GH100 System Architecture Specification

Revision History

Version	Author / Editor	Description	Date
0.10	Pervez Aziz	Initial version. Includes stakes in the ground for digital TXFIR, VGA adaptation algorithm (ymx target criterion), preliminary CDR gradient definition (zero crossing only transition combinations).	6/21/2019
0.11	Pervez Aziz	CDR gradient computation split into separate proportional and integral paths. Simplified gradient calculation for proportional path specified to allow reuse of the GA100 prop. path VCO DAC.	7/5/2019
0.12	Pervez Aziz	Update to nominal 7 bit ADC. Use +6 level at 75% of ADC vfs instead of 95% with prior 6 bit ADC. Update various pam4pr1 levels accordingly in terms of 7 bit code values. For the existing sets of zero crossing transitions use all 18 of them as default. Propose tri level sign for error in the CDR gradient as default. Accordingly remove prop. elsum clipping.	7/11/2019
0.13	Pervez Aziz	Added overview RX block diagram. Merged RXFFE in from separate RXFFE spec. Includes two possible amplitude constraint frameworks and proposed sets of tap ranges, corresponding bit widths.	9/24/19
0.14	Pervez Aziz Viswanath Annampedu	Polished TXFIR description and included explicit tap ranges to meet IEEE and PCIE gen5 specs. Equivalent bit widths noted. Removed fixed f(0) option for RXFFE. Updated RXFFE ranges based on Imtinan's SLAN simulation data. Equivalent bits noted for each TXFIR/RXFFE coefficient based on ranges. Tentative RXFFE output signal width specified. DFFE overview and detailed description added. Tentative DFFE coefficient width and signal path width specified.	11/11/2019
0.15		Sorry – skipped this accidentally.	
0.16	Viswanath Annampedu	Added details on RX ADC offset calibration gradient. (spec doc named 0.16 but this table showed it as 0.15).	11/21/2019
0.17	Pervez Aziz	Revamped VGA/AGC adaptation to be based on ymx criterion without requiring max operation across 32T, much simpler gradient. Flow chart for VGA adaptation provided. Gradients for slicer (level) adaptation provided, gradient for RXFFE adaptation provided, as well as flow chart / sequencing covering joint VGA/slicer(level)/CDR/RXFFE adaptation. Updated CDR gradient to what is simulated now in BASES. Apples/apples BASES comparison and potential update pending on 28 transitions from Wei-Hou which is in RTL now. Noted TBD areas of current content in red.	12/9/2019

0.18	Pervez Aziz	(1) Simplification of ymx estimation gradient using logical OR across 32T instead of sum. (2) Change accthr to 0 from 1 in ymx adaptation. (3) Add generic gmac block diagram for yl1, rxffe adaptation. (4) Modify yl1 adaptation to use +/-4 sliced data as well (5) Simplify yl1 and RXFFE adaptation to use 3 level sign of yslc. (6) Simplify yl1 and RXFFE adaptation to use 2 level sign error instead of 3 level sign error. (7) Reduced set of RXFFE tap ranges and added coefficients up to f(5). (8) Add gmac block diagram for yl1/rxffe adaptation (9) Made note that low power RXFFE mode to be f(-1) to f(1) only, block diagram not updated yet.	12/19/2019
0.19	Pervez Aziz and Viswanath Annampedu	Filled in a few missing details for gradient and accumulation used in BASES and system simulations. Explicit instantiation of transitions used.	01/24/2020
0.20	Pervez Aziz	Updated RXFFE to be 3pre/8pst. Reverted RXFFE to fixed *f(0)* case with *f(0)*=128. Specified tap ranges based on Imtinan's SLAN data as well as correlation with BASES simulations. Revised RXFFE output bit width to next stage to be 12 bits. Introduced f(1) constrained adaptation. Updated CDR integral path gradient to follow proportional path i.e. use 3 level sign error and data per Vish's simulation data. TXFIR definition completed. To fit TXFIR output to DAC range with div by 4 operation adopted Toan's proposal to make internal coefficients 1/84 resolution.	02/17/2020
0.21	Pervez Aziz	Fixed typo *f(-2)* to *f(2)* on constrained *f(1)* adaptation equation. Created an equation based description for input bit width reduction for the RXFFE and added corresponding reduction number's from GuoHao's SLAN simulation data proposal, and made corresponding reduction of RXFFE output and slicer input bit width to 11 bits. Minor update to *f(7), f(8)* ranges. Replaced equalization sequencing diagram with updated one (subject to further review by broader team). After discussion with Toan make *c(-1)* limit to be -31 (in 84 domain). So increase to -31 makes it -23 in 63 domain. Removed reference to pre-cursor cancellation and second stage for the DFFE implementation, data width made 11 bits consistent with RXFFE output, new implementation block diagram from Vish without pre-cursor replaces existing one.	03/13/2020
0.22	Pervez Aziz	Updated eq. sequencing diagram to reflect individual level	04/08/2020

		adaptation after CDR is enabled. Added details on adapting individual data levels (error slicer thresholds) and computing data slicer thresholds from bisecting these levels. Updated data level bit width to be 11 bits. Updated el_sum computations for slicer level and RXFFE adaptation to reflect gmacs operating at 64T vs. 32T. Noted desire to have total RXFFE bypass option for pcie gen5. Based on Guo Hau's SLAN simulations showing benefit in phase acquisition range improvement, added option for modified MMPD *f(1)* constrained adaptation which excludes *f(2)* from the current constraint.	
1.00 (jump rev # from 0.23)	Pervez Aziz, Yuan-Hao Tung, Viswanath Annampedu, Wei-Hou Tay, Imtinan Elahi	TXFIR: *c(0)*min, explicit rounding table description, tweak to tap range. Final choices of CDR gradient transition tables. Noted and described original BASES table as default acq. table, RTL table from WeiHou as trk table and included backup acq/trk tables from Yuan-Hao. Minor updates to some higher level block diagram, update top level sequencing diagram. Summarized adaptation mode types for various loops. Added ATT/VREF adaptation illustration. Added vref code to ADC VFS voltage relationship equation. Added CTLE example magnitude responses and SLAN grid search flow chart from Imtinan. Changed CDR gradient processing to sign-sign and included top loop filter block diagrams. High level description of illegal data det / phase kick (csdet) for pam2pr1 and pam4pr1 and related pattern sets included. Included extra pam4pr1 patterns from WeiHou. Added basic section on phase offset calibration (phos). Included details for additional RXFFE adaptation features: LMS and fixed *f(1)*. Updated RXFFE coefficient ranges to occupy full power of 2 ranges instead of non-power of 2 clipped/limited. Updated DFFE tap position/value ranges and included DFFE position adaptation details from Yuan-Hao. Added comprehensive RX Cal section to include startup and continuous calibration loop details from Yuan-Hao. Updated example BASES (current snapshot) simulation results and included additional example results for more loops.	11/09/2020

Table of Contents

1 Introduction

An overview of the GH100 NVHS4 architecture is shown in Figure 1.

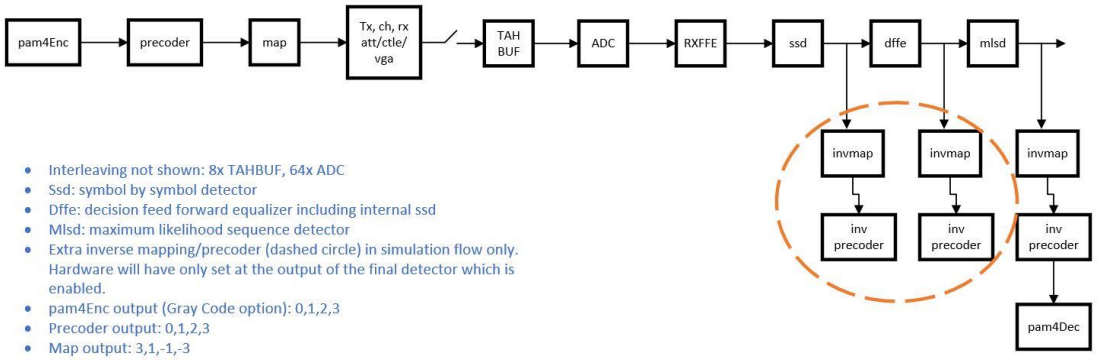


Figure 1: High level system architecture block diagram for pam4pr1 mode.

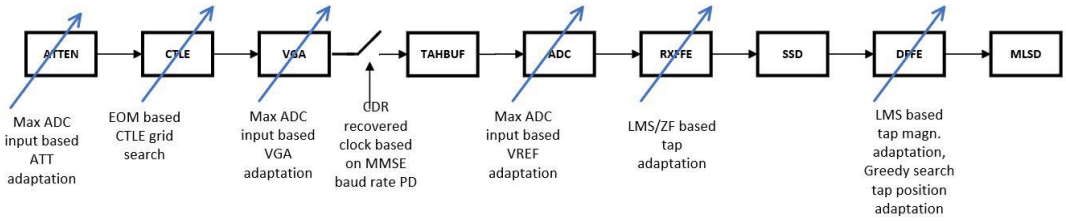


Figure 2: High level functional receiver block diagram

The block diagram of Figure 2 provides an overview of the receiver (RX) architecture. It consists of a usual continuous time linear equalizer (CTLE) whose output is sampled with a recovered clock derived by the variable gain adjust (VGA) block. The VGA output is quantized with an analog to digital converter (ADC) for further digital equalization by a RX feed forward equalizer (RXFFE) and a decision feed forward equalizer (DFFE). The DFFE output is finally decoded by a maximum likelihood sequence detector (MLSD). The MLSD is further processed (outside of the PHY) with a Reed-Solomon decoder for error correction. Note that one or more blocks may be optionally powered down. Also the adaptation algorithms for various loops may be derived from outputs tapped at programmable locations in the data path chain e.g. CDR could be driven by the ADC output, RXFFE output or first stage DFFE output. Further details are not shown in this diagram to maintain overall clarity of the data path flow.

2 Digital TXFIR

GH100 will implement a digital TXFIR. A block diagram of it is shown in Figure 3. This is a full rate functional diagram only and not meant to dictate the implementation which for example could make use of pipelining or half rate processing. The TXFIR is shown with 5 taps to be IEEE 802.3ck specification compliant. There are 3 pre-cursors and 1 post-cursor. The TXFIR operation can be described with the following equation:

$$y(n)=x(n)c(-3)+x(n-1)c(-2)+x(n-2)c(-1)+x(n-3)c(0)+x(n-4)c(1)$$

.

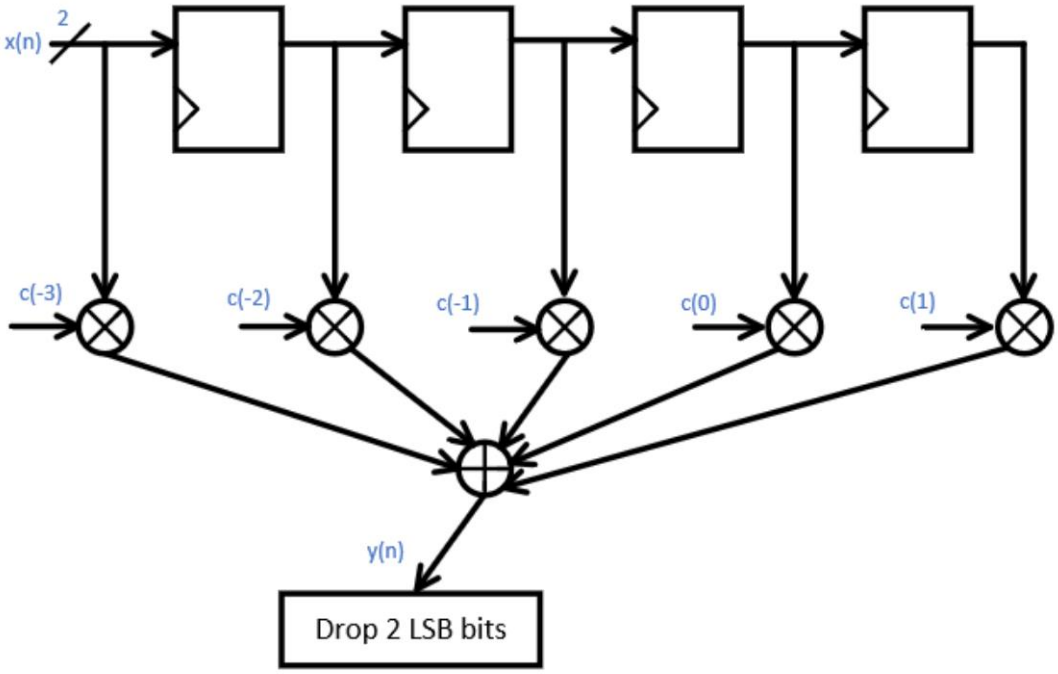


Figure 3: Full rate block diagram of digital TXFIR

The TXFIR inputs *x(n)* consist of : [-3, -1, 1, 3]. As with the analog TXFIR of GA100, the sum absolute value of TXFIR coefficient is constrained to a max value, *M*. Note that *c(0)* will be computed from the other coefficients to maintain the sum magnitude constraint.

$$c(0)=M-[\ |c(-3)|+|c(-2)|+|c(-1)|+|c(1)|\ ]$$

Initially the plan was for the sum abs constraint to be M= 63 with *c(0)* taking on a max value of 63 for 6 bit coefficient magnitudes (as we had before). With the TXFIR coefficient constraints the TXFIR output can be at most -3*63 to +3*63 or -189 to +189. However our TX FIR DAC is a 7 bit DAC to minimize power, it covers a range of -63 to +63. This would have required a division by 3 of the TXFIR output to fit the DAC range. Toan has proposed to expand the coefficient space to cover a range of 84 for the max *c(0)* i.e. *M*=84 instead of *M*=63. With *M*=84, the max TXFIR outputs are +/- 252. This can be easily scaled by a factor ¼ i.e. dropping of 2

LSBs to obtain a range of -63 to +63 which fits the DAC range. The resulting **internal** representation of the TXFIR coefficient values and ranges are shown in Table 1.

Coefficient	**Min Range**	**Max Range**	**Min Range**	**Max Range**	**Number of Bits**
	LSBs	**LSBs**	**(% of max c(0))**	**(% of max c(0))**	
c(-3)	-7	0	-8.3	0	3
c(-2)	0	11	0	13	4
c(-1)	-31	0	-37.0	0	5
c(0)	45 (+)	84	54 (+)	100	7
c(1)	-28	0	-33	0	5

Table 1: Transmitter FIR internal coefficient ranges and bit widths in 1/84 domain to meet IEEE 802.3CK and PCIE Gen5 specifications. Note: The c(1) range is determined by pcie gen5. For IEEE it would have been 20%. (+) c(0)min is based on the IEEE 802.3CK spec. but not implemented in the RTL – it needs to be enforced by the controller in how it programs the TX or how it allows up/dns to the TX to be processed.

The ranges specified are internal range because PCIE and some other standards do not support coefficients with 1/84 resolution. Therefore from an external interface point of view, standards which support only coefficients with 1/63 resolution will have to be translated to into the internal coefficients of Table 1 with resolution of 1/84. This can be done using a look up table (LUT) which performs the mapping using a round operation:

*c(i)_internal = round(c(i)_external *84/63)* except for *i=0*

The enclosed spreadsheet contains the actual LUT for positive values of the coefficient. Note that the input magnitude is specified for 0 to 63. However, for *c(-3), c(-2), c(1)* the maximum magnitude will be less and only the entries relevant for the corresponding range need to be supported.

coeff	scaled coeff 84
63	84
62	83
61	81
60	80
59	79
58	77
57	76
56	75
55	73
54	72

53	71
52	69
51	68
50	67
49	65
48	64
47	63
46	61
45	60
44	59
43	57
42	56
41	55
40	53
39	52
38	51
37	49
36	48
35	47
34	45
33	44
32	43
31	41
30	40
29	39
28	37
27	36
26	35
25	33
24	32
23	31
22	29
21	28
20	27
19	25
18	24
17	23
16	21
15	20
14	19
13	17
12	16

11	15
10	13
9	12
8	11
7	9
6	8
5	7
4	5
3	4
2	3
1	1
0	0

Table 2: Converting 1/63 domain external coefficients to 1/84 domain internal coefficients

For protocols such as NVHS which can be updated to support 1/84 resolution coefficients, the coefficients would be programmed directly at the 1/84 resolution.

Coefficient	**Min Range**	**Max Range**	**Min Range**	**Max Range**	**Number of Bits**
	LSBs	**LSBs**	**(% of max c(0))**	**(% of max c(0))**	
c(-3)	-5	0	-7.9	0	3
c(-2)	0	8	0	12.7	4
c(-1)	-23	0	-37.0	0	5
c(0)	34 (+)	63	54 (+)	100	6
c(1)	-21	0	-33	0	5

Table 3: Transmitter FIR internal coefficient ranges and bit widths in 1/63 domain to meet IEEE 802.3CK and PCIE Gen5 specifications. Shown in 6 bit format. (+) c(0)min is based on the IEEE 802.3CK spec. but not implemented in the RTL – it needs to be enforced by the controller in how it programs the TX or how it allows up/dns to the TX to be processed.

