LCLS-II LLRF Quench and Detune Revisited

Larry Doolittle, LBNL, 2016-10-25

Simplify the usual cavity equation of state by including only one driven port, ignoring $1/Q$ of the probe, and postponing consideration of beam current. That leaves us with

$$\frac{d\vec{V}}{dt} = a\vec{V} + b\vec{K}_1$$

$$a = j\omega_d - \frac{1}{2}\omega_0\left(\frac{1}{Q_0} + \frac{1}{Q_1}\right)$$

$$b = \omega_0\sqrt{\frac{(R/Q)}{Q_1}}$$

If $Q_0 >> Q_1$, it's possible to write

$$\Re(a) = -|b|^2 \cdot \frac{1}{2\omega_0(R/Q)}$$

where ω_0 and R/Q should be quite well known.

Re-cast the state equation in terms of DSP-processed ADC readings $\vec{M}_V$ and $\vec{M}_K$, unitless where full-scale is represented as 1. Thus we need calibration constants c_V with units of Volts, and c_K with units of $\sqrt{\text{Watts}}$, so that

$$\vec{V} = c_V \cdot \vec{M}_V$$

$$\vec{K}_1 = c_K \cdot \vec{M}_K$$

Substitute and rearrange the state equation to get

$$\frac{d\vec{M}_V}{dt} = a\vec{M}_V + \left(b\frac{c_K}{c_V}\right)\vec{M}_K$$

a quick and direct *in-situ* experiment (short on-pulse followed by passive decay, recording waveforms of M_V and M_K) can determine $\Re(a)$ and $\beta \equiv b(c_K/c_V)$ with good accuracy. This same experiment is also useful to determine or cross-check the SEL phase offset term.

The relation above gives us

$$\frac{c_K}{c_V} = \frac{\sqrt{-2\omega_0(R/Q)\Re(a)}}{|\beta|}$$

which can be cross-checked with what we hope are independent measures of c_K and c_V.

All that is left is to rearrange the state equation one more time to

$$a = \frac{1}{\vec{M}_V} \cdot \left[\frac{d\vec{M}_V}{dt} - \beta \vec{M}_K \right]$$

The FPGA can compute $\vec{M}_V$ and $\vec{M}_K$ with good accuracy at very high rates (up to $\sim$3 MS/s). Getting acceptable accuracy from a meaningful derivative term will take longer; probably on the scale of 10 μs to 100 μs.

The strategy for tuning and quench detection comes into focus now. A DSP engine is fed a continuous stream of $\vec{M}_V$ and $\vec{M}_K$ with time separation T, from which it can also compute $d\vec{M}_V/dt$. It is pre-loaded with a (complex) value of β. After each new value of $\vec{M}_V$ and $\vec{M}_K$ is loaded, the equation above is used to compute (complex) a. The imaginary part of a is the detune frequency ω_d, which can be passed to the resonance control subsystem. The real part of a is negative; if it moves above some pre-loaded threshold (maybe 0.9 of the value of a found in the setup process), it is interpreted as a dramatic decrease in Q_0, and the quench detector trips.

This mechanism will, at least in theory, work under all modes of operation: fill, SEL, or closed-loop. As $|\vec{M}_V|$ gets smaller, measurement noise increases, and below some threshold the computation should be inhibited. In that case, the tune algorithms should freeze, and we hope it's safe to assert no-quench. From a tuning-loop perspective, we want to enforce a narrow range of voltage validity, and even some dwell-time within that range, to avoid trying to follow irrelevant Lorentz detuning when the gradient is not what's needed for beam operations.

Computing the derivative involves a complex difference followed by multiplying by the scalar representing $1/T$. One full complex multiply gives the second term inside the brackets. Then multiply by the conjugate of $\vec{M}_V$, and divide by $|\vec{M}_V|^2$. All told about 18 fixed-point scalar multiplies and 10 adds or subtracts are required.

So far this analysis has used s^{-1} as the units for $1/T$, β, and therefore a. Some other choice of inverse-time units will make sense when implementing this with fixed-point DSP. The software is free to configure that when choosing the numerical values of $1/T$ and β to load into the DSP engine. With a signed 18-bit representation and a bit-quantum of 0.01 Hz, full-scale would be $\pm$1310 Hz.

Restoring the beam current term to this analysis is not conceptually difficult, it just requires additional calibration and timing input. The real un-answered question is how to measure drifts in the β parameter in-place during CW operation.

The restriction that $|\vec{V}|$ is not small is a troubling one for a quench detector. If $\vec{V}$ is stuck at zero, even with forward power applied, the cavity could equally well be quenched or far

off-resonance. While a phase-ignoring power balance computation can possibly detect that condition, it's hard to generalize to the case with beam current, since the power absorbed is always phase-sensitive. Let's develop the power balance technique anyway, and claim that beam current will only be turned on when $\vec{V}$ is far from zero, so that the state-vector approach is running smoothly. Thus at least one of the two methods will always be valid.

First, refresh our memory that the reverse wave in the waveguide is

$$\vec{R} = c\vec{V} - \vec{K}$$

(ignoring waveguide losses) where $c = 1/\sqrt{Q_1(R/Q)}$. This knowledge is helpful when setting up test benches.

The way to account for the cavity's stored energy is through the equation

$$U = \frac{V^2}{\omega_0(R/Q)}$$

$$\frac{dU}{dt} = 2\Re\left(\vec{V}\frac{d\vec{V}}{dt}\right) \cdot \frac{1}{\omega_0(R/Q)}$$

The dissipated power in the cavity can then be estimated as

$$P_{\text{diss}} = |\vec{K}|^2 - |\vec{R}|^2 - \frac{dU}{dt}$$

If this exceeds some threshold power P_Q, we can claim a quench condition is detected. This representation purposefully uses the same notation and input measurements as the state-vector method.

Converting to a hardware perspective, a trip is caused by a positive sign on the quantity

$$f_K|\vec{M}_K|^2 - f_R|\vec{M}_R|^2 - f_V 2\Re\left(\vec{M}_V \cdot \frac{d\vec{M}_V}{dt}\right) - f_Q$$

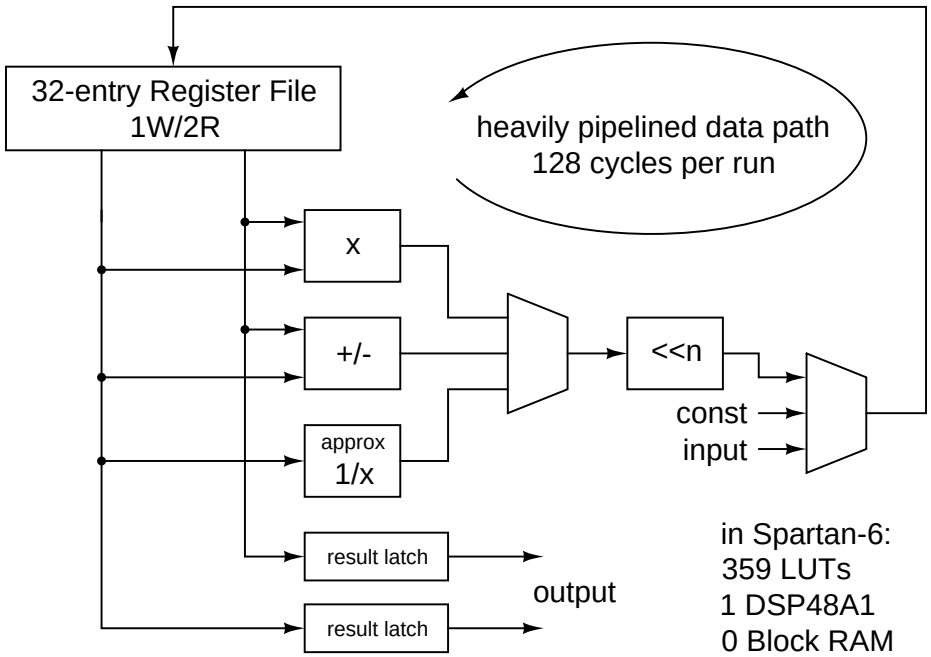
where $f_K = fc_K^2$, $f_R = fc_R^2$, $f_Q = fP_Q$, and

$$f_V = f\frac{c_V^2}{\omega_0(R/Q)}$$

and the free parameter f allows scaling from SI Watts to whatever internal number system is convenient. Here we see the software needs to calibrate and download f_K, f_R, f_V, and f_Q. The DSP engine needs to perform 9 fixed-point scalar multiplies and 6 adds or subtracts.

The arithmetic described here is more complicated than people usually associate with direct FPGA programming techniques. Indeed, functionality like this has historically been done on a dedicated DSP chip. Fortunately, FPGAs are big and fast enough now to subsume that programmability by means of various architectures of soft-cores. Our use case does not call even for a general-purpose soft-core, but rather one that can do a fixed sequence of fixed-point arithmetic (adds, subtracts, multiplies) every time a new set of RF vector measurements comes in.

Such a streamlined DSP core is on-the-shelf at LBNL; its basic architecture is shown here.



It is capable of running at higher clock speeds than even the double-rate DAC clock in the LCLS-II FPGA. It can perform one batch of this arithmetic in 80 cycles: 16 to load measurements and parameters, 45 useful arithmetic instructions, 2 output instructions, and 17 no-op cycles to wait for results to flow through its pipeline. The actual representation of the jobs to be done (both state-variable and the power-balance, and an FIR computation of the derivative of cavity voltage) is written as 17 lines of stylized Python, plus comments and surrounding overhead (see appendix). This gets machine-converted to an 80×22-bit program memory. As always, a good test bench is essential!

The 80 cycles of computation take little time compared to the many microseconds needed to develop a decent dV/dt measurement. The hardware resources used are tiny both compared to what's available on the chip, and compared to what would be needed for a direct mapping of the arithmetic to hardware.

Appendix: Program Listing

```
#!/usr/bin/python

# SRF cavity analog state computer
# Takes in cavity field, forward, and reverse vector measurements
# and computes the cavity detune frequency, decay parameter, and
# power imbalance for the purposes of a tuning loop and quench detector.
# Keeps a history of the previous four cavity field measurements so it
# can get dV/dt.

# Output of this program should be both valid c99 and valid input
# for the scheduler/mapper.
# See the rest of the Digaree infrastructure for details.

from cgen_lib import cgen_init, given, mul, sub, cpx_sub, cpx_mul
from cgen_lib import cpx_scale, cpx_dot, cpx_inv_conj, cpx_mul_conj
from cgen_lib import cpx_mag, set_result, cpx_persist, cpx_copy, cpx_add
cgen_init("cgen_srf.py")

# History of measured cavity voltages, used to compute dV/dt
# Initial values in simulation are read from init.dat or init2.dat.
cpx_persist("v1")
cpx_persist("v2")
cpx_persist("v3")
cpx_persist("v4")

# These lines declare the input variables,
# first six streamed from the radio
given("k_r")  # forward
given("k_i")  # forward
given("r_r")  # reverse
given("r_i")  # reverse
given("v_r")  # cavity
given("v_i")  # cavity
# next eight host-settable
given("beta_r")
given("beta_i")
given("invT")
given("two")  # needed by 1/x macro
given("sclr")
given("sclf")
```

```
given("sclv")
given("powt")

# Get (still unscaled) derivative
# Implements [-2 -1 0 1 2] FIR
cpx_sub("dv1", "v", "v4", 3)  # note multiply-by-4
cpx_sub("dv2", "v1", "v3", 2)  # note multiply-by-2
cpx_add("dvx", "dv1", "dv2", 3)  # note multiply-by-4
# Result is the amount that V will change in 80*T.
# Including the second-order CIC used to generate input samples,
# this computation has a 3*T group delay.

# State-variable computation of the complex number a,
# yielding detune frequency and decay rate
cpx_inv_conj("x5", "v", 0, 3)
cpx_scale("dvdt", "dvx", "invT", 1)
cpx_mul("x3", "k", "beta", 1, 1)
cpx_sub("x4", "dvdt", "x3", 2)   # some evidence this shift should be 1
cpx_mul_conj("a", "x4", "x5", 2, 2)
set_result("ab", "a_r", "a_i")

# Power balance measure of cavity dissipation; uses magnitudes only
cpx_mag("magr", "r", 0)  # reverse
mul("powr", "sclr", "magr", 0)
cpx_mag("magf", "k", 0)  # forward
mul("powf", "sclf", "magf", 0)
sub("wgnet", "powf", "powr", 1)  # net power transferred by waveguide
cpx_dot("dvsq", "v", "dvx", 2)   # 2 * V * dV/dt = d/dt(V^2)
mul("dudt", "dvsq", "sclv", 3)   # dU/dt = power to stored energy
sub("diss", "wgnet", "dudt", 1)  # est. of dissipation in cold cavity
sub("perr", "diss", "powt", 1)   # allow for measurement error
set_result("cd", "diss", "perr")  # trigger quench fault if perr > 0

# Watch these like a hawk:  order of execution matters,
# unlike everything else here
cpx_copy("v4", "v3")
cpx_copy("v3", "v2")
cpx_copy("v2", "v1")
cpx_copy("v1", "v")
```