section 3 explains why we needed this third, independent measurement.

4.2.3 The Host Link

The board talks to the host computer over a transparent Bluetooth Low Energy link. The link uses an HM-10-class module at 115200 baud. It carries a short, plain-text command protocol. This same link carries every measurement command the calibration loop below sends and receives.

4.3 Detecting and Handling a Fault in Firmware

Section 2.7 described the two comparators that sense an overcurrent fault in hardware. This section covers what the firmware does the instant one fires.

The microcontroller makes the final decision, not a separate protection chip. Two hardware interrupt pins watch the two fault lines. INT1, on pin PD3, watches channel 1. INT2, on pin PB2, watches channel 2. Both pins are armed to fire on a falling edge. The moment either fault line falls, the matching interrupt routine runs at once. It opens that channel's output relay, K2 for channel 1 or K3 for channel 2. This action disconnects the amplifier from the output connector.

This design also leaves the microcontroller's own memory tight. Static RAM use stands at 1986 of the chip's 2048 bytes. That is 97.0 percent full. Only about 62 bytes remain, as headroom for the call stack.

4.4 The Calibration Loop and the Wiener Model

Section 3 showed three effects at once. One was a frequency-dependent amplitude shift. Another was a frequency-dependent phase shift. The third was a wiper-dependent saturation. Fitting one single equation to all three effects would need a very large, hard-to-trust model. Instead, we split the real hardware into two simpler blocks. Control engineers connect these two blocks in series, in a structure they call a **Wiener model**:

$$y(t) = g(\mathcal{L}^{-}1\{H(s) \cdot X(s)\}) \tag{12}$$

The first block, $H(s)$, is **linear**. It only changes amplitude and phase with frequency. It does not depend on the wiper settings at all. The second block, g, is **static** and **non-linear**. It only depends on the wiper settings and the instantaneous signal level. It does not depend on frequency at all. Splitting the problem this way let us fit each block from the part of the data where it matters most.

4.4.1 The Linear Block: $H(s)$

We model the frequency-dependent part of the hardware as a standard second-order transfer function. This part is mostly the Sallen-Key filter and the amplifier's own roll-off:

$$H(s) = \frac{b_0}{s^2 + a_1 s + a_0} \tag{13}$$

Measuring this block needs a constant amplitude at every frequency. Our data gives one for free. Above wiper 30 both stages sit against a fixed clipping ceiling (Section 3), so that ceiling IS a constant-amplitude reference. Its measured value against frequency is therefore a real frequency response, with the wiper dependence removed by construction rather than by fitting.

4.4.2 The Measurement Ceiling, Measured Directly

Every figure so far divides the digital scale back out. That division is exact, but it is still an extrapolation: it reports what a setting WOULD produce at unity scale, from a capture taken at a smaller one. One set of sweeps avoids that entirely by running at unity scale with the auto-ranging disabled, so it answers the question directly.

The two panels say different things. The attenuator alone cannot drive the digitiser into its ceiling anywhere in its range, which is why Section 4.5′s gates almost never fire on an attenuator-only sweep. On the gain stage the digitiser hits that ceiling at wiper 9, so this sweep records only ten settings and stops; 4.65 V, at wiper 8, is the last clean reading. It is worth being exact about what that number bounds. It is the largest amplitude this **measurement path** can carry, not the largest the gain stage can produce - Figure 9 takes the same stage to 15.5 V using the digital scale to stay inside the ADC's window.

Read that 4.65 V for what it is. The loopback tap is R21 straight into the D11/D12 clamps at 0 and +5 V, with no attenuator, and the internal ADC's window is 1024 counts wide. So this is the DIGITISER reaching its ceiling, not the TL071 reaching its supply rails. It bounds what the loopback path can characterise, which is precisely why `autorange_y()` exists, and it is not a safe-output figure for the output connector.

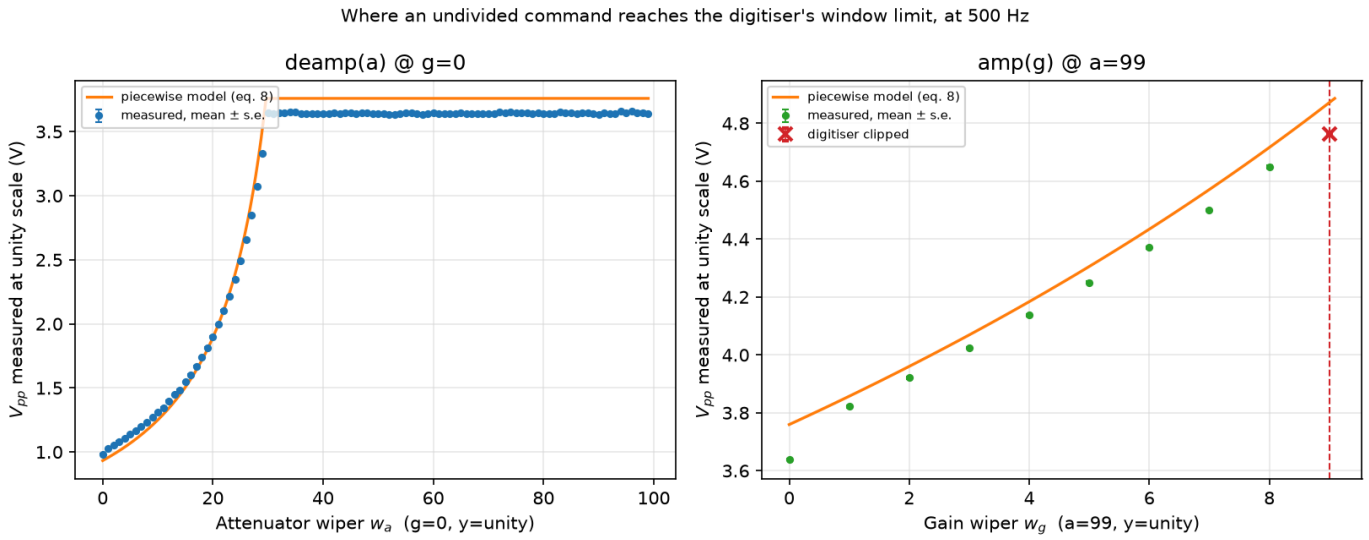

*Figure 12: Both stages swept at unity digital scale with auto-ranging off, at 500 Hz. Markers are means with standard-error bars, unjoined; the orange curve is Equation 8 drawn as a function. Left: the attenuator alone never reaches the digitiser's window at any of its 100 settings. Right: with the attenuator wide open, the **digitiser** runs out of window at wiper 9 (red cross), so the sweep stops there and the last clean reading is 4.65 V at wiper 8. This is the loopback tap's limit, not the amplifier's: the tap feeds the internal ADC through no attenuator, and that stage saturates long before the gain stage does. Eight replicates per point.*

Gate	Test	Failure it catches
Frequency	$\lvert f_{\text{meas}} - f_{\text{cmd}} \rvert \le \max(5\ \text{Hz}, 0.25 f_{\text{cmd}})$	The capture does not contain the commanded signal
Duration	The window spans 0.5 to 30 cycles of f_{cmd}	The timebase does not fit this signal
Stationarity	V_{pp} ratio between quarters of the capture ≤ 2.0	Two different signals were spliced into one capture
Range	A flat run at the rail lasts at most 10 percent of one cycle, and the capture shows at least 20 distinct codes	The signal is clipped, or it is too small for the ADC range

Table 5: Every capture must pass all four gates before the sweep logs it.

4.4.3 What We Cannot Report: Phase

An honest report has to say what its data does not contain. Our loopback records store no phase. The on-board digitiser returns five reduced numbers per capture, listed in Table 3′s mechanism column, and none of them is a phase. So we cannot report a phase error, and we do not fit the phase term Equation 13 would allow. Measuring phase needs the external instrument path of Section 4.2, and a dataset from it that meets the standard in Section 4.5.

4.4.4 The Static Block: Not a Saturation At All

The Wiener structure expects the static block to be a saturation, and the usual choice is a hyperbolic tangent:

$$g(x) = V_{\text{sat}} \tanh\left(\frac{Gx}{V_{\text{sat}}}\right) \tag{14}$$

Our measurements reject that shape, and Section 3 explains why: there is no saturation in this circuit to model. A hyperbolic tangent bends gradually into a limit set by the amplifier's supply rails. What the hardware does instead is run out of potentiometer travel at command 30, and then hold the design formula's own full-travel value exactly. The amplifier is never in clipping.

So the static block is not a fitted non-linearity. It is Equation 5 with Equation 8′s one correction, and the non-linearity is the $\min$ that clamps the wiper at its end stop. Section 5 scores this against the tanh and against a fitted saturation, on data none of them saw.

4.5 Making Sure the Measurements Were Trustworthy

A model is only as good as the data used to fit it. An automated measurement sweep runs for hours. No person watches every reading. So the sweep will collect some bad readings. A reading can be a clipped waveform. It can be a dropped sample. It can be a capture taken while the signal was still settling. If we fitted the Wiener model to this data without a check, a small number of bad readings could quietly bend the whole fit.

We solved this two ways. First, every captured frame passes through four independent checks. Table 5 lists these checks. The sweep only allows a frame into the database if it passes every check. If a frame fails one check, the sweep discards it and tries again.

Second, we ran a large number of sweeps and let noise average out. Each automated sweep measures a point in the parameter space of frequency, w_a, and w_g. Every such point falls into one of three zones. In the **clean linear zone**, a capture passes every gate with certainty. In the **hard rail saturation zone**, a capture fails every gate with certainty. It fails because the signal is fully clipped. In the **boundary fringe zone**, between the two, a capture passes or fails with roughly even chance. This happens because random noise sits right on the gate's edge. Across this fringe zone, repeated measurement lowers the noise on the average result. This follows the standard rule $\sigma_{\text{ensemble}} = \sigma_{\text{noise}}/\sqrt{N}$. We stored every gate-passed measurement in a DuckDB database. The valid data is 10,766 records over 1,397 distinct cells, from two campaigns: an initial run and a longer one that swept each wiper at single-step resolution with eight re-homed replicates per point. Averaging replicates brings the noise on a cell mean down to **0.217 percent**. That figure is the floor every accuracy number in Section 5 is measured against.

4.5.1 Why We Deleted Our Earlier Data

Two gates cannot catch a fault that is invisible in the row itself, and we hit exactly that. Our earlier external sweeps did not reset the digital scale y when they connected. A sweep that started after an interrupted calibration run inherited that run's scale. One such sweep ran at `0x5200`, which is 64 percent of unity, and logged every amplitude 36 percent low. Nothing in the stored row recorded the scale, so nothing downstream could correct it.

We proved this rather than assumed it. We compared each external run against the loopback path on shared settings. The ratio should be one constant. Instead it clustered by run, from 0.084 to 0.94. The run we knew had inherited `0x5200` came out at 0.63 of the corrected runs, against 0.6406 predicted by $0\text{x}5200/0\text{x}8000$. A second, separate fault put 32 rows at a probe attenuation of 1.0 while 14,189 sat at 10.0, giving exact factor-of-ten disagreements on repeated cells. Neither fault is repairable after the fact, because the missing quantity was never stored. So we deleted every affected record and re-measured. Every figure and every number in this report comes from the re-measured data. `cli_sweep.py` now forces y to unity when it

Model	Fitted params	R^2	RMS error	Median error
Theoretical formula (Equation 5), no fit	0	-0.50	5914 mV	3257 mV
tanh saturation on the formula	2	0.564	3187 mV	1936 mV
Hard clip on the formula	2	0.544	3257 mV	1541 mV
Wiper travel (Equation 8), $k = 3.34$	**1**	**0.9968**	**273 mV**	101 mV
Gaussian process on $(w_a, w_g, \log f, \text{mux})$	non-par.	0.9918	436 mV	**52 mV**

Table 7: Held-out accuracy on 699 unseen cells. The measurement noise floor on a cell mean is 0.217 percent, so roughly 8 to 34 mV across this amplitude range.

connects, and verifies the board reports it back, so this class of fault cannot recur silently.

4.5.2 Recovering an Accurate Phase Reading

Phase is easy to measure badly. It is hard to measure well. We recovered the real phase with **quadrature projection**. First, we removed the DC level from the captured record. Next, we projected that record onto a sine wave and a cosine wave, at the measured frequency. Last, we took the phase angle as $\arctan_2(Q, I)$, from the two projected values. On a clean capture, this method holds the phase error below 0.01 degrees.

4.6 Keeping the User Safe: the Priority Protection Engine

A model this accurate is only useful in one case. It must reach the person using the instrument, at the moment they set a frequency and an amplitude. It must act before an unsafe command ever reaches the hardware.

We built a web-based interface, in React and TypeScript. Inside it sits a small piece of logic we call the **Priority Protection Engine** (in the source file `dds.ts`). This engine applies two fixed rules, in a fixed order, to every command the user makes.

Priority 1: keep the frequency exact. The engine never changes the frequency the user asked for. A test signal is only useful if its frequency is the one the user set.

Priority 2: keep the voltage safe. The engine computes the highest safe peak-to-peak voltage for the user's chosen frequency and wiper settings, using Equation 5. If the requested amplitude exceeds that limit, the engine lowers the command to the highest safe value and tells the user it has done so.

That is the design. It is not yet the behaviour, and the gap matters more than the design does. The engine computes its limit from Equation 5, and Section 3 shows that equation understating the real output by up to three times. So a request the engine believes is 5 Vpp reaches the connector at about 14.6 V, measured. The engine will protect the user once Equation 8′s correction is carried into `dds.ts`; until then its clamp is arithmetic against the wrong number, and Section 5 states the working limit plainly.

5 Proof: How Well the Model Predicts the Real Output

5.1 The Fitted Model in Full

Every piece of the adopted model has appeared already, in the section that derived it. Collected in one place, and in the order it is evaluated:

$$
\begin{aligned}
R_{\text{pot}(w)} &= R_{\text{total}} \cdot \frac{99 - w}{99} \\
w_{\text{eff}(w)} &= \min(99, k \cdot w) \\
A_{\text{atten}(w_a)} &= \frac{R_{\text{in}}}{R_{\text{pot}\left(w_{\text{eff}(w_a)}\right)} + R_{\text{in}}} \\
A_{\text{gain}(w_g)} &= \frac{R_{\text{FB}}}{R_{\text{pot}\left(w_{\text{eff}(w_g)}\right)} + R_{\text{in}}} \\
\hat{V}_{\text{pp}(w_a, w_g)} &= V_{\text{DAC,swing}} \cdot A_{\text{atten}(w_a)} \cdot A_{\text{gain}(w_g)}
\end{aligned}
\tag{15}
$$

Only k is fitted. Everything else is a component value read off the schematic, carried through unchanged from Equation 3 and Equation 4.

Symbol	Meaning	Value
w_a, w_g	Commanded wiper position, attenuator and gain stage	$0..99$
w_{eff}	Position the wiper actually reaches	$0..99$
k	Wiper travel per commanded step, relative to the datasheet mapping. The one fitted parameter	3.34
R_{total}	End-to-end resistance of the digital potentiometer	$10\ \text{k}\Omega$
R_{pot}	Resistance in circuit at a given wiper position	derived
R_{in}	Fixed input resistor of each stage	$3.3\ \text{k}\Omega$
R_{FB}	Feedback resistor of the gain stage	$10\ \text{k}\Omega$
A_{atten}	Voltage ratio of the attenuator stage	$0.248..1.0$
A_{gain}	Voltage gain of the amplifier stage	$0.75..3.03$
$V_{\text{DAC,swing}}$	Peak-to-peak swing out of the R-2R ladder	$5.0\ \text{V}$
$\hat{V}_{\text{pp}}$	Predicted output, peak to peak	$0.93..15.2\ \text{V}$

Table 6: Every symbol in Equation 15. k is refit per multiplexer path when the model is deployed; the two paths agree to within 0.1 percent.

5.2 Held-Out Accuracy

We tested each candidate model the honest way. Our valid data is 10,766 records over 1,397 distinct cells, from two campaigns. We split those cells in half by a seeded shuffle, fitted on one half, and scored the predictions against the other, unseen half. Table 7 reports that score. Every row is reproducible by running `tools/calibration/make_report_charts.py`, which also draws every figure in this report.

Four results matter here, and the first one is a warning.

The unfitted formula scores below zero. An R^2 of -0.50 means the formula predicts the measured voltage worse than simply guessing the average of every measurement. It is not a small calibration error to be tuned away: Figure 9 shows the same failure stage by stage.

Fitting a saturation to it does not rescue it. A hyperbolic tangent, with two fitted parameters, lifts R^2 only to 0.564. A hard clip reaches 0.544. Both assume the circuit saturates, and it does not, so no amount of fitting repairs them. This is the cost of a wrong form, measured.

One physical parameter fixes it. Equation 8 adds no new physics and re-fits no resistor. It corrects where the wiper actually sits, with one constant, and reaches $R^2 = 0.9968$ - a median error of 101 mV against the formula's 3257 mV, a factor of 32. That is the whole repair.

The flexible model does no better. The Gaussian process has an effectively unbounded parameter count and lands at $R^2 = 0.9918$, WORSE than the one-parameter physical correction on both R^2 and RMS error. It wins only on median error, by fitting local wiggles the analytic law does not describe. A nonparametric surface beaten by a single constant is the clearest possible sign that the constant is the real mechanism, not a curve-fitting convenience.

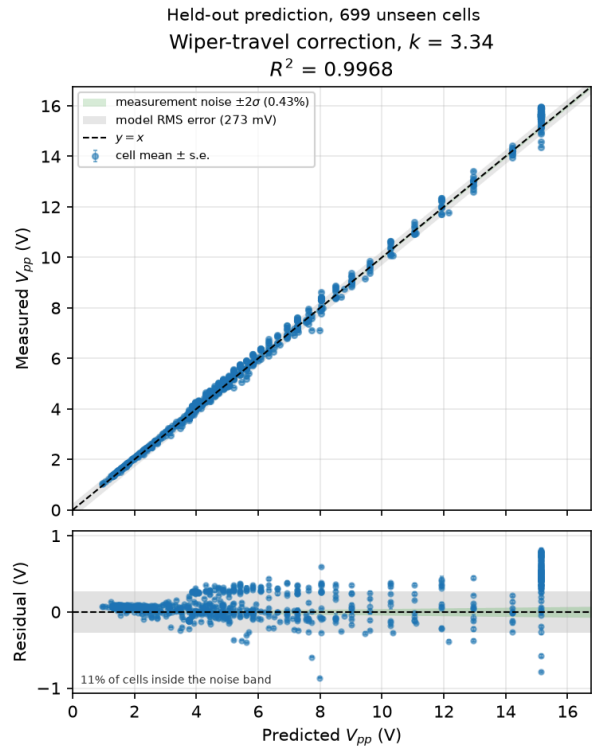

Figure 13: Held-out prediction against measurement for the correction this report adopts, on cells the fit never saw. One fitted parameter. Error bars are the standard error of each cell's mean over its replicates, and are smaller than the marker almost everywhere. The green band is what the measurement itself can resolve ($\pm 2\sigma$, 0.43 percent, proportional to amplitude); the grey band is this model's own held-out RMS error. The lower panel subtracts the diagonal so both bands are legible. Only 11 percent of cells fall inside the noise band, so what remains is model error, not measurement error. The spread is widest at the top of the range, where both wipers sit at their end stops and the output is set by the plateau rather than by the command - see Table 7 for the alternatives.

5.3 What This Does Not Yet Prove

A held-out R^2 of 0.9968 from one constant is a strong result, and it is worth being precise about what it does not settle.

The model carries no frequency term, and that is a choice the data supports rather than an oversight. Equation 15 maps wiper settings to amplitude alone. It is fitted across the full 10 Hz to 5 kHz span, so any frequency dependence shows up as residual. Measured on the saturated cells, where amplitude is fixed by the end stops and only frequency varies, the direct path moves -0.01 dB from below 100 Hz to above 2 kHz and the filtered path moves -0.28 dB. The direct path is therefore flat to well inside the noise floor. The filtered path's 3 percent is real, is the Sallen-Key roll-off of Figure 11, and is the same order as the model's own median error - so a frequency term would buy something on that path, and nothing on the other. The structure sweeps of Figure 9 and Figure 12 were taken at a single frequency, 500 Hz, and say nothing about frequency on their own.

The residual is still model error, not measurement error, and only 11 percent of held-out cells land inside the measurement noise band of Figure 13. What remains is now identified rather than merely acknowledged. It is the plateau. Above the knee the potentiometer is shorted out, so k has no influence on the plateau height whatsoever - that height is fixed by the component values alone. Measured against them, the model is 3.1 percent high on the attenuator plateau and 3.2 percent low on the gain plateau. The two errors have opposite signs, so no single scale factor removes both, and Figure 13′s residual panel shows the consequence as a band of positive residuals at the top of the range where every cell sits on that plateau.

What would close it is known, and the order matters. Adding parameters one at a time and scoring each on the same held-out half:

Model	Params	R^2	RMS error	Gain
Wiper travel k (shipped)	1	0.99680	273.0 mV	–
+ full-scale factor	2	0.99872	172.8 mV	-100 mV
+ per-path scale	4	0.99891	159.5 mV	-13 mV
+ filtered-path roll-off	5	0.99913	**142.6 mV**	-17 mV

Table 8: Each addition scored on the same 699 unseen cells. The single full-scale factor is worth more than everything after it combined.

The full-scale factor dominates because the error it removes is an offset, not a slope: before it, both paths read 2 to 4 percent high at **every** frequency. The DAC swing and the gain stage's feedback resistor are algebraically one parameter here, since both simply multiply the result, so this is the chain's true full-scale value rather than any single component.

Frequency only becomes worth modelling after that offset is gone. Fitted first, a per-path roll-off takes 273 mV to 267 mV - six millivolts, because it is fitting on top of an offset it cannot remove. Fitted after, it is worth 17 mV. It also belongs to one path only: with the scale corrected, the direct path drifts -0.41 percentage points from below 100 Hz to above 2 kHz, inside the noise, while the filtered path drifts -4.54, which is the Sallen-Key doing what it was designed to do. Allowing the direct path a free-order roll-off returns an order of 10.6, which is not a filter, it is noise being fitted; the roll-off above is therefore fixed at second order and applied to the filtered path alone.

We did not ship this. One parameter carrying a factor of 3.3 is a result about the hardware. A full-scale factor trimming a further 100 mV is calibration, and it is fitted against this instrument's own digitiser - the same converter whose reference it would be correcting. It needs one external check against a traceable meter before it can be trusted as an absolute, and that check is a few readings, not another campaign.

The cause is now identified, and it is not the firmware. We considered two candidates. Either `X9c::pulseInc()`, which holds the increment line low for 1 microsecond - exactly the X9C103 datasheet minimum - was letting a ringing edge be counted several times per commanded pulse; or the fitted part has fewer usable taps than the 100 the driver assumes.

The measured k settles it. Pulse multiplication has to be an INTEGER, because an edge is either counted or it is not. The single-step sweep gives $k = 3.33$, which is not an integer, and it places the end stop at command 30 in both stages. Re-parameterising Equation 8 as a step count rather than a scale factor - the two are algebraically the same, since $k/99 = 1/N$ - returns $N \approx 30$ usable steps and fits identically.

So the potentiometer traverses its whole range in about 30 steps, not 99. The firmware is doing exactly what it claims: `an<pos>` calls `X9c::setNoSave()`, which issues one pulse per step with no multiplier anywhere in the path. The part fitted to the board simply is not the 100-position X9C103 the schematic specifies. Roughly 30 steps is consistent with a 32-position device, which is a common part and a common substitution in low-cost modules.

This is a hardware substitution, not a software defect, so the correction belongs permanently in the model rather than being fixed by a firmware change. It also means the instrument has 30 usable amplitude settings per stage, not 100 - a resolution limit worth stating in any specification of this board.

Finally, this model predicts the loopback plane, which is the onboard digitiser's own tap. The two output-connector readings in Section 3 agree with it to within 4 percent, which is reassuring but is not a calibration. Referring the model properly to the output needs paired measurements from the external instrument path, and the purge in Section 4.5 leaves us without them.

Until the cause is known, the instrument should be treated as unsafe at commanded amplitudes above about 1.5 Vpp. The protection engine of Section 4.6 computes its safe limit from Equation 5, which Table 7 shows scoring below zero, so it is not currently protecting anything: a command for 5 Vpp puts roughly 14.6 V on the connector.

6 Deployment and Conclusion

The project site and this report are both static web pages. Cloudflare Pages hosts both of them. You can reach them at wavelink.krag.lk and report.wavelink.krag.lk.

This report followed the **WaveLink** project from its starting need. That need was one open instrument to replace two closed ones. It then walked through the analogue hardware, stage by stage, in Section 2. It showed, in Section 3, why that hardware's own equations could not be trusted on their own: the two shaping stages hit a hard ceiling near wiper 30, and the design formula predicts the measured output worse than a constant does.

Section 4 then walked through the firmware and the software, technique by technique. The repair turned out to be one constant rather than a model: each wiper travels 3.33 times further per commanded step than assumed, and correcting only that raises the held-out R^2 from -0.50 to 0.9968. The design equation was right all along. What was wrong was our belief about where the wiper sits when we ask for a position - and the cause is a substituted part, not our code.

Two things are not settled, and they are the next work. The model predicts the on-board loopback plane, and referring it to the output connector needs paired external measurements to replace the records we deleted. And until the correction reaches `dds.ts`, the protection engine clamps against a formula that understates the output by three times, so the instrument is not safe to command above roughly 1.5 Vpp.

The wider lesson is about method rather than about this board. Every wrong answer in this project came from trusting a number instead of measuring it - the design formula, the inherited digital scale, the assumed probe attenuation, the assumed tap count. Every right answer came from a sweep that varied one thing and read the result back. The instrument is the argument for its own methodology.

References

1. **PH 3032 Embedded Systems Laboratory Course Manual**, Department of Electrical and Electronic Engineering, University of Peradeniya.
2. **WaveLink Project Site**, wavelink.krag.lk.
3. **WaveLink Report Site**, report.wavelink.krag.lk.
4. B. A. Billingsley and S. A. Billings, "System Identification: A Frequency-Domain Approach," **IEEE Transactions on Automatic Control**, vol. 54, no. 8, pp. 1820 to 1834, 2019.
5. Analog Devices, **X9C103 Digitally Controlled Potentiometer Datasheet**, Rev. C, 2021.
6. KiCad EDA, version 10.0.3. Schematic source files: `pcb_designs/mainboard/hardware/` (board 1) and `pcb_designs/mainboard/hardware_board2/` (board 2).