3 Signal Nomenclature

GH100 is an ADC based Serdes. Let *yadc(n)* be the ADC output sample. Let *yrxffe(n)* be the output of the RXFFE. We assume that equalization to PAM4PR1 is achieved at the output of the RXFFE to the extent of being able to produce a distinct 7 level eye. Of course, additional ISI cancellation happens through the DFFE. However, for now we focus on the RXFFE output which will drive our VGA, CDR, slicer, RXFFE, and offset adaptation algorithms.

In addition to the raw output, the RXFFE will produce the tentative symbol by symbol detector (SSD) decisions or slicer decisions, $yslc\_rxffe(n)$. For the pam4pr1 there will be 7 distinct SSD decision values. The SSD outputs can be +6, +4, +2, 0, -2, -4, -6 (in the hardware implementation they could be divided by 2 to reduce the number of bits processed in subsequent arithmetic). Let us consider the corresponding values to be *ylp6, ylp4, ylp2, yl0, ylm2, ylm4, yl6m*. Note that these data levels will also correspond with the error slicer thresholds at this level. Let us also consider the corresponding data slicer thresholds at *ylp5, ylp3, ylp1, and ylm1, ylm3, ylm5*.

SSD Symbol $yslc\_rxffe$	**SSD Value Name** (Data Level = Error Slicer Threshold)	**RTL Name**
6	*ylp6*	*eth6*
4	*ylp4*	*eth5*
2	*ylp2*	*eth4*
0	*yl0*	*eth3*
-2	*ylm2*	*eth2*
-4	*ylm4*	*eth1*
-6	*ylm6*	*eth0*

Table 4: Mapping of YSSD values to corresponding full resolution data signal / error slicer threshold levels and equivalent RTL names.

SSD Threshold Name Data Slicer Threshold	**RTL Name**
ylp5	*y5*
ylp3	*y4*
ylp1	*y3*
ylm1	*y2*
ylm3	*y1*

ylm5	*y0*

Table 5: Data slicer thresholds and equivalent RTL names.

We will be deriving all quantities used by various adaptation loops from *yrxffe(n)* and corresponding SSD symbol decision values, $yslc\_rxffe$. Also, *during initial FLL mode* we will assume that the signal is nominally symmetric and essentially ignore any impact of transmitter Rlm. This means that positive and negative levels will be considered to be symmetric and actually all related to the nominal *ylp1* level via integer ratios.

After the initial FLL mode, we will later fine tune our levels further by adapting each level *yl[p,m][6,4,2],yl0* individually and will result in slicer thresholds *yl[p,m][5,3,1].*

4 Top Level Equalization / Clock Sequencing

We will initially perform an acquisition of the ADC input referred offset through a start up offset calibration acquisition (ADCVOS). Likewise there will be an initial acquisition of the CTLE offset (CTLEVOS) and VGA offsets (VGAVOS) and ADC gains (GOS). The clock data recovery (CDR) loop will not be enabled during this time which means we will effectively be in FLL mode. We will discuss further details of the offset calibration later.

Assuming the initial (startup) calibration is performed, we then adapt the VGA based on a "max" criterion to maintain the ADC output maximum within a certain programmed lower and upper window. Once the VGA places our signal amplitude within the desired window we perform an initial adaptation of the slicer levels at the RXFFE output. We seed the initial value of the slicer levels based on the achieved max ADC output target multiplied by the DC gain of the RXFFE. After this initial slicer adaptation, we enable the CDR and jointly adapt it with the RXFFE, slicer levels, CTLEVOS and PHOS loops. We then enable the periodic ADCVOS adaptationfollowed by periodic ADCGOS adaptation. The DFFE taps can then be subsequently adapted. This procedure is repeated for different values of cdr_phase_offset. A diagram of this sequencing is shown in Figure 4

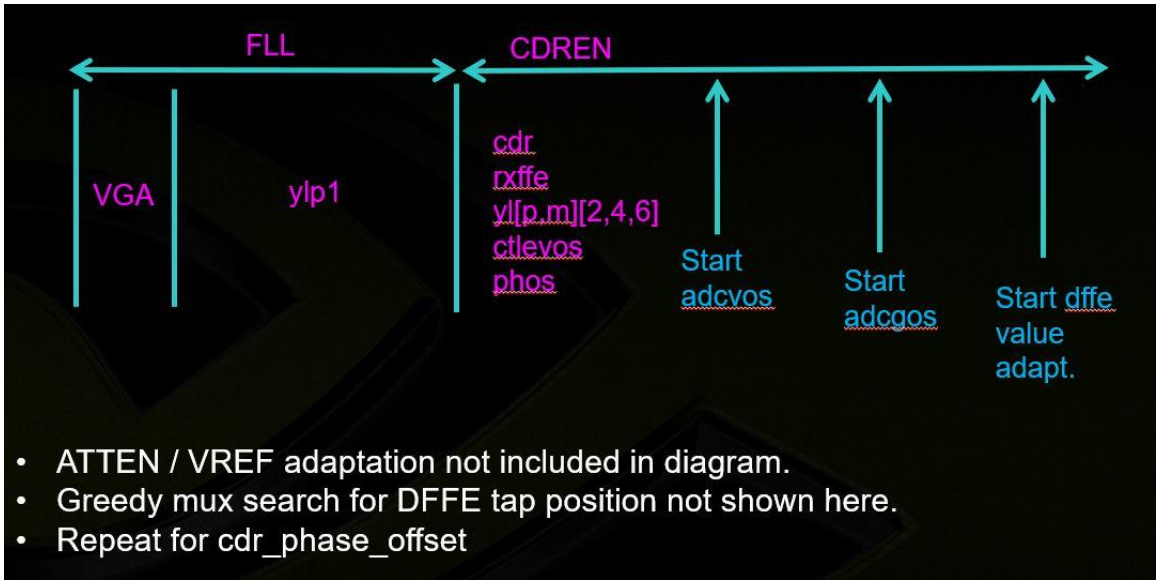


Figure 4: Joint adaptation sequencing.

This sequencing scheme has been simulated in BASES. For BASES simulation purposes, in FLL mode we have a frequency dither superimposed on a static frequency offset (SFO). During BASES CDR enabled mode the frequency dither is disabled and only the SFO is enabled so that the injected SFO is easily correlatable with the observed integral path register behavior of the CDR loop filter.

After the initial / startup calibration the various loops will be either adapted only once, adapted continuously or adapted periodically. This is documented in Table 6

Adaptation Loop	**Adaptation Mode After Initial/StartUp Calibration**
VGA	One time
ATT	One time
CDR	Continuous
ADCVOS	Continuous
ADCGOS	Continuous
CTLEVOS/BLW	Continuous
PHOS/PHCAL	Continuous
RXFFE	Periodic
ETH (slicer)	Continuous for 106G, otherwise periodic.
DFFE	Periodic
VREF	Periodic

Table 6: Adaptation modes for various loops

5 VGA Adaptation Algorithm

5.1 Top Level Flow

With an ADC based Serdes we need to maintain the proper amplitude at the ADC input to avoid significant ADC overload/clipping. We can do this by comparing the ADC samples to a desired max ADC output target and updating the VGA accordingly.

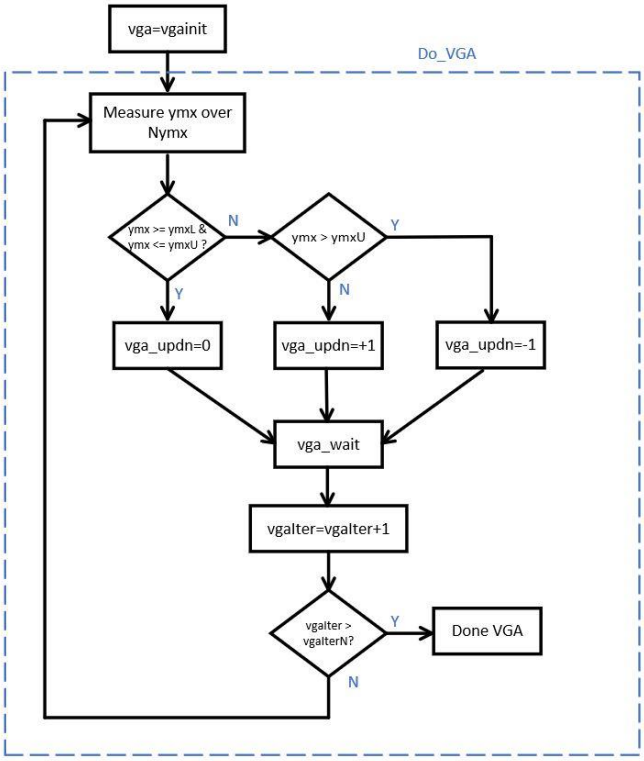


Figure 5: VGA adaptation flow chart

A top level flow of the VGA adaptation is shown in Figure 5. We the VGA is initialized to some code and starts adapting from there. We measure the maximum ADC output "*ymx*" over some interval of time. If the *ymx* exceeds the upper window value *ymxU* we decrement the VGA by one code, if it falls below the lower window value *ymxL* we increment the VGA by one coder. If the *ymx* falls within the window with equality we do nothing. The whole process repeats a finite number of times as determined by the parameter *vgaIterN*.

VGA Parameter	**Programmable Values** **(Tentative)** **(Final TBD)**	

ymxL	60, 54, 48	
ymxU	[48..62]	
vgaitersN	1 to 32	

Table 7: VGA adaptation parameters

VGA DAC Min Output	1dB
VGA DAC Max Output	8.5dB
VGA DAC Step	1dB

Table 8: Nominal VGA table properties.

5.2 *ymx* Measurement

We measure *ymx* by effectively trying to find the outer envelope of the ADC output signal within some window of time. The algorithm to do this is shown in Figure 6.

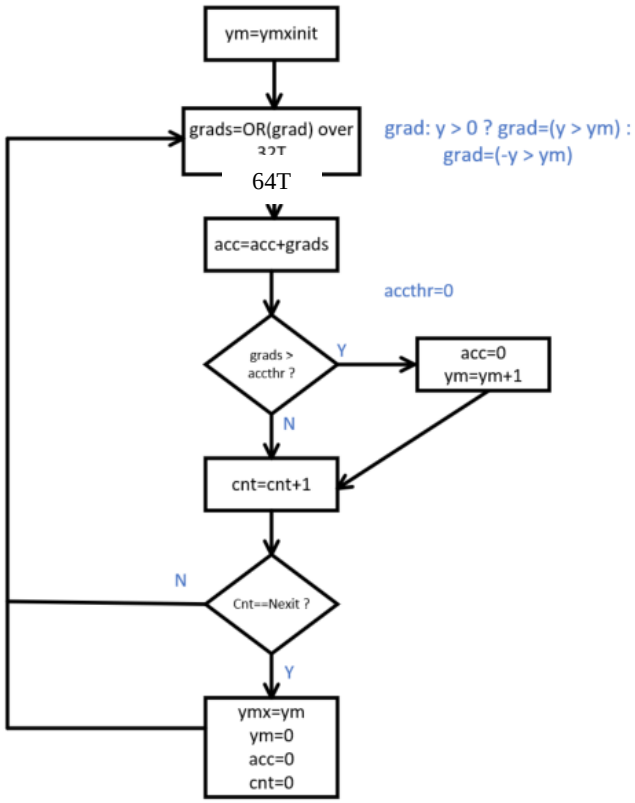


Figure 6: ymx algorithm to drive VGA loop. In this figure, y=yadc

$$[yadc(n) > 0 \;\; grad(n) = (yadc(n) > ym) : grad(n) = (-yadc(n) < ym)\,] \qquad \text{Eq(1)}$$

The gradient computation for the *ymx* calculation is shown in Eq(1) and it essentially determines if the absolute value of the current ADC sample is greater than the current tentative *ymx* value *ym*.

Parameter	Value / Programmability	Units
accthr	1 (need not be programmable)	LSB
Nexit	64, 128, 256, 512	64T.
ymxinit	0	
ym, ymx	6 bits wide	
acc	1 bit	

Table 9: ymx measurement parameters

5.3 Attenuation / Vref Loops

In addition to being able to adapt the VGA, the RX is able to adapt an attenuator or ATT setting incorporated into the RX termination circuit. It is also able to adapt the ADC full scale voltage VREF if a combination of adapting the VGA and ATT setting do not result in the the ADC output achieving the desired target range.

The attenuator supports 4 values as noted in the table below.

Attenuator / termination code (trmAtten)	Attenuation Factor	Attenuation (dB)
0	1x	0
1	0.66x	-3.61
2	0.55x	-5.19
3	0.44x	-7.13

Table 10: Attenuator / termination settings

The nominal ADC VFS is 275mV. The equation relating VREF voltage to vref_code is:

ADCVFS=167mV + 2.4mV**vref_code[5:0]* (Default Range)

Thus the nominal 275mV would be set using a *vref_code* of 45. If adapting VREF, the *vref_code* should be swept no more than +/15 codes i.e. the resulting ADVFS should be kept in the range of 239mV to 311mV. There is also an extended range mode in which case

ADCVFS=167mV + 3.3mV**vref_code[5:0]* (Extended Range)

The ATT / VREF knobs can be invoked and adapted per the following block diagrams:

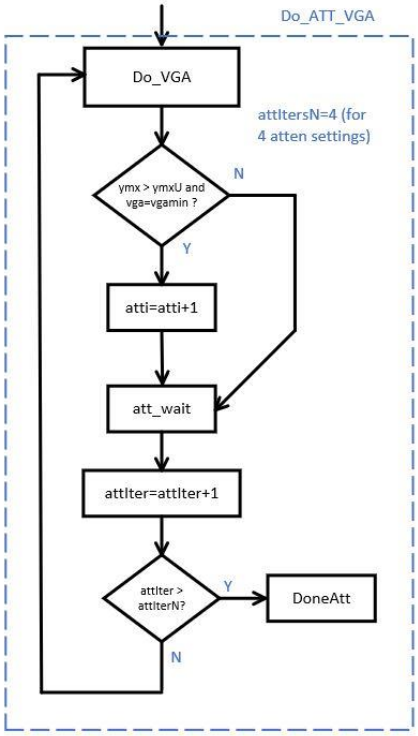


Figure 7: Invoking the attenuator (ATT) in addition to the VGA

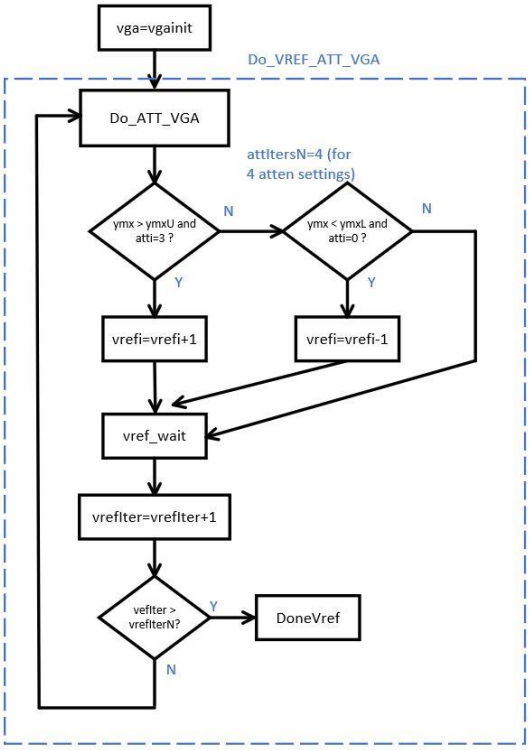


Figure 8: Invoking both the attenuator (ATT) and VREF control

6 CTLE Knobs, Example Responses, Grid Search

The continuous time linear equalizer (CTLE) used in the analog front end can be controlled through 6 independent parameters: *dcg*, *hfg*, *lfp*, *lfg*, *mfg*, *mfp*. The parameters control:

- *dcg[2:0]* – DC gain
- *hfg[3:0]*– high frequency peaking around fBaud/4
- *lfg[2:0]* – low frequency gain
- *lfp[2:0]*– low frequency pole
- *mfg[2:0]* – mid frequency gain
- *mfp[2:0]* – mid frequency pole

The (*lfg*,*lfp*,*mfg*,*mfp*) perform what is considered long tail (LT) equalization i.e. in the time domain equalize the tail of the pulse response further away from the main lobe. In the frequency domain, the "*lf*" controls tend to equalize the signal spectrum in the areas 300 MHz and the "*mf*" controls tend to equalize the signal spectrum in the 300MHz to 3GHz region. Note that the dc gain is not a pure flat gain as changing the *dcg* value will impact the low frequency characteristic of the CTLE as well.

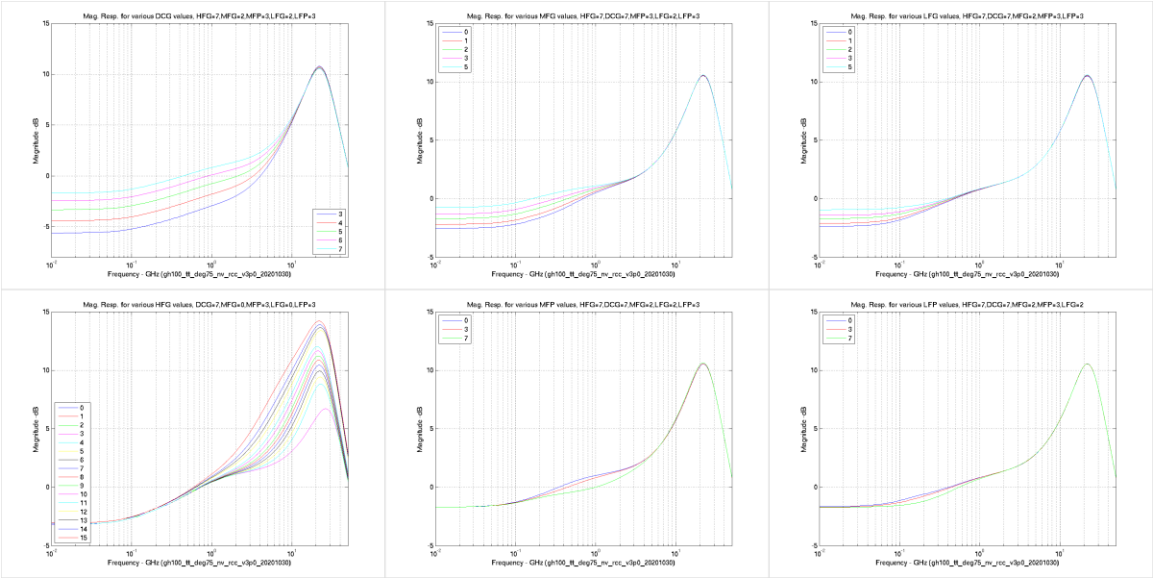


Figure 9: Example CTLE magnitude responses showing impact of varying individual control knobs. Responses are post-layout TT, 75 degrees C.

To improve the performance of very short channels including chip to module (C2M) channels with a PR1 equalization target we require an added low-pass filter (LPF) function. This is incorporated into the lower values the *hfg* values. In particular, *hfg*={0,1,2} transfer functions are copies of *hfg*=3; however, *when* hfg={0,1,2}, the LPF in the termination block is enabled with its corner frequency being a function of *hfg* – *hfg*=0 has the lowest corner frequency and *hfg*=2 has the highest corner frequency. As noted in the RX block diagram the CTLE is adapted using a

grid search method based on a performance figure of merit (FOM). The following method has been used:

Set a given LTE preset (e.g. 2323, 0303, 5353, ...)

Step-1: For each tuple (dcg,hfg), where dcg ∈{3:7}, hfg ∈{0:15}

A) Set trmAttn=0

B) Find VGA to meet ymxL ⩽ ymx ⩽ ymxU

C) If (ymx ⩽ ymxU && VGA ⩽ VGAmax), pick trmAttn

D) Else trmAttn+=1; goto step-B

E) Compute FOM

Pick (dcg,hfg) with the best FOM

Step-2: For each tuple (mfg,mfp,lfg,lfp), where mfg&lfg ∈{0,1,2,3,5}, mfp&lfp ∈{0,3,7}

A) Set trmAttn=0

B) Find VGA to meet ymxL ⩽ ymx ⩽ ymxU

C) If (ymx ⩽ ymxU && VGA ⩽ VGAmax), pick trmAttn

D) Else trmAttn+=1; goto step-B

E) Compute FOM

Pick (mfg,mfp,lfg,lfp) with the best FOM

7 Slicer / Level Placement and Adaptation

7.1 Slicer Level Placement During FLL Mode

Let *yslc_rxffe(n)* be the slicer RXFFE output. We obtain the sliced values as follows.

- If (*yrxffe(n) > ylp5*) then *yslc_rxffe(*n)=+6
- else if (*yrxffe(n) > ylp3 & yrxffe(n) <=ylp5*) then *yslc_rxffe(n)*=+4
- else if (*yrxffe(n) > ylp1 & yrxffe(n) <=ylp3*) then *yslc_rxffe(n)*=+2
- else if (*yrxffe(n) > ylm1 & yrxffe(n) <= ylp1*) then *yslc_rxffe(n)*=+0
- else if (*yrxffe(n) > ylm3 & yrxffe(n) <=ylm1*) then *yslc_rxffe(n)*= -2
- else if (*yrxffe(n) > ylm3 & yrxffe(n) <= ylm3*) then *yslc_rxffe(n)*= -4
- else *yslc_rxffe(n)*= -6

Eqtn(2A)

During FLL mode, the levels are obtained by adapting the single quantity *ylp1* and computing others from it as follows:

- *ylp6=6*ylp1*
- *ylp5=5*ylp1*
- *ylp4=4*ylp1*
- *ylp3=3*ylp1*
- *ylp2=2*ylp1*
- *ylm1=-ylp1*
- *ylm2=-2*ylp1*
- *ylm3=-3*ylp1*
- *ylm4=-4*ylp1*
- *ylm5=-5*ylp1*
- *ylm6=-6*ylp1*

Eqtn(2B)

The error at the RXFFE output is computed as follows:

- *err_rxffe(n)=y_rxffe(n)-ylp1*yslc_rxffe(n)* Eq(2)

Note that equation for *err_rxffe(n)* is a mathematical expression only and need not be explicitly computed if it is simpler to directly compute the sign of this error as used by the *ylp1* and RXFFE adaptation loops using a digital slicer.

7.2 Slicer Level Adaptation During FLL Mode

7.2.1 *ylp1* Initialization

We will adapt the slicer level *ylp1* using a LMS type of algorithm. After the VGA has converged, we initialize *ylp1* to *ylp1init=ymxL*rxffe_dcgn*. The idea is that the slicer level can be

approximated by the lower value of the ADC output times the RXFFE dc gain which is given by the sum of the RXFFE taps *f(l)* per the expression below.

$$rxffe\_dcgn = \sum_{l=-N2}^{N1} f(l)$$

7.2.2 Bit Width Considerations For *ylp1* and Other Data Levels

From this *ylp1init* value we further adapt the *ylp1* value. Let us now consider bit widths required for *ylp1* and other related quantities. We also recommend initial RXFFE coefficient values of [0 0 -30 128 98 0 0 0 0 0 0]. For *ymxL=60* we obtain *yl1init*=1960. The RXFFE output is nominally 15 bits, however, in reality we actually drop 4 bits to obtain a 11 bit output. The yl1init correspondingly becomes 122. The corresponding *ylp6* (nominally 6 times *ylp1)* is 732 which as a signed quantity is a 11 bit number.

Let us also consider this from examining some nominal simulation data. The data levels with full resolution 15 bit RXFFE ranges in decimal from [-8abc to +8def] (e.g. -8935 to +8966) i.e. 15 bits, dropping the 4 bits to have the corresponding 11 bit RXFFE output, this becomes [-5pqr to +5xwz] i.e. a 11 bit number. Although in principle we can optimize the bit width for each individual level, it may be simpler to keep a common 11 bit width for all the data levels.

7.2.3 Adaptation of *ylp1*

The gradient for slicer level adaptation is:

*Δylp1=sgn[err_rxffe(n)]*sgn[yslc_rxffe(n)]* when *yslc_rxffe(n)*=+/-6 or +/-4
(gradient=0 otherwise)

Eq(3)

where

$$sgn(x) = (x => 0\ ?\ 1 :\ -1) \quad \text{Eq(4)}$$

Note that since we adapt *ylp1* this precludes a division by 6 which would have been required had we adapted *ylp6* and computed *ylp1* from *ylp6*.

This gradient is used to update a traditional GMAC of the type we have used in prior projects to adapt the DFE taps as shown in Figure 10. As we will see from the RXFFE section, the +6 level *ylp6* is nominally 11 bits wide. So the nominal *ylp6* will be a 10 bit magnitude quantity. Here we adapt *ylp1* assuming it is a 10 bit unsigned magnitude. Using the traditional GMAC notation, we obtain the 64T accumulated version of the *ylp1* gradient as:

$$elsum\_ylp1 = \sum_{k=0}^{63} \Delta ylp1(n) \quad \text{Eq(5)}$$

The gmac accumulator is updated as follows.

*gmac_ylp1(n)=gmac_ylp1(n-1) + 2^(gmac_shift_ylp1)*elsum_ylp1* Eq(6)

Slicer Level Adaptation GMAC parameter	**Value**	**Comments**
Integer Bits (represents *ylp1*)=NI	11	
Number of Fractional Bits=NF	15	
gmac_shift_ylp1	0 to 15	Tentative

Table 11: *ylp1* adaptation gmac parameters

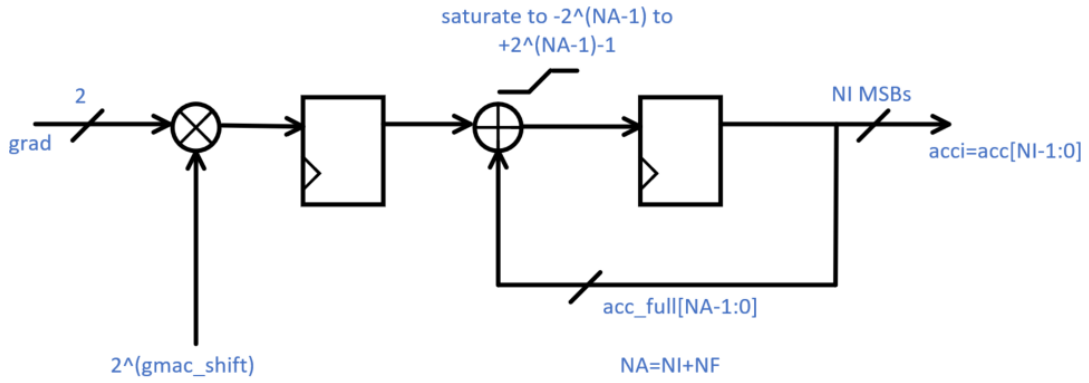


Figure 10: Block diagram of traditional GMAC structure used for *ylp1*, RXFFE adaptation, and data level (error slicer) adaptation. GMAC updates at the 64T rate.

.

7.3 Slicer Level Placement and Adaptation Once CDR is Enabled

Once we have moved out of FLL mode and enabled the CDR, we will the expressions in Eq (2A) to as before to determine the digital slicer outputs. However we will no longer use the expressions in Eq (2B) to compute the various levels from *ylp1*. The errors will be computed independently for each of the data levels as per Table 12.

SSD Symbol $\hat{y}(n)$	*e(n)*
6	*y(n)-ylp6*
4	*y(n)-ylp4*
2	*y(n)-ylp2*
0	*y(n)*
-2	*y(n)-ylm2*
-4	*y(n)-ylm4*

-6	*y(n)-ylm6*

Table 12: Error computation for different values of SSD outputs.

We will adapt all the data levels (error slicer thresholds) individually and compute the data slicer thresholds from bisecting the data levels.

The adaptation gradient for the data levels are as follows:

Δylp6=sgn[err_rxffe(n)] when *yslc_rxffe(n)*=+6 (gradient=0 otherwise)
Δylp4=sgn[err_rxffe(n)] when *yslc_rxffe(n)*=+4 (gradient=0 otherwise)
Δylp2=sgn[err_rxffe(n)]) when *yslc_rxffe(n)*=+2 (gradient=0 otherwise)
*Δylp0=sgn[err_rxffe(n)]*0* when *yslc_rxffe(n)*=0 (effectively not adapted for now, removal altogether TBD)
*Δylm2=(-1)*sgn[err_rxffe(n)]* when *yslc_rxffe(n)*=-2 (gradient=0 otherwise)
Δylm4=(-1)-sgn[err_rxffe(n)]* when *yslc_rxffe(n)*=-4 (gradient=0 otherwise)
Δylm6=(-1)-sgn[err_rxffe(n)]* when *yslc_rxffe(n)*=-6 (gradient=0 otherwise)

Eqtn(6A)

The accumulated gradient is calculated in a manner similar to that for *ylp1* during FLL mode:

$$elsum\_yl[p,m]\# = \sum_{k=0}^{63} \Delta yl[p,m]\#(n)$$ Eq(6B)

Likewise the, the gmac accumulator is updated in a similar manner as well

*gmac_yl[p,m]#(n)=gmac_yl[p,m]#(n-1) + 2^(gmac_shift_yl[p,m]#)*elsum_ylpm[#]*

Eq(6C)

The same gmac structure of Figure 10 used to adapt *ylp1* can be used to adapt all data levels (error slicer thresholds). Likewise, Table 11 can be used as the gmac parameters for the adaptation of all *yl[p,m]#* and *yl0*.

The data slicer thresholds are computed from the data level (error slicer thresholds as follows):

ylp5=(ylp6 + ylp4) / 2
ylp3=(ylp4 + ylp2) / 2
ylp1=(ylp2 + yl0) / 2
ylm1=(yl0 + ylm2) / 2
ylm3=(ylm2 + ylm4) / 2
ylm5=(ylm4 + ylm6) / 2

Eq (6D)

The current expectation is that this computation to obtain the data slicer values will occur every 64T for each update of the error slicer thresholds. However, it is possibile that this computation could occur less frequently than every 64T.

8 Clock Data Recovery (CDR)

8.1 CDR Gradient Computation

At a high level we continue to use the MMSE CDR gradient for GH100. However, since our equalized signal is PAM4PR1 we have a much richer set of transitions to consider. We will be exploring what is a good minimal set of transitions of the PAM4PR1 symbol by symbol detector (SSD) symbol decisions. The initial attempt has been to explore only a very limited set of transitions all of which involve a zero crossing of the tentative PAM4PR1 SSD output. The CDR will be closed at the RXFFE output using sliced data and errors computed there. Other transition tables have been explored by various team members as well with different performance considerations for acquisition and tracking. For acquisition we desire the gradient S curve to be like a sine wave with out false lock points and for tracking we desire small steady state jitter at the sampling frequency. However, in conjunction with illegal data detection / phase kicking or csdet as known in the RTL, the simplified transition set performs the best in acquisition. This table has been commonly called the "BASES" table. Another set of transitions known as the "RTL table was chosen for tracking. In addition there is a set of "ACQ" and "TRK" tables as back up tables for acquisition and tracking. Before describing these different transition tables let us describe the generic gradient computation before the pattern filtering is applied based on these transition tables.

General Gradient Computation

Let the slope of the signal be denoted as:

$$slp(n) = [yslc\_rxffe(n)\text{-}yslc\_rxffe(n-2)] \quad \text{Eq(7)}$$

Note that *err_rxffe(n)* results in a raw multi-bit gradient based on the full RXFFE output width. However, a multi-bit gradient would require a multi-bit proportional path VCO which is not desirable. To accommodate reuse of the GA100 proportional path VCO which takes up to gradient sums of up to +/- 32 we explored some simplifications and propose the following as a stake in the ground to allow reuse of the GA100 proportional path VCO. We further quantize the error and slope signals to values *eq(n)* and *slpq(n)* as follows:

$$slpq(n) = (slp(n) \geq 0\ ?\ 1:\ -1) \qquad \text{Eq(9)}$$

Although Eq(9) technically results in a 3 level output (1, 0, -1), in reality based on the choice of transitions, the quantized slope values computed by Eq(9) will always be 1 or -1.

$$eq(n) = (err_{rxffe(n)} \geq 0\ ?\ 1:-1) \qquad \text{Eq(10)}$$

The CDR gradient for the proportional path then becomes

$$\Delta CDR\_prop(n) = eq(n-1)slpq(n) \quad \text{Eq(11)}$$

Based on this we accumulate the gradients across 32T to form the elsum:

$$elsum\_prop(n) = \sum_{k=0}^{31} \Delta CDR\_prop(n) \qquad \text{Eq(12)}$$

elsum_prop(n) maps to *ph_up[31:0]* and *ph_dn[31:0]*. This is OR'd with *ph_up[31:0]* and *ph_dn[31:0]* from the csdet/phase kick path. The output of the OR'd quantities become the final *ph_up[31:0]* and *ph_dn[31:0]* sent to the proportional path VCO.

For the integral path we have recently shown that using the same quantized gradient as the proportional path is sufficient i.e.

$$\Delta CDR\_integ(n) = \Delta CDR\_prop(n)$$

$$elsum\_integ(n) = \sum_{k=0}^{31} \Delta CDR\_integ(n) \qquad \text{Eq(13)}$$

Note that the integral path accumulator widths have NOT been changed from GA100.

A block diagram of the above computations is shown below. The diagram is for functional purposes only. The actual implementation is left up to the RTL team – for example the entire process of computing *ΔCDR_prop(n)* from *e(n)*, $yslc\_rxffe(n)$ could in principle be performed by one big look up table.

With the gradient and computations described in this section, it was possible to obtain a for Chid16 acquisition using 3000 ppm SSC with 1000ppm static offset with a subsequent reasonable jitter tolerance curve and reasonable performance with seed variations.

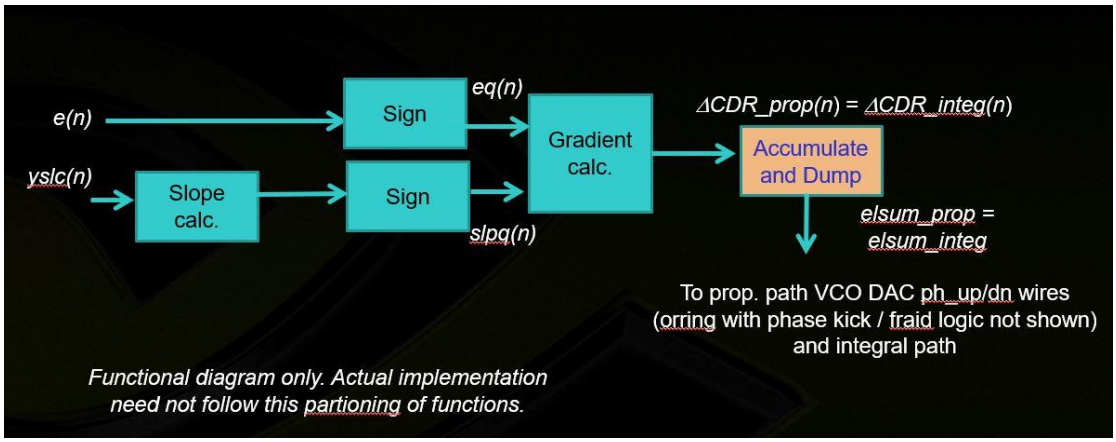


Figure 11: top level cdr sign processing and gradient accumulation. In this figure *yslc(n)=yslc_rxffe(n).*

Simplified "BASES" Pattern Filtering Table As Default Table for Acquisition Mode

We consider sets of 6 transitions of the form ($yslc\_rxffe(n-2), yslc\_rxffe(n-1), yslc\_rxffe(n)$)=(L, 0, 0), (0, 0, L), (L, 0, -L), (-L, 0, L), (-L, 0, 0), (0, 0, -L) where L can be 2, 4, or 6. We consider all the above transitions for L=2, 4, 6 i.e. we have a total of 18 transitions. In addition we cover the following transitions 12 transitions.

(L1,0,L2): L1=2,4,6, L2=-2,-4,-6, |L1| ~= |L2| (covered above already when equal)

(L1,0,L2): L1=-2,4,6, L2=2,4,6, |L1| ~= |L2| (covered above already when equal)

The total of 30 zero crossing transitions are explicitly enumerated in the table below.

yslc(n-2)	*yslc(n-1)*	*yslc(n)*	*slp(n)*

-2	0	2	4
-4	0	4	8
-6	0	6	12
2	0	-2	-4
4	0	-4	-8
6	0	-6	-12
-2	0	0	2
0	0	-2	-2
0	0	2	2
2	0	0	-2
-4	0	0	4
0	0	-4	-4
0	0	4	4
4	0	0	-4
0	0	-6	-6
-6	0	0	6
0	0	6	6
6	0	0	-6
-2	0	4	6
-2	0	6	8
2	0	-4	-6
2	0	-6	-8
-4	0	2	6
-4	0	6	10
4	0	-2	-6
4	0	-6	-10
-6	0	2	8
-6	0	4	10
6	0	-2	-8
6	0	-4	-10

Although the table shows *slp(n)* we use *slpq(n*) as noted before.

"RTL" Pattern Filtering Table As Default Table For Tracking Mode

yslc(n-2)	*yslc(n-1)*	*yslc(n)*
-6	-2	-2
-6	-2	0
-6	-2	2
-6	-2	4
-6	0	0
-6	0	2
-6	0	4
-6	0	6
-4	-2	4
-4	0	-2
-4	0	0
-4	0	2
-4	0	4
-4	0	6
-4	2	0
-4	2	2
-4	2	4
-4	2	6
-2	-2	-6
-2	-2	4
-2	0	-4
-2	0	4
-2	0	6
-2	2	6
0	-2	-6
0	-2	4
0	0	-6
0	0	-4
0	0	4
0	0	6
0	2	-4
0	2	6
2	-2	-6
2	0	-6
2	0	-4
2	0	4
2	2	-4
2	2	6
4	-2	-6
4	-2	-4

4	-2	-2
4	-2	0
4	0	-6
4	0	-4
4	0	-2
4	0	0
4	0	2
4	2	-4
6	0	-6
6	0	-4
6	0	-2
6	0	0
6	2	-4
6	2	-2
6	2	0
6	2	2

"ACQ" Pattern Filtering Table As Backup Table For Acquisition Mode

yslc(n-2)	*yslc(n-1)*	*yslc(n)*
-4	-2	2
-4	-2	0
2	-2	-4
4	-2	-6
-6	0	4
-6	0	6
-6	0	2
-4	0	4
-4	0	2
-4	0	0
-4	0	6
-2	0	4
-2	0	6
-2	0	2
0	0	-4
0	0	4
2	0	-4
2	0	-6
2	0	-2
4	0	-4
4	0	-2

4	0	0
4	0	-6
6	0	-4
6	0	-6
6	0	-2
-2	2	4
4	2	0
4	2	-2
6	2	-4

This has also been referred to the “Acq0” table in some system documentation.

“TRK” Pattern Filtering Table As Backup Table for Tracking Mode

yslc(n-2)	*yslc(n-1)*	*yslc(n)*
-4	-2	2
-4	-2	0
2	-2	-4
4	-2	-6
-6	0	4
-6	0	6
-6	0	2
-4	0	4
-4	0	2
-4	0	0
-4	0	6
-2	0	4
-2	0	6
-2	0	2
0	0	-4
0	0	4
2	0	-4
2	0	-6
2	0	-2
4	0	-4
4	0	-2
4	0	0
4	0	-6
6	0	-4
6	0	-6
6	0	-2

-2	2	4
4	2	0
4	2	-2
6	2	-4

8.2 CDR Loop Filter (Post Elsum Computation)

A high level block diagram of the CDR loop filter is shown in Figure 12. The phase detector (PD) block computes the gradient, $\Delta CDR\_prop(n)$ based on the equations and filtering tables described in the prior sections. Although conceptually the gradients can be thought of as being summed to *el_sum[6:0]* as in Eq (12), for the proportional path each of the 32T gradient terms are encoded as up / downs and control their individual proportional DCO controls. The *el_sum* quantity is actually computed and used further inside the integral path section.

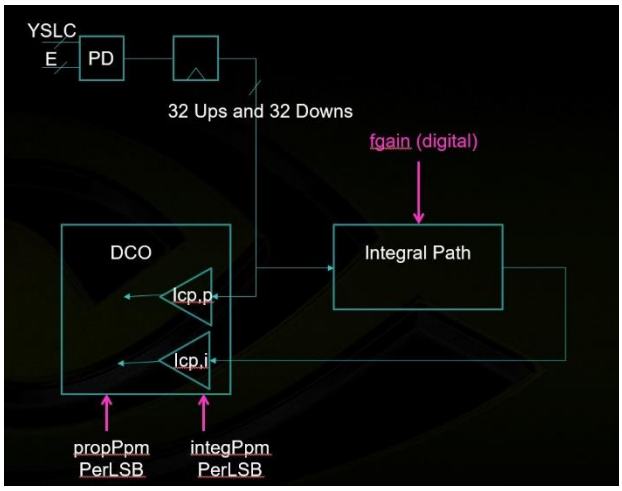


Figure 12: High level diagram of the CDR loop filter

The nominal recommended values for the *propPpm* and *integPpm* are 25ppm and 150ppm respectively and for *fgain* it is 24. Note that in the hardware there is an option to program different values for these parameters for acquisition and tracking. Details of the integral path are shown in Figure 13.

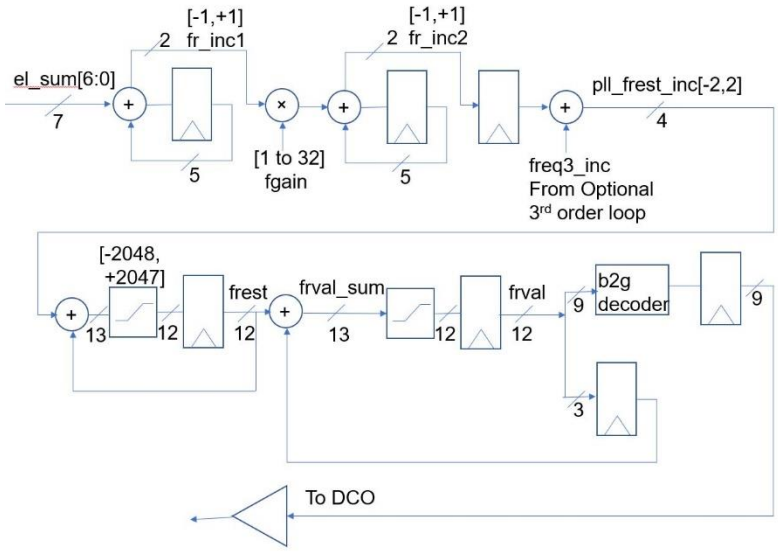


Figure 13: Integral path details

8.3 CDR Acquisition Robustness Enhancement Through Illegal Data Detect and Phase Kicking ("CSDET")

In GA100/NVHS3 we introduced to UPHY a mechanism to significantly enhance CDR acquisition robustness using a technique called illegal data detect followed by phase kicking. This is also known as "csdet" within the RTL team with "cs" representing cycle slip detection. When the CDR is not locked well and in particular when it is locked T/2 away from the correct sample phase, the slicer will produce detected data which violates the PR1 constraint. The concept is to detect the presence of such illegal data coming from the slicer and use this knowledge to "kick" the phase of the CDR to force it out of the sampling phase state which gives rise to the illegal data. The kick is an added increment to both proportional and integral path nominal el_sum computed. To decide which direction to "kick" the phase in we use the sign of the prior clock cycle's accumulated gradient i.e. the sign of the prior clock cycle's el_sum. We essentially reinforce the direction in which the phase trajectory was traveling in upon detecting illegal data.

We also kick the phase only when a certain number of illegal data detects has occurred within a given window which is typically the CDR clock cycle window size of 32T. When the number of illegal data detects exceeds a programmable threshold, T, we execute the phase kick. A high level block diagram of this architecture is shown in Figure 14.

In this architecture instead of implementing the phase kick through a sum we do so using a logical OR function. The up/downs of the main gradient path are OR'd with the phase kick value translated to the corresponding up/down values. Note that OR-ing can happen across the first *K* Up/Downs. The implementation with OR gates is equivalent to adding *K* to the sum of the gradients over 32UI. Very importantly, note that the latency in phase kick logic path does not increase main path latency. Only main path latency penalty is due to the OR gate.

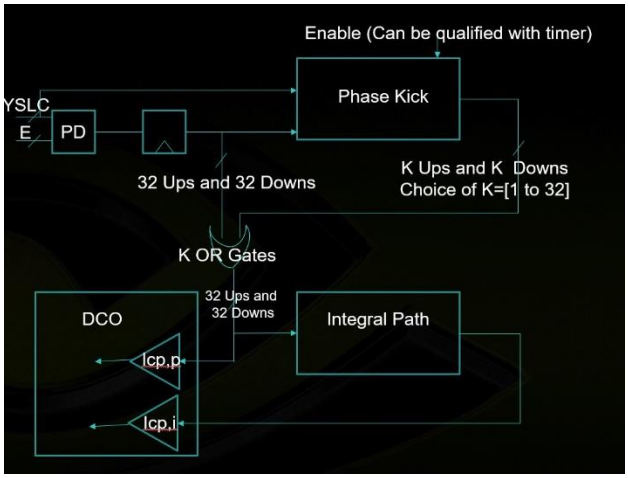


Figure 14: High level block diagram of illegal data detect and phase kicking also known as "csdet"

The illegal data detection can be implemented for both pam2pr1 as in NVHS3 as well as pam4pr1 in NVHS4. In nvhs3/ga100 with pam2pr1, employing this technique allowed the frequency acquisition range to increase from about +/- 2000ppm to +/- 10000ppm or more. For NVHS4 with pam4pr1, employing this technique allows the frequency acquisition range to be enhanced from +/- few hundred ppm (or less) to again +/- 10000ppm or more.

8.3.1 Illegal Data Detection for PAM2PR1

Here the slicer output can take on values of 2, 0, -2. A minimal condition to detect illegal data is if the slicer output of 2 is followed by -2 or -2 is followed by 2 i.e.

- *yslc(n-1)=2* and *yslc(n)=-2 OR yslc(n-1)=-2* and *yslc(n)=2*

The incorporation of the illegal detect functionality is shown in the following high level block diagram of

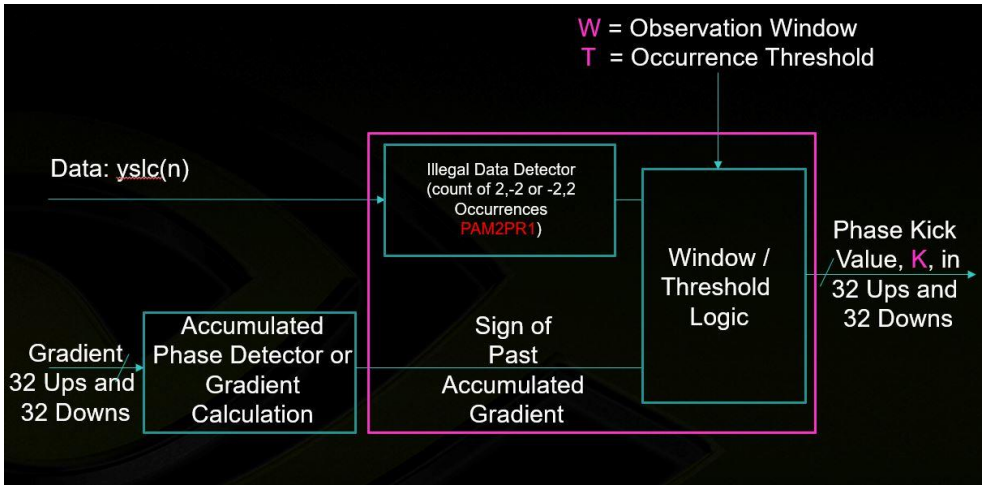


Figure 15: Illegal data detect (csdet) for pam2pr1 and how it is used for phase kicking.

Although in terms of performance we didn't see a significant improvement we implement the more expansive set of constraints as shown in the following table:

- *(yslc(n)=2, yslc(n+1)=-2) OR (yslc(n)=-2, yslc(n+1)=2)*
- *(yslc(n-1)=2, yslc(n)=0, yslc(n+1)=2)*
- *(yslc(n-1)=-2, yslc(n)=0, yslc(n+1)=-2)*

8.3.2 Illegal Data Detection for PAM4PR1

For pam4pr1 a minimal set of illegal data which violate the PAM4PR1 constraint are:

- *yslc(n-1)=-6, yslc(n) > 0* OR *yslc(n-1) > 0, yslc(n)=-6*
- *yslc(n-1)=-4, yslc(n) > +2* OR *yslc(n-1) > 2, yslc(n)=-4*
- *yslc(n-1)=-2, yslc(n)=+6* OR *yslc(n-1)=+6, yslc(n)=-2*
- *yslc(n-1)=+2, yslc(n)=-6* OR *yslc(n-1)=-6, yslc(n)=2*
- *yslc(n-1)=+4, yslc(n) < -2* OR *yslc(n-1) < -2, yslc(n)=+4*
- *yslc(n-1)=6, yslc(n) < 0* OR *yslc(n-1) < 0, yslc(n)=+6*

Using the additional set of conditions improves the performance somewhat and these are also implemented in the final RTL:

- *((yslc(n-1)=-6, yslc(n)=-6, yslc(n+1)>0) OR (yslc(n-1) =6, yslc(n)=6, yslc(n+1)<0))*
- *((yslc(n-1)=-6, yslc(n)=-4, yslc(n+1)=-6) OR (yslc(n-1) =-6, yslc(n)=-4, yslc(n+1)>2))*
- *((yslc(n-1)=6, yslc(n)=4, yslc(n+1)=6) OR (yslc(n-1) =6, yslc(n)=4, yslc(n+1)<-2))*
- *((yslc(n-1)=-6, yslc(n)=-2, yslc(n+1)<-2) OR (yslc(n-1) =-6, yslc(n)=-2, yslc(n+1)=6))*
- *((yslc(n-1)=6, yslc(n)=2, yslc(n+1)>2) OR (yslc(n-1) =6, yslc(n)=2, yslc(n+1)=-6))*
- *((yslc(n-1)=-6, yslc(n)=0, yslc(n+1)<0) OR (yslc(n-1) =6, yslc(n)=0, yslc(n+1)>0))*
- *((yslc(n-1)=-4, yslc(n)=-6, yslc(n+1)>0) OR (yslc(n-1) =4, yslc(n)=6, yslc(n+1)<0))*
- *((yslc(n-1)=-4, yslc(n)=-4, yslc(n+1)>2) OR (yslc(n-1) =4, yslc(n)=4, yslc(n+1)<-2))*
- *((yslc(n-1)=-4,yslc(n)=-2, abs(yslc(n+1))=6) OR (yslc(n-1)=4,yslc(n)=2, abs(yslc(n+1))=6))*
- *((yslc(n-1)=-4, yslc(n)=0, yslc(n+1)<-2) OR (yslc(n-1) =4, yslc(n)=0, yslc(n+1)>2))*

- *((yslc(n-1)=-4, yslc(n)=2, yslc(n+1)<0) OR (yslc(n-1) =4, yslc(n)=-2, yslc(n+1)>0))*
- *((yslc(n-1)=-2, yslc(n)=-6, yslc(n+1)>0) OR (yslc(n-1) =2, yslc(n)=6, yslc(n+1)<0))*
- *((yslc(n-1)=-2, yslc(n)=-4, yslc(n+1)>2) OR (yslc(n-1) =2, yslc(n)=4, yslc(n+1)<-2))*
- *((yslc(n-1)=-2, yslc(n)=-2, yslc(n+1)=6) OR (yslc(n-1) =2, yslc(n)=2, yslc(n+1)=-6))*
- *((yslc(n-1)=-2, yslc(n)=0, yslc(n+1)=-6) OR (yslc(n-1) =2, yslc(n)=0, yslc(n+1)=6))*
- *((yslc(n-1)=-2, yslc(n)=2, yslc(n+1)<-2) OR (yslc(n-1) =2, yslc(n)=-2, yslc(n+1)>2))*
- *((yslc(n-1)=-2, yslc(n)=4, yslc(n+1)<0) OR (yslc(n-1) =2, yslc(n)=-4, yslc(n+1)>0))*
- *((yslc(n-1)=0, yslc(n)=-6, yslc(n+1)>0) OR (yslc(n-1) =0, yslc(n)=6, yslc(n+1)<0))*
- *((yslc(n-1)=0, yslc(n)=-4, yslc(n+1)>2) OR (yslc(n-1) =0, yslc(n)=4, yslc(n+1)<-2))*
- *((yslc(n-1)=0, yslc(n)=-2, yslc(n+1)=6) OR (yslc(n-1) =0, yslc(n)=2, yslc(n+1)=-6))*

9 Phase Offset / Phase Calibration

The phase calibration (PHCAL) also known as the phase offset calibration (PHOS) loop is similar to that in GA100/NVHS3. The PHOS loops works to compensate for phase mis-match among the 8 interleaves (NVHS4 rank1 samplers). Note that the PHOS loop operates at the 64T rate and so the accumulated gradient phos_grad is a 7 bit number unlike the *el_sum* for the main CDR loop. The fundamental gradient equation is still the same as the main CDR gradient equation (MMSE) i.e. Equations 7, 9, 10 and we then have the PHOS gradient for the *l*th interleave as:

$$\Delta phos(n) = eq(n-1)slpq(n)$$

Based on this we accumulate the gradients across 32T.

$$phos\_grad\_32T(j,l) = \sum_{k=0,(k \bmod 8)=l}^{31} \Delta phos(k)$$

The gradient for each interleave will pick up 4 terms from the summation across 32T. For example, for the 0th interleave we have

$$phos\_grad\_32T(j,0) = \Delta phos(0) + \Delta phos(8) + \Delta phos(16) + \Delta phos(24)$$

The gradient at the 64T rate is obtained as:

$$phos\_grad\_64T(q,l) = phos\_grad\_32T(j,l) + phos\_grad\_32T(j-1,l)$$

where *j* refers to a 32T domain time index and where q is a 64T domain time index The final gradient using the 0th interleave as a reference is obtained as follows

$$phos\_grad(q,l) = phos\_grad\_64T(q,l) - phos\_grad\_64T(q,0)$$

The loop filter is unchanged from GA100 and is shown below in Figure 16.

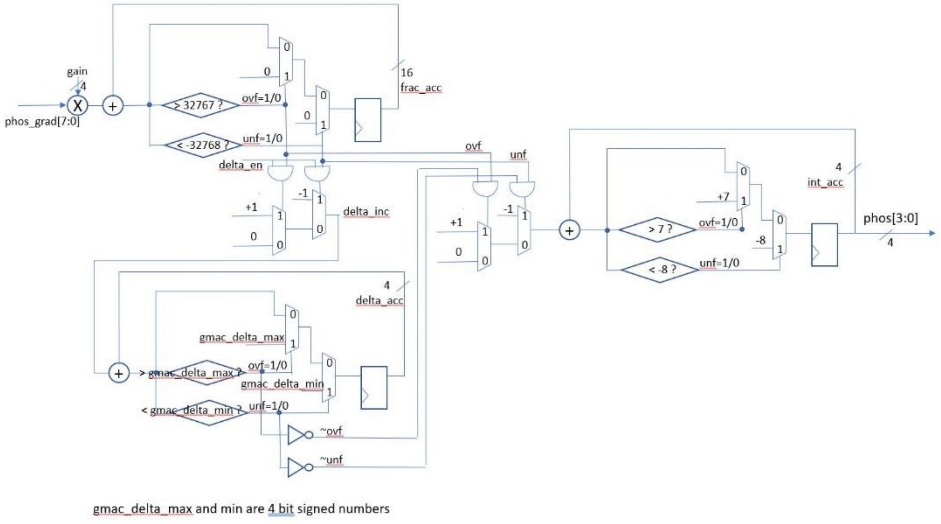


Figure 16: PHOS loop filter block diagram

The PHOS loop gradient is filtered on a different set of patterns than used for the main CDR gradient. The pattern filtering is used when calculating the gradient $\Delta phos(n)$ expression above. The table is shown below:

Pattern Filtering Table for PHOS Gradient

yslc(n-2)	*yslc(n-1)*	*yslc(n)*
6	6	0
4	2	-4
4	4	-2
4	6	0
4	6	2
2	2	-4
2	4	-2
2	4	0
2	6	0
2	0	4
0	0	-6
0	0	-4
0	2	-4
0	2	-2
0	4	-2
0	-4	2
0	-2	2
0	-2	4
0	0	4
0	0	6
-2	0	-4
-2	-6	0
-2	-4	0
-2	-4	2
-2	-2	4
-4	-6	-2
-4	-6	0
-4	-4	2
-4	-2	4
-6	-6	0

10Digital RXFFE

10.1RXFFE Overview

GH100 will implement a digital 12 tap RXFFE. Let *f(i)* denote the RXFFE tap coefficients.

The RXFFE will have 3 pre-cursor taps *f(-3)* to *f(-1)*, a main tap *f(0)*, and 8 post-cursor taps *f(1)* to *f(8)*.

For an input sample *w(n)*, the RXFFE output is

$$y(n)=w(n)f(-3)+w(n-1)f(-2)+w(n-2)f(-1)+w(n-3)f(0)+w(n-4)f(1)+w(n-5)f(2)+w(n-6)f(3)+w(n-7)f(4)+w(n-8)f(5)+ w(n-9)f(6)+w(n-10)f(7)+w(n-11)f(8)$$

A high level block diagram the RXFFE in Figure 17. The last 3 taps are not shown for simplicity. This is a full rate functional diagram only and not meant to dictate the implementation which for example could make use of pipelining between intermediate sums. Both configurations could be achieved using a 6 tap RXFFE while setting the appropriate tap to 0. However, this would not be power optimal since all delay elements would be active regardless of the configuration chosen. The mux selects in the block diagram intend to show the functionality of selecting between the two modes only, it is not meant to determine the RTL implementation which will achieve the best power savings in not using all 6 taps for each configuration. The RTL can achieve the desired configurations for example by appropriate clock gating of the appropriate delay elements.

Also the actual hardware will operate on blocks of 32 input samples. The RXFFE should be designed such that each tap has a programmable **enable / disable** control which if invoked to be disabled results in power savings through clock gating or other relevant design technique.

Moreover, for PCIE GEN5 we would like to have the option of **totally bypassing the RXFFE** (vs. simply setting non f((0) taps to zero) to save latency for the CDR loop.

The RXFFE input *x(n)* consists of a nominal 7 bit signal from the ADC output: [-64 to 63].

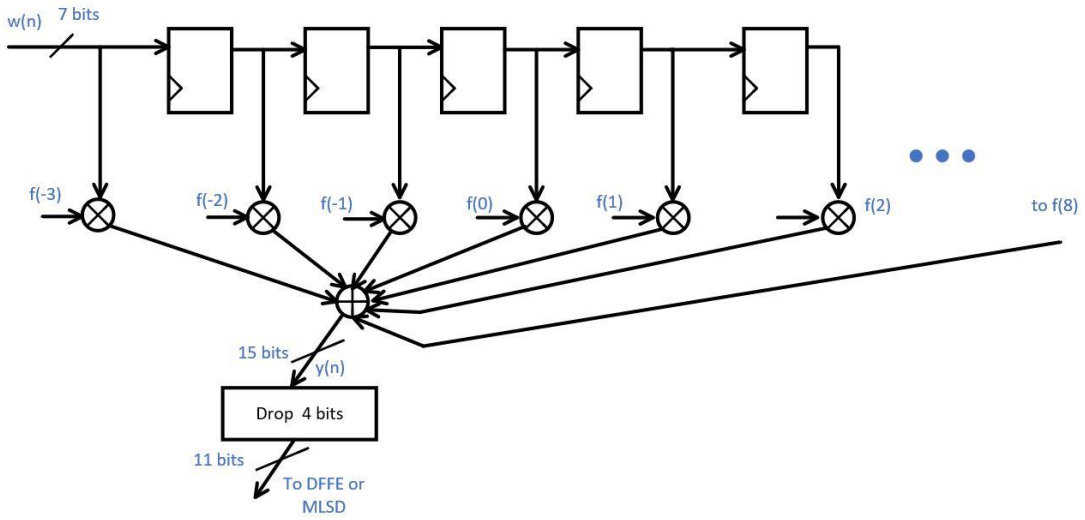


Figure 17: Behavioral full rate block diagram of the digital RXFFE. Each tap should be controlled through an enable/disable feature such that disabling the tap also results in power savings. Input bit width reduction not shown here – discussed below.

As discussed in the next section the bit width of *w(n)* multiplying with different tap coefficients can be reduced.

10.2 Digital RXFFE Input Bit Width Reduction

Although the nominal RXFFE input bit width is 7 bits, some coefficients can end up using fewer input bits to save power without any significant loss in performance. The input bit width applied to each coefficient is shown in Table 13. For each ADC sample *w(n)* let $wt_m(n)$ be the corresponding truncated sample where the *m* LSBs have been dropped based on a right shift operation (>>).

$$wt_m(n) = w(n) >> m$$

However, to preserve the linear relationship among the data/coefficient product terms we need to now scale this quantity back to the prior nominal range. Let $w_m(n)$ denote the corresponding sample obtained by the same number of bits by filling in with zeros with a left shift operation (<<).

$$w_m(n) = wt_m(n) << m$$

The value of *m* is documented for each tap in Table 13 and in the following equation which denotes the more power optimized computation of the RXFFE output.

$y(n)=w_3(n)f(-3)+w_1(n-1)f(-2)+w(n-2)f(-1)+w(n-3)f(0)+w(n-4)f(1)+w(n-5)f(2)+w_2(n-6)f(3)+w_2(n-7)f(4)+w_2(n-8)f(5)+ w_2(n-9)f(6)+w_3(n-10)f(7)+w_4(n-11)f(8)$

10.3 Digital RXFFE Tap Coefficient and Signal Scaling

We can constrain the tap coefficients in a couple of different ways – with a sum magnitude constraint or a fixed *f(0)* constraint. We have chosen to move from the sum magnitude constraint to the fixed *f(0)* constraint. With the fixed *f(0)* constraint being a power of two the multiply operation becomes a shift and smaller taps may be able to maintain more resolution with respect to the fixed *f(0)* value.

10.3.1 Fixed *f(0)* Constraint

In this framework, *f(0)* is always fixed at a pre-determined value of *f(0)=M=128*. The motivation for considering this framework is the fact that no multiplication is needed for the *f(0)* tap, only a bit shift. On a historical note, the GA100 test chip analog RXFFE was analyzed and designed within this framework. With this framework, the tap ranges are not directly mappable from the external standards TXFIR ranges where the TXFIR ranges are typically specified as % of the maximum *f(0)* value whereas here *f(0)* is fixed. However, smaller tap magnitudes can maintain higher resolution relative to the peak *f(0)* value. The set of ranges is given in Table 13.

Coefficient Range	**Input Bits Dropped** ***m***	**Input Bit Width**	**Final Input Range** $w_m(n)$	**Min** ***f(l)*** **Range**	**Max** ***f(l)*** **Range**	**Number of Bits**
		LSBs		**LSBs**	**LSBs**	
f(-3)	3	4	[-64,56]	-16	15	5
f(-2)	1	6	[-64,62]	-64	63	7
f(-1)	**0**	**7**	[-64,63]	-128	127	8
f(0)	**0**	**7**	[-64,63]	128	128	8+fxd sgn
f(1)	**0**	**7**	[-64,63]	-128	127	8
f(2)	**0**	**7**	[-64,63]	-64	63	7
f(3)	2	5	[-64,60]	-32	31	6
f(4)	2	5	[-64,60]	-32	31	6
f(5)	2	5	[-64,60]	-32	31	6
f(6)	2	5	[-64,60]	-16	15	5
f(7)	3	4	[-64,56]	-16	15	5
f(8)	4	3	[-64,48]	-8	7	4

Table 13: RXFFE tap ranges and input bit width reduction values

The tap ranges are noted both in units of LSBs. Note that the tap ranges occupy the full power of 2 ranges implied by the corresponding number of bits Although further range limitation could have benefit under difficult / marginal joint CDR/RXFFE scenarios as observed in some simulations, the full power of 2 ranges have been kept to simplify the design.

In this framework, the extreme values of the RXFFE output possible would be in the range of -58176 to 56565 which leads to a 17 bit quantity including sign. This would be theoretically possible but observed only for a pathological situation when each tap was railed and the ADC output samples were also railed with the appropriate polarity. With typical channels and good equalization settings we observe a 15 bit number i.e. upper two MSBs of the magnitude will typically be zero in the full 17 bit number. This was also discussed with an example as noted in Sec 7.2.2. The implementation should plan clipping accordingly to ensure that the RXFFE never over flows.

10.3.2 Signal Scaling and Bit Width Post-RXFFE

The equalized RXFFE signal's +6 level median will vary across different channels depending on the equalization setting and the AGC value needed to meet the ADC amplitude linearity constraints. We will learn this median as part of the AGC / +6 level adaptation process. A fixed number of bits will be dropped at the output of the RXFFE before being fed to the DFFE stage. Based on SLAN channel simulation data we expect that we will be able to drop LSBS (right shift) from the full 17 bit number to produce a **11bit RXFFE output** to drive the pam4pr1 slicer and DFFE stage.

10.4 RXFFE Adaptation

We support multiple flavors of RXFFE adaptation. In terms of the adaptation gradient we support zero forcing (ZF) gradient as well as least mean squared (LMS) gradient. We also have options to impose constraints on the adaptation to constrain how *f(1)* behaves. One set of constraints involving computing *f(1)* based on the values of several other taps. This set is called the MMPD or modified MMPD constraint. Another constraint involves not adapting or computing *f(1)* at all but simply programming it and optimizing it based on a RX performance quality metric. This is called the fixed *f(1)* constraint.

10.4.1 Zero Forcing Gradient for RXFFE Adaptation

The gradient described here is equivalent to a zero forcing solution found to perform well through a comparison with a similar complexity LMS algorithm. The gradient equation for the *l*th tap excluding *f(1)* (*l* unequal to 0, 1) is as follows:

$$\Delta f(l,n) = sgn[err\_rxffe(n-3)] * trisgn[yslc\_rxffe(n-l)] \qquad \text{Eq(14)}$$

where the *sgn* function has been defined earlier and

$$trisgn(x) = (x > 0\ ?\ 1 :\ (x <\ 0\ ?\ -1 :\ 0)) \qquad \text{Eq(14A)}$$

and *l*=0,1,2,5,6,7,8,9,10,11 for *f(-3), f(-2), f(-1), f(2), f(3), f(4), f(5), f(6), f(7), f(8)*

10.4.2 LMS Adaptation for RXFFE Adaptation

For the LMS adaptation we correlate the error with the RXFFE input to compute the gradient as follows:

$$\Delta f(l,n) = sgn[err\_rxffe(n)] * sgn[w(n-l)] \qquad \text{Eq(14B)}$$

where *w(n)* is the RXFFE input.

10.4.3 Averaged or Filtered RXFFE Adaptation Gradient

Again using the traditional GMAC structure we compute an accumulated sum of the RXFFE gradient across 64T which is then gain shifted and added to the final GMAC accumulator.

$$elsum\_f(l) = \sum_{k=0}^{63} \Delta f(l,n) \qquad \text{Eq(15)}$$

The gmac accumulator for the *l*th tap is updated as follows where here the units of *n* is in units of 64T.

$$gmac\_f(l,n) = gmac\_f(l,n-1) + 2^{\wedge}(gmac\_shift\_f(l)) * elsum\_f(l) \qquad \text{Eq(16)}$$

If viewed as 2's complement decimal numbers, there is no need for additional decimal point alignment, i.e. adding the RXFFE gear shifted gradient with gear shifted gradient LSBs aligned to the full accumulator LSBs will give the correct answer.

RXFFE Tap	**Number of Integer Bits = NI**	**Number of Fractional Bits = NF**	**gmac_shift_f(l) (Tentative)**
f(-3)	5	15	0 to 15
f(-2)	7	15	0 to 15
f(-1)	8	15	0 to 15
f(2)	7	15	0 to 15
f(3)	6	15	0 to 15
f(4)	6	15	0 to 15
f(5)	6	15	0 to 15
f(6)	5	15	0 to 15

f(7)	5	15	0 to 15
f(8)	4	15	0 to 15

Table 14: RXFFE adaptation gmac parameters using the traditional GMAC used to adapt DFE taps in prior projects.

10.4.4 MMPD Constrained f(1) Adaptation

In this scenario note that we will not adapt *f(1)* with the ZF or LMS gradient. Instead we will constrain *f(1)* to adapt per the following criterion:

$$f(1) = f(0)+f(-1)-f(2) + f1DTarg \qquad \text{Eq(17)}$$

where *f1DTarg* is a delta target value for *f(1)* away from the value which would be obtained with *f1DTarg*=0. Since the f1DTarg parameter effectively behaves like a CDR phase offset, in deference to prior PHY nomenclature we rewrite this as:

$$f(1) = f(0)+f(-1)-f(2) + cdr\_phase\_offset \qquad \text{Eq(18)}$$

where *cdr_phase_offset[7:0]* has a range of

$$-128 <= cdr\_phase\_offset <= 127$$

and is register programmable parameter. Thus every adaptation update cycle, or potentially less frequently, *f(1)* should be computed from the other taps.

Based on empirical observations of transient simulations, practical values of cdr_phase_offset are likely to fall in the range of -96 to 116 or so to obtain a linear range of sample phase variation. The exact relationship will vary from channel to channel of course. For high values of cdr_phase_offset, other RXFFE taps will tend to saturate limiting the ability of linear sample phase movement.

10.4.5 Modified MMPD Constrained f(1) Adaptation

In a modified version of the MMPD constraint we exclude the use of *f(2)* in the calculation of f(1) to obtain the following constrained equation:

$$f(1) = f(0)+f(-1)+ cdr\_phase\_offset \qquad \text{Eq(19)}$$

Note that the above constraint effectively can play the role of shifting the settled CDR sampling phase. Hence the use of the term cdr_phase_offset above.

10.4.6 Fixed f(1) Constraint

In this scenario, *f(1)* is neither adapted using the ZF or LMS gradient nor adapted to track a function of other tap values. Here *f(1)* directly plays the role of cdr_phase_offset i.e. sweeping *f(1)* effectively shifts (mostly linearly) the settled CDR sampling phase i.e.

$$f(1)=cdr\_phase\_offset \qquad \text{Eq (20)}$$

The actual useful range will vary from channel to channel. Negative values are not beneficial. For some channels although *f(1)=0* may yield a good error rate, for others *f(1)* will have to be higher than 40. Although for many channels the error rate curve flattens to a minimum with higher *f(1)*, for some channels there may be an eventual modest increase in the error rate at the highest *f(1)* values.

11 Digital Feed Forward Equalization (DFFE)

11.1 DFFE Conceptual Overview

The basic idea behind DFFE is the iterative use of tentative decisions to improve the accuracy of ISI estimation. The quality of tentative decisions and hence the compensation of ISI, is expected to get better and better with each iteration/stage. Improved symbol estimates reduce the probability of mis-correction in the next stage and gives better symbol estimates than the previous stage. DFFE (with enough iterations/stages) can potentially achieve the same level of performance as DFE with much less complexity. The number of DFFE stages required depends very heavily on the reliability of the initial tentative decisions passed on to the first DFFE stage.

The following is for general understanding of the DFFE concept only. They do not reflect implementation details.

Detected symbol provided by DFFE at any iteration, i, is given by

$$\hat{x}(n)^{(i)} = Q\left(y(n) - \sum_{j=1}^{N_h} h_j(n)\hat{x}(n-j)^{(i-1)}\right)$$

Where 1 <= i < R, and R is the total number of iterations, and Q is the slicer function. In this example, only post-cursors (including h_1) are used for ISI cancellation from the input, $y(n)$. Note that the same input is used for cancelling ISI in all stages/iterations. A block diagram with 3 post cursor taps DFFE with R = 4 (Number of Iterations) is shown below in Figure 18. Precursor cancellation is not done here.

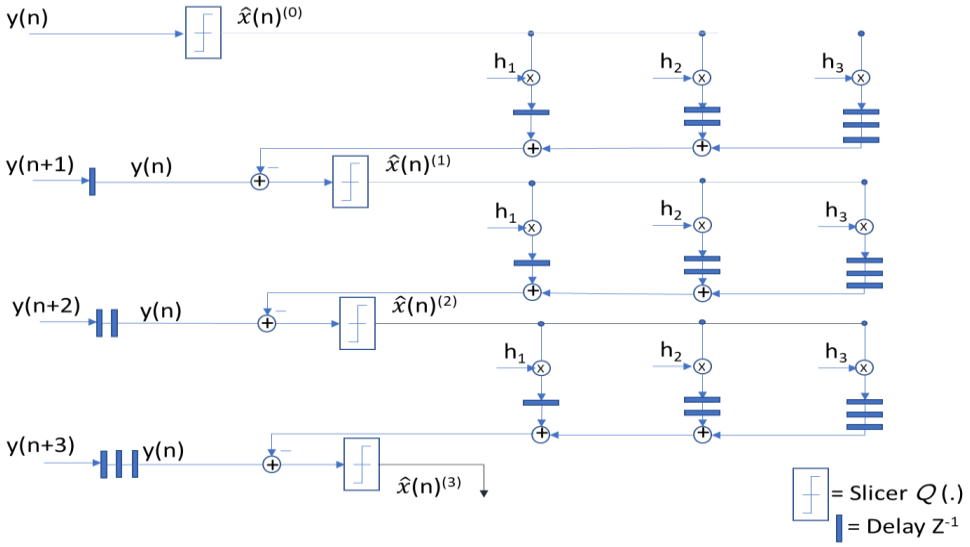


Figure 18: DFFE block diagram for a PR0 equalized system

A block diagram with 2 pre cursor and 3 post cursor taps DFFE cancellation with R = 4 (number of iterations) is shown below in Figure 19.

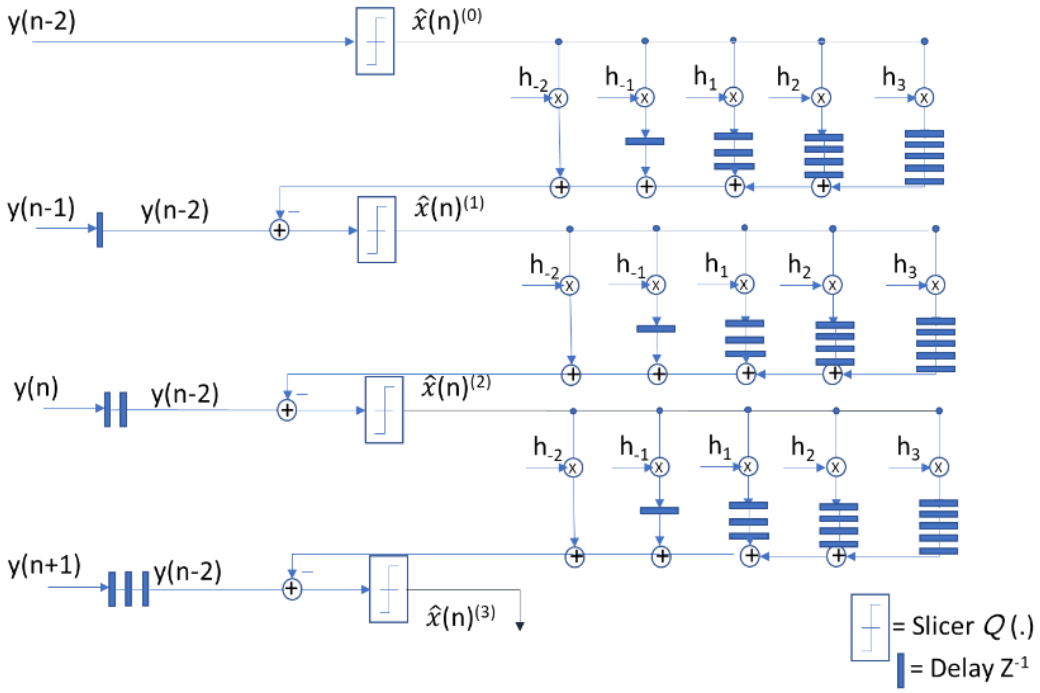


Figure 19: DFFE block diagram for a PR0 equalized system including pre-cursor cancellation.

Finally, a block Diagram with 2 pre cursor and 3 post cursor taps DFFE cancellation with R = 4 (number of iterations) for a PR1 system is shown in Figure 20 below. It can be noted that ISI from h_1 is not cancelled.

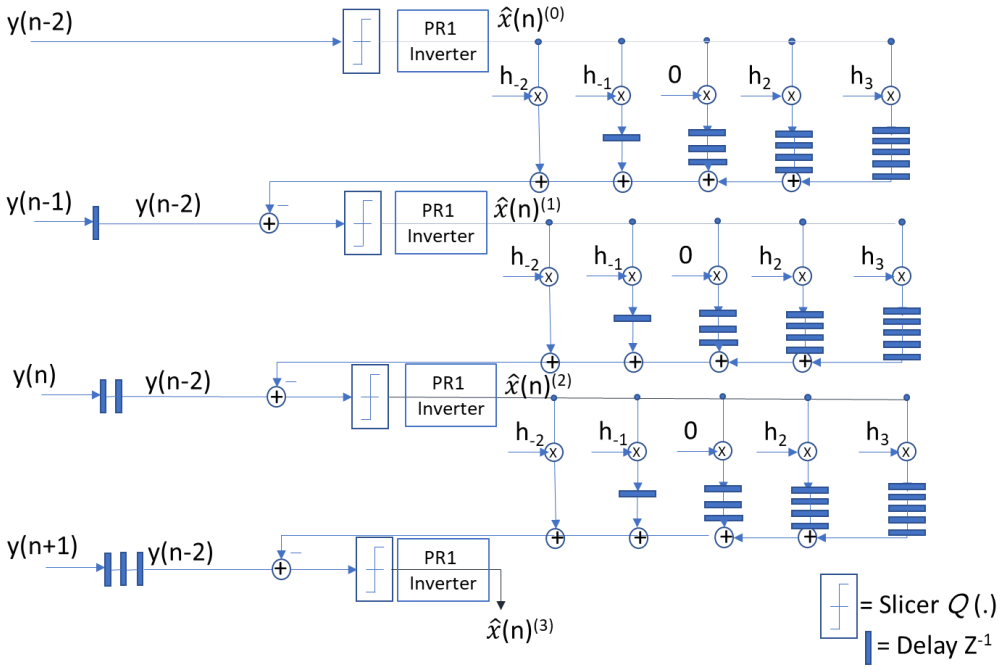


Figure 20: DFFE block diagram for a PR1 equalized system

11.2 DFFE Implementation Details

GH100 will use DFFE after rxFFE to improve the quality of ISI cancelled samples going into the MLSD. Although the prior exemplary diagrams incorporate the use of pre-cursor cancellation, it has been decided to remove this ability from the final implementation based on performance vs. power tradeoffs. Since GH100 uses PAM4 PR1 equalization and the ISI is added on PAM4 signals, a PR1 inverter needs to be there inside the DFFE. Unlike traditional DFFE, the GH100 DFFE with PR1 inverter will cause some error propagation and simulations show that much of the ISI correction and symbol error rate improvements happens in the first stage of DFFE, and improvements from more stages are not significant. Although the initial plan was to have a two stage DFFE, the final design choice is to have one stage of DFFE based on performance/power considerations. A detailed GH100 DFFE implementation block diagram is shown below in Figure 21.

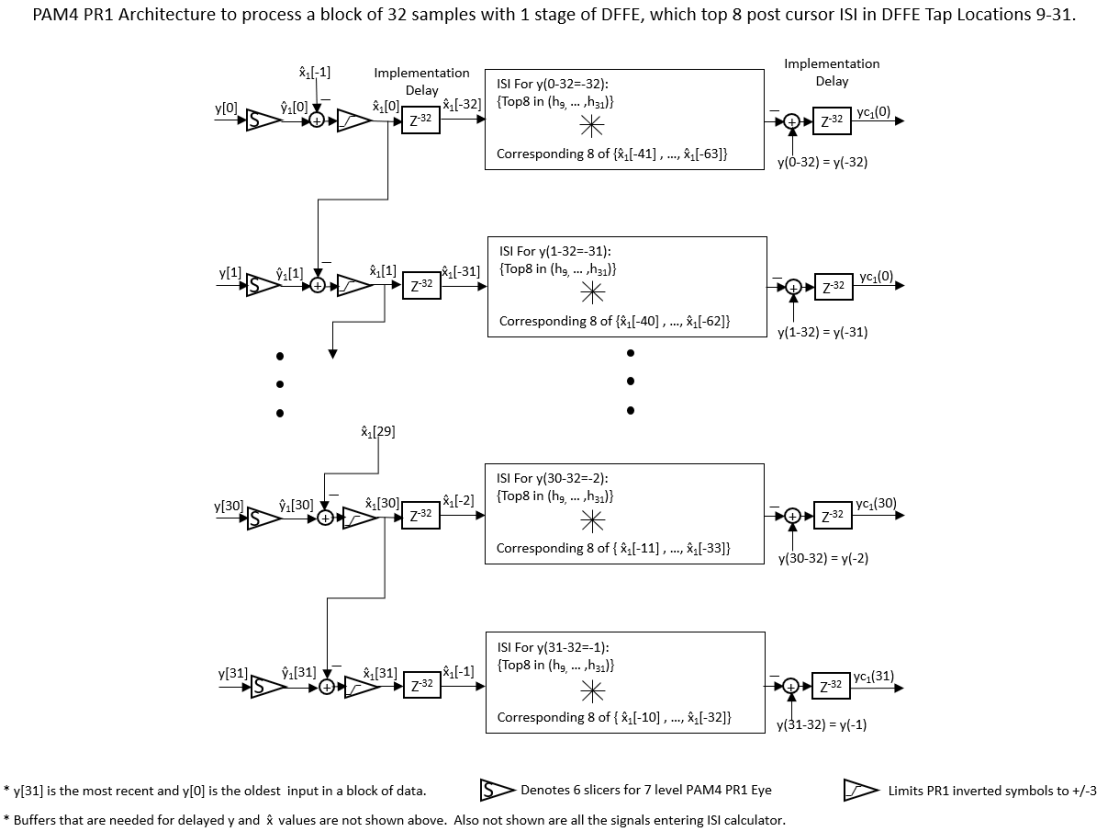


Figure 21: Detailed implementation block diagram for PR1 equalized DFFE

The input to the first stage DFFE, y[], could either be the ADC output or the rxFFE output. Since the bit widths of ADC output and rxFFE output are different, the ADC output going into the first stage DFFE must be padded with enough zeros in the LSBs so that the bit widths of the arithmetic inside the DFFE can be kept the same for both cases.

The bit widths of different signals pertaining to DFFE are given below in Table 15. Please note that the bit widths proposed below are tentative and subject to change as more fixed point simulations are yet to be done. Note that if treated as 2s compliment arithmetic operations, the

LSBs of the ISI terms xhat(n)*h(n) terms in the DFFE will already be aligned to the LSBs of the RXFFE output based on the proper adaptation of these taps.

Signal Name	**Bit Width**
rxFFE output going into DFFE, y[]	Keep 11 bits
DFFE ~~Pre~~-Post Coefficients, ~~{h$_{-2}$,h$_{-1}$},~~ Top 8-post Coefficients in {h_9,h_{10},…h_{32}}	4 bits (-8to +7)
DFFE output, yc[], going into MLSD	Keep 11 bits

Table 15: DFFE fixed point parameters

Also note that yc_1[], the clean signal from first stage DFFE ~~going into the second stage DFFE~~, should also be kept at 11 bits before final slicing.

An alternate implementation of the PR1 inverter is shown below in Figure 22.

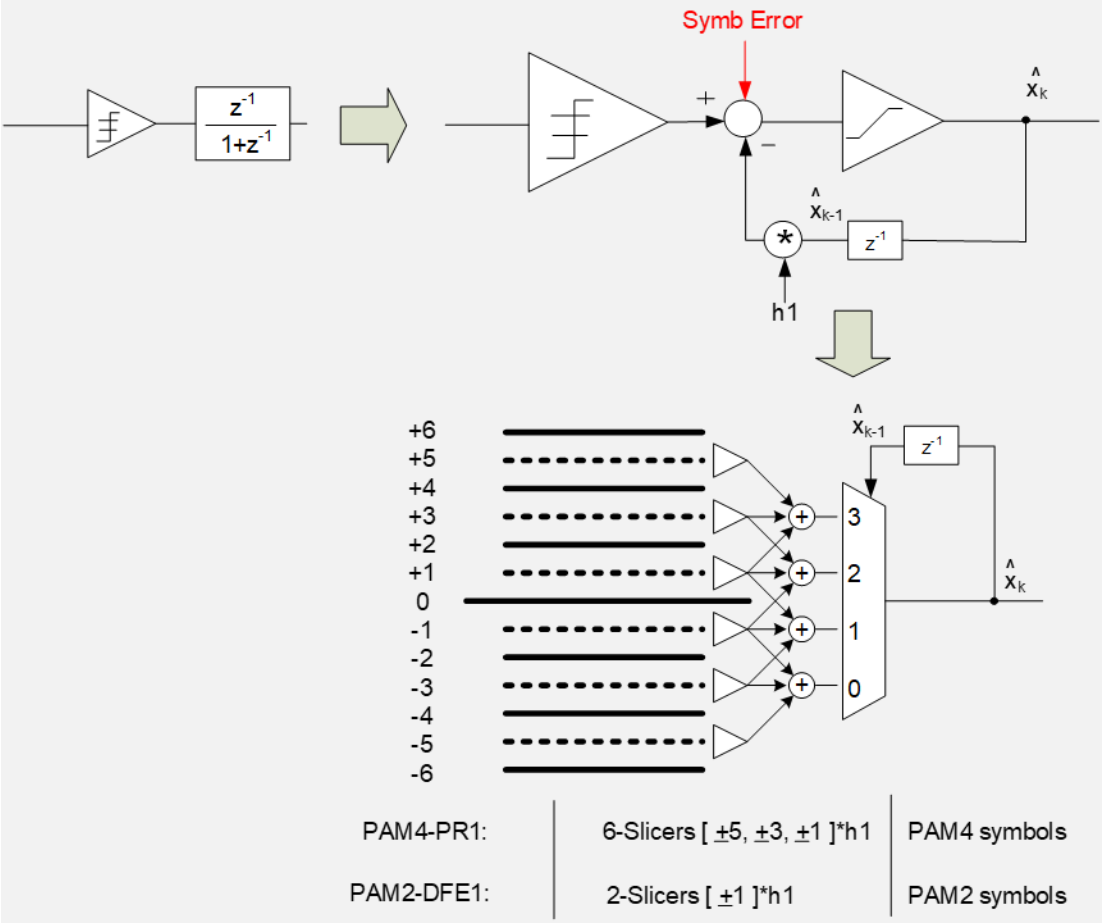


Figure 22: Alternate implementation of a PR1 inverter

11.3 DFFE Input Tap Selection / Location

The coverage range of DFFE is from post-9 to post-32 taps respective to main-tap, where such long coverage range has capability to cover reflection induced by 45mm PKG; however, to consider HW implementation and performance trade-off, 8 floating taps with programmable location is implemented. Normally, small PKG will result in near reflection, and large PKG will result in far reflection, and the location to PKG trace length could be calculated based on $K = \frac{2 \times L \times baud\_rate}{c}$, where L is the PKG trace length, c is the speed of electricity (1.5×10^8).

PKG length	Reflection location
12mm	9UI
24mm	17UI
32mm	23UI
40mm	28UI
45mm	32UI

To have the capability to select 8 taps from 24 possible taps location, a MUX selection scheme is designed. The input and coefficient location selection scheme is described below

DFFE post-tap cover range	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	
DFFE floating MUX1	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	4:1 MUX
DFFE floating MUX2	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	4:1 MUX
DFFE floating MUX3	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	4:1 MUX
DFFE floating MUX4	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	4:1 MUX
DFFE floating MUX5	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	4:1 MUX
DFFE floating MUX6	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	8:1 Mux
DFFE floating MUX7	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	8:1 Mux
DFFE floating MUX8	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	8:1 Mux

Each row of this table represents the connection of each MUX to tap location, and tap location is show on each column. MUX1~5 will have only 4 possible selections and MUX6~8 will have 8 possible selections. Each tap is covered by two MUXs to increase to coverage flexibility, and the MUX combination is diverse to avoid confliction.

The selection of MUX connection is done by a greedy algorithm that we start to distribute MUX from strongest tap to weakest tap, and the algorithm will distribute tap on small number MUX to simplify the procedure. Since there is limitation on the tap to MUX distribution, algorithm might left few taps in top-8 list not distributed, and we will use those spare MUXs to cover strong taps which is not in top-8 list. Here is an example

CHID	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
5(numbers stand for magnitude sorting)	2	7	6	3		4	5		1				8											
Regular interleaved 5 + interleaved 3																								
DFFE post-tap cover range	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
DFFE floating MUX1	2																							
DFFE floating MUX2							5																	
DFFE floating MUX3			6																					
DFFE floating MUX4									1															
DFFE floating MUX5																								
DFFE floating MUX6				3																				
DFFE floating MUX7		7																						
DFFE floating MUX8						4																		
missing reflection													8											

Step1: start on tap17 (1st strongest): from MUX4 and MUX8, we distribute on MUX4

Step2: start on tap9 (2nd strongest): from MUX1 and MUX6, we distribute on MUX1

Step3: start on tap12 (3rd strongest): from MUX4 and MUX6, since MUX4 is taken by tap17 so we distribute on MUX6

Step4: start on tap14 (4th strongest): from MUX1 and MUX8, since MUX1 is taken by tap9 so we distribute on MUX8

Step5: start on tap15 (5th strongest): from MUX2 and MUX6, we distribute on MUX2

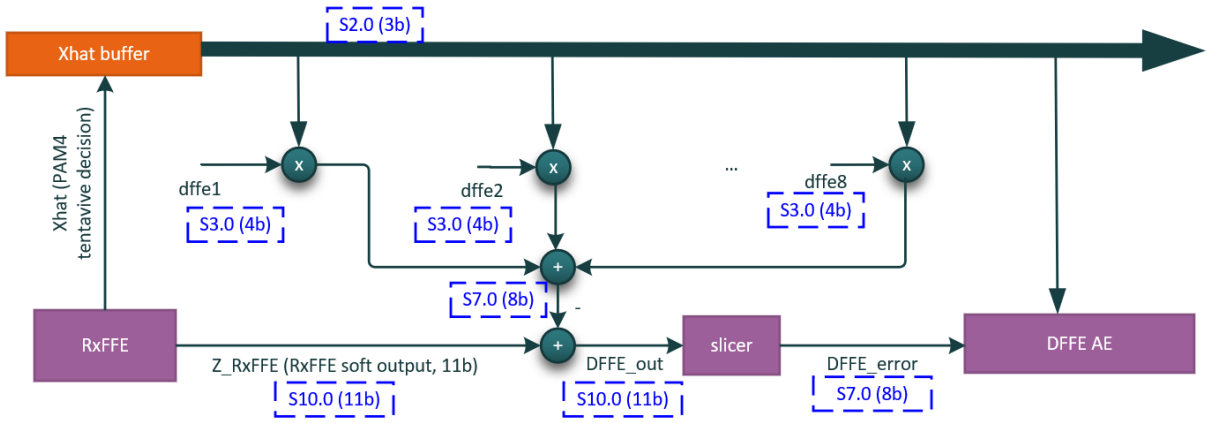
Step6: start on tap11 (6th strongest): from MUX3 and MUX8, we distribute on MUX3

Step7: start on tap10 (7th strongest): from MUX2 and MUX7, since MUX2 is taken by tap15 so we distribute on MUX7

Step8: start on tap21 (8th strongest): from MUX3 and MUX6, but MUX3 is taken by tap11 and MUX6 is taken by tap12, so we do not cover this tap

Next step: start from 9th to 24th to see which one is the strongest to be able to distribute on MUX5

11.4 DFFE ISI Reconstruction and Cancellation



DFFE re-constructs ISI by using tentative decision from RxFFE. DFFE use the re-constructed ISI to improve the signal quality of RxFFE output soft signal. It could be expressed in follow in equation

$$ISI[n] = \sum_{i=0}^{7} dffe[i]xhat[n - idx[i]]$$

$$DFFE_{out}[n] = RxFFE_{out}[n] - ISI[n]$$

, where idx[i] follows the connection determined by greedy algorithm from MUX connection table, and n is time index and i is tap index. A demonstration of time index of RxFFE output and tentative decision data is shown as below

RxFFE out (z)	z1	z2	z3	z4	z5	z6	z7	z8	z9	z10	z11	z12	z13	z14	z15	z16	z17	z18	z19	z20	z21	z22	z23	z24	z25	z26	z27	z28	z29	z30	z31	z32
RxFFE slice (yhat)	y1	y2	y3	y4	y5	y6	y7	y8	y9	y10	y11	y12	y13	y14	y15	y16	y17	y18	y19	y20	y21	y22	y23	y24	y25	y26	y27	y28	y29	y30	y31	y32
RxFFE PR1inv (xhat)	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15	x16	x17	x18	x19	x20	x21	x22	x23	x24	x25	x26	x27	x28	x29	x30	x31	x32
DFFE input at 32																																z32
DFFE ISI					h8		h7					h6	h5	h4	h3				h2	h1												
DFFE out																																dz32 = z32 - (h1*x20 + h2*x19 + h3*x15 + h4*x14 + h5*x13 + h6*x12 + h7*x7 + h8*x5)
DFFE input at 31																															z31	
DFFE ISI					h8		h7					h6	h5	h4	h3				h2	h1												
DFFE out																																dz31 = z31 - (h1*x19 + h2*x18 + h3*x14 + h4*x13 + h5*x12 + h6*x11 + h7*x6 + h8*x4)
DFFE input at 30																														z30		
DFFE ISI			h8		h7					h6	h5	h4	h3				h2	h1														
DFFE out																																dz30 = z30 - (h1*x18 + h2*x17 + h3*x13 + h4*x12 + h5*x11 + h6*x10 + h7*x5 + h8*x3)

Idx of the example above is 12, 13, 17, 18, 19, 20, 25, 27.

11.5 DFFE Tap Value Adaptation

We will adapt the DFFE using an LMS type of algorithm which use DFFE filter input and slice error for adaptation gradient.

The gradient equation for the ith tap excluding dffe(i) (i unequal to 0 to 7) is as follows:

$$grdout = \sum_{j=0}^{63} sign(\hat{x}\,(n - j - idx(i)))sign(e_{dffe}(n - j))$$

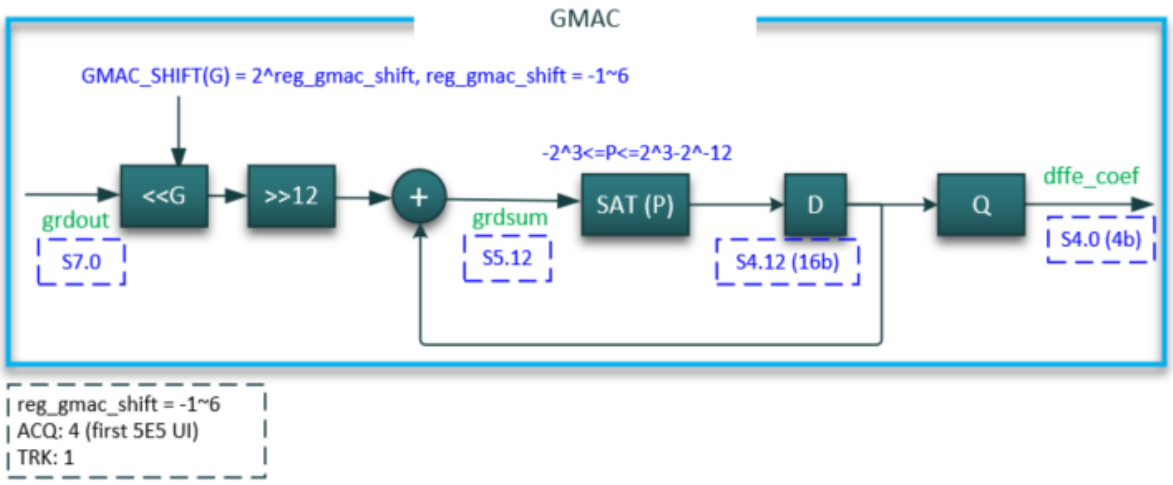
$$grdsum(i,k) = grdsum(i,k-1) + 2^{GMAC\_SHIFT} * grdout$$

The 2-level sign function is defined as

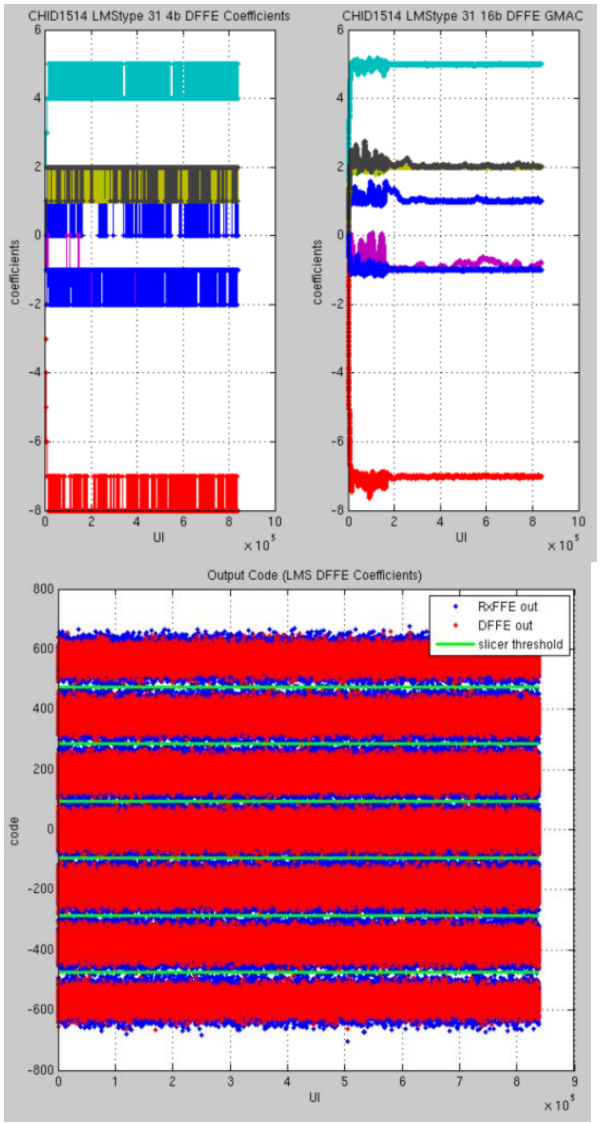
$$\text{sign(x)}=\begin{cases} 1, & x \geq 0 \\ -1, & x < 0 \end{cases}$$

The $e_{dffe}(n)$ is the error slicer of DFFE, n is time index and idx(i) shows the input connection of ith tap. Since the conventional GMAC structure is adopted, the GMAC_SHIFT controls the bandwidth of this adaptation loop, and the default settings will be 2^6 for acquisition and 2^3 for tracking, where acquisition takes 5E5 UI.

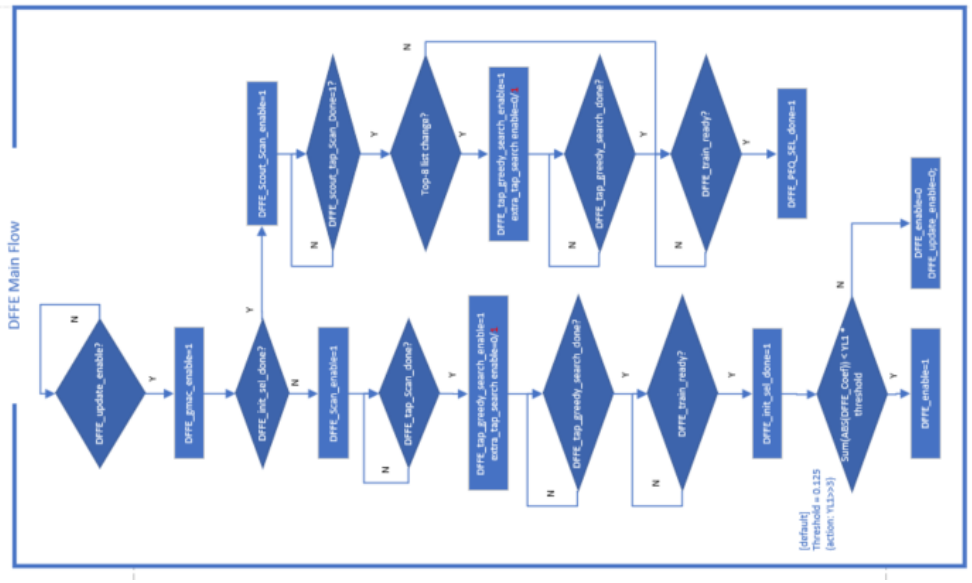
The DFFE coefficients that used to do convolution has fixed point value range 4b (-8~7), so we need to further take shift and quantization (floor) to obtain 4b DFFE coefficients from 16 gradsum buffer.



An example of transient behavior of DFFE coefficients is shown as below.

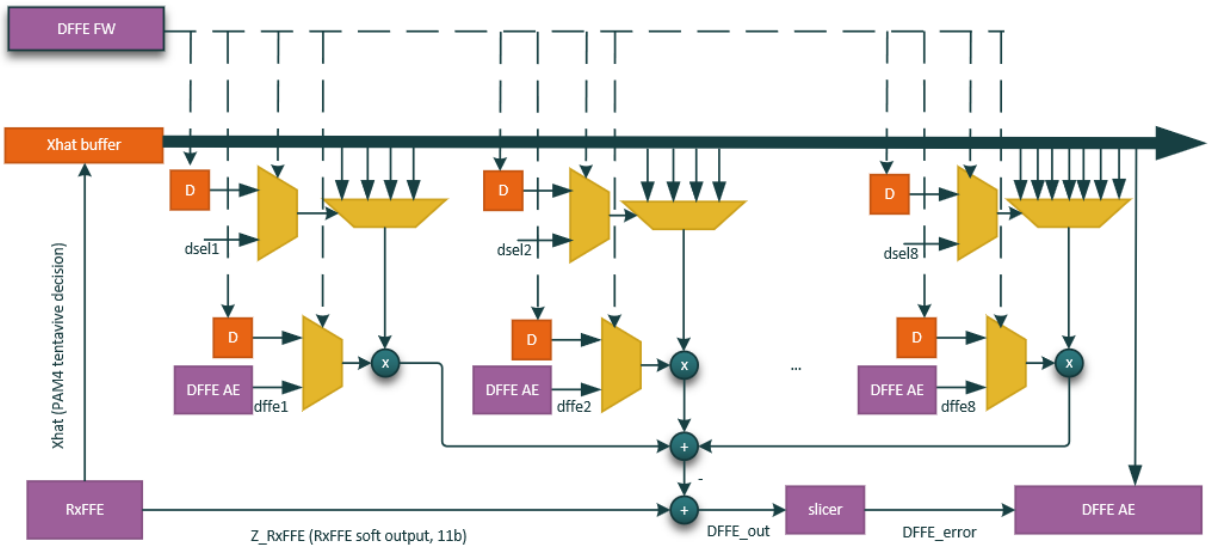


11.6 DFFE Firmware Flow

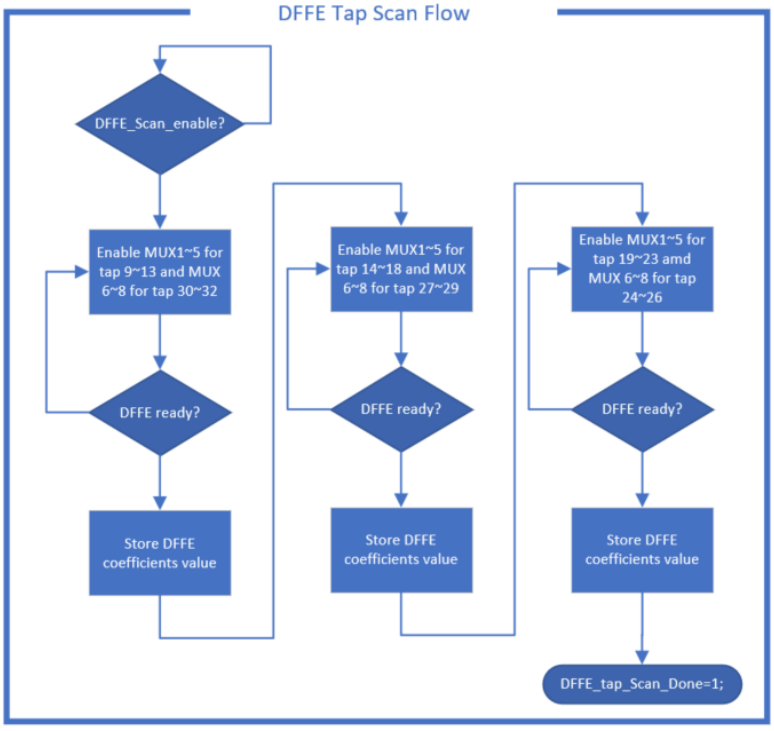


DFFE FW will first do tap scan at beginning of being enabled. After tap9 to tap32 are scanned, the greedy algorithm will distribute MUX according to sorted tap magnitude. After taps are selected, we will determine to enable or disable based on the normalized L1-norm of DFFE coefficients, where DFFE is disabled if the normalized L1-norm is smaller than YL1*threshold (currently, threshold is 0.125, which can be implemented as simple right shift; however, this should be programmable with simple operation). This action is employed to save power on some small reflection channels that we have very minor performance gain or even performance loss by enable DFFE. In periodic EQ, DFFE will use MUX9 to scan post9 to post32 often to see if top-8 strong reflections are changed or not. If the top-8 is changed, the MUX selection should be re-distributed by greedy algorithm.

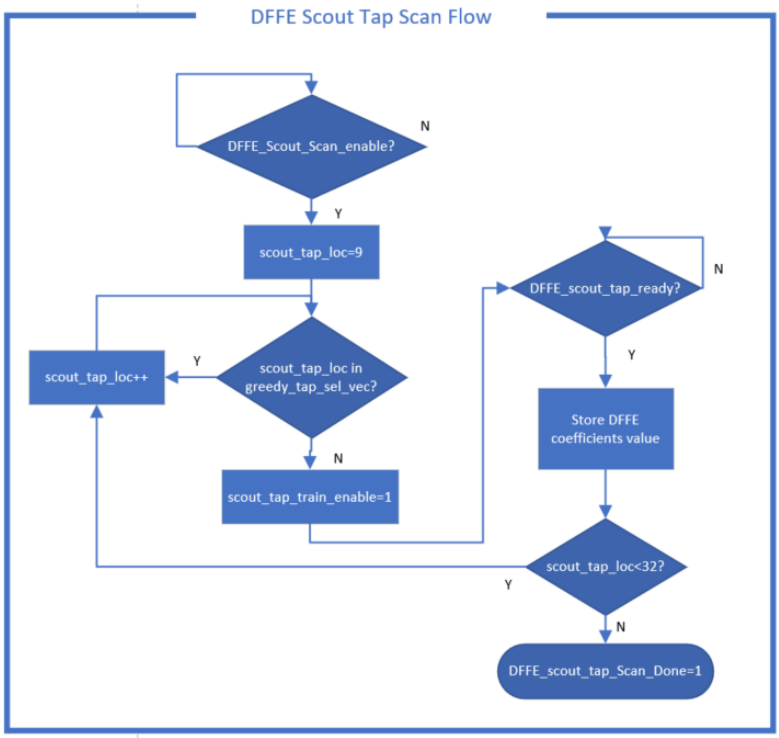
In order to have MUX switching simultaneously without glitching, the HW implementation is shown as below to switch input and DFFE coefficients



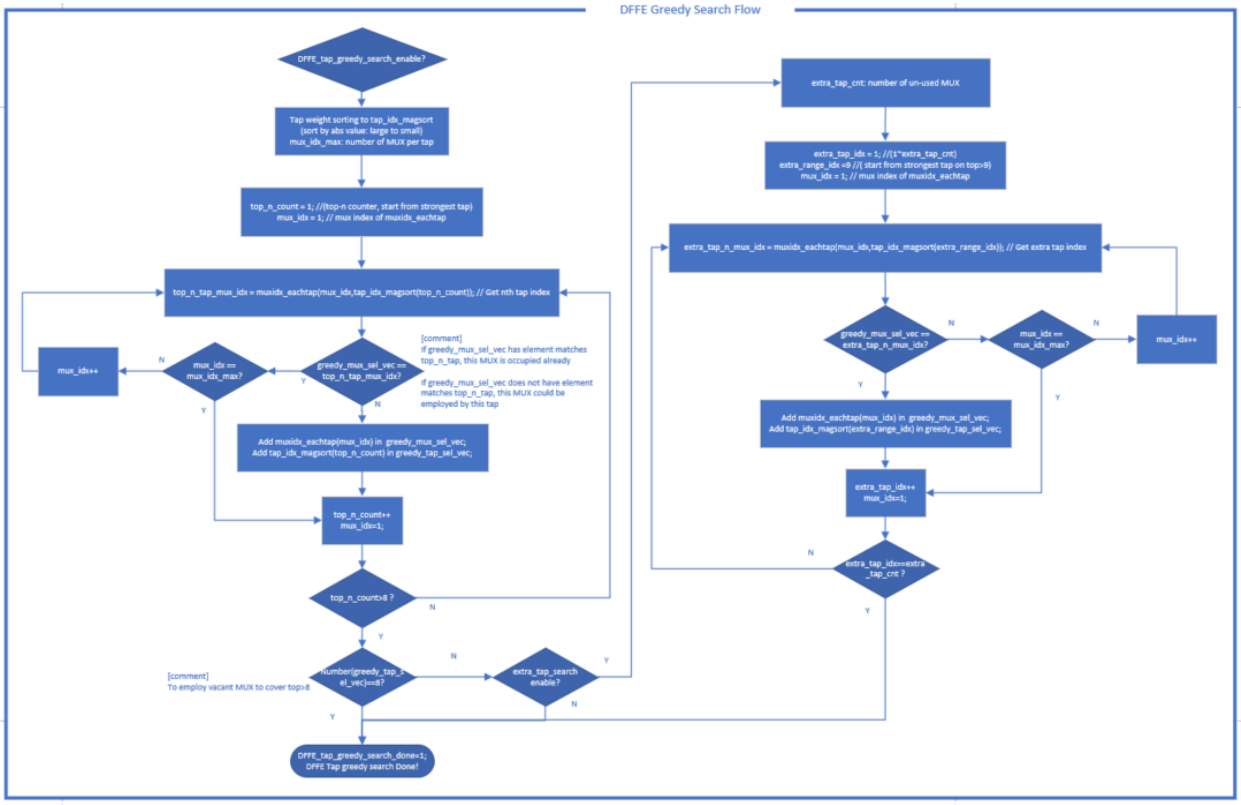
The tap scan procedure of initial link-up is described below.



The tap scan procedure of periodic EQ is described below.



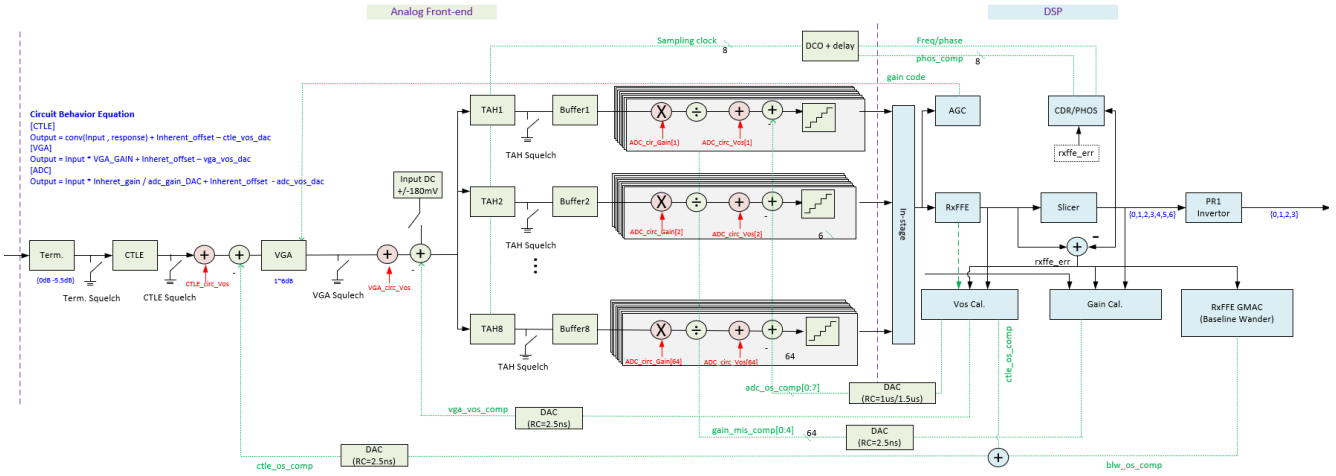
The greedy search algorithm for MUX distribution is described below.



12 Rx Calibration

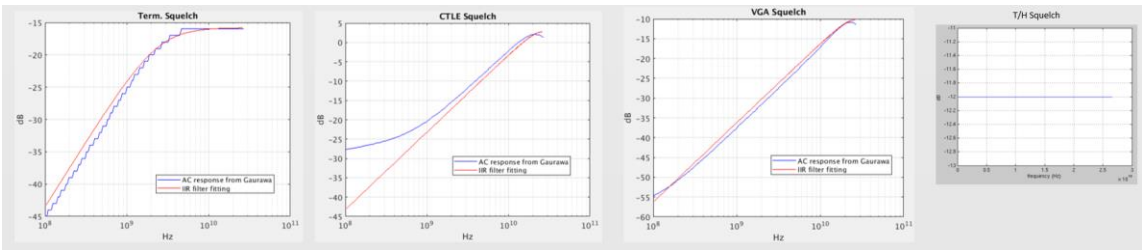
12.1 Analog Front-end and Calibration Overview

The analog front-end of GH100 includes R-termination, continuous time linear equalizer (CTLE), voltage gain amplified (VGA), and 8 track-and-hold (T/H) followed by 8 buffer and 8 7b time-interleaved ADC (TI-ADC). There are total 64 TI-ADCs, denoted ADC0, ADC2,…,ADC63, and conversion rate of each ADC is 830MHz for 106Gbps protocol. Sampling period between two samples from the same ADC is 64UI. For example, consecutive samples from ADC0 are 64 UI apart in time. The detail analog front-end block diagram is shown below



Ideally, ADC samples the analog signal uniformly in time and provides digital output with the thermal noise and quantization noise effect. However, in a real-world ADC, impairments such as voltage offset error, gain error and clock-skew also impact system performance. There are offset error at CTLE and VGA as well. Different power mode of CTLE and gears of CTLE will have different offset error, and different gears of VGA will also have different offset error.

To cope with offset error, gain error, and clock-skew, CTLE Vos, VGA Vos, ADC Vos, ADC Gain, PHOS compensation circuits are implemented. There is build-in DC signal control for gain error calibration to deal with residual Vos while doing ADC gain calibration. There are squelch circuits at R-term, CTLE, VGA, and T/H to mute signal from channel for startup calibration; however, the squelch is not perfect, and there will be signal leaked as noise affecting Rx calibration. The AC responses of squelch circuits are shown below:



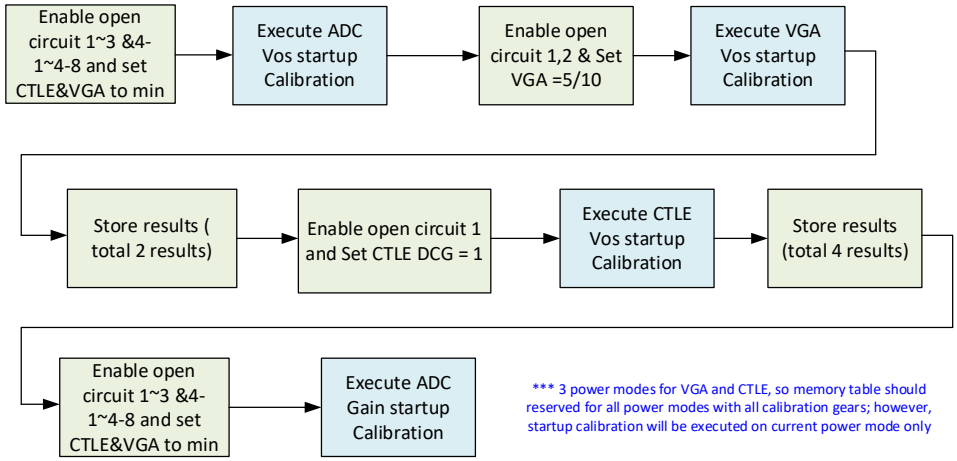
The related design spec. and circuit impairments are listed in the table below:

Circuit Info.	Impairment Value		Compensation Spec.			
	Range	Temp. drift	# of bit	Step size	Comp. Range	INL/DNL
ADC Vos	σ = 5mV Range: +/-30mV	14.3/80^{o}C	8	0.7mV	+/-40mV	
VGA Vos	σ = 4.5mV Range: +/-27mV		6	1.5mV	+/-32mV	
CTLE Vos	σ = 4mV Range: +/-24mV		6	1.0mV	+/-32mV	
Gain	σ = 0.5% Range: +/- 3%	+/-0.4%/80^{o}C	5	0.27%	+/-4%	
PHOS	σ = 0.2ps Range: 1ps		5	0.2ps	+/-3ps	

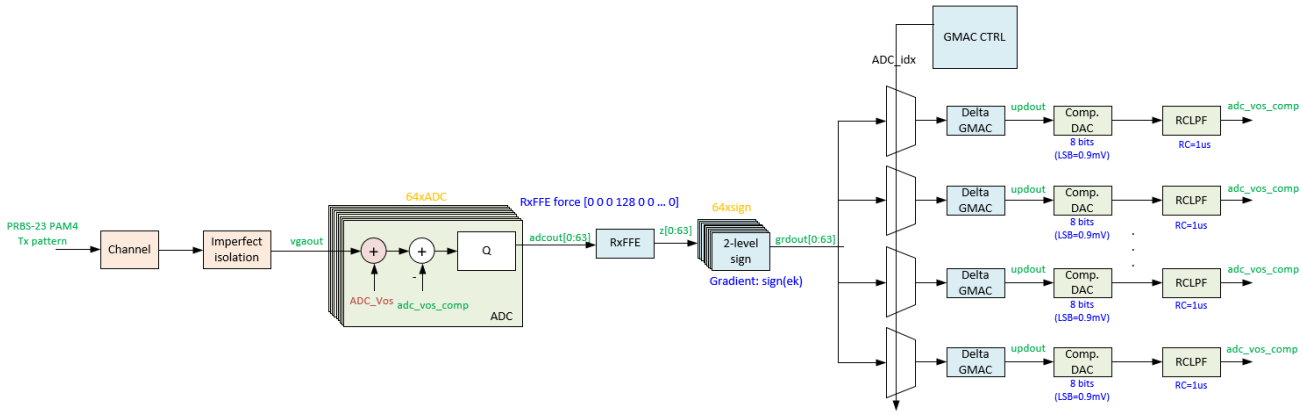
There are two stages of Rx calibration, startup mode and continuous mode. Startup mode will be executed before EQ/CDR. Continuous mode will be executed with CDR and EQ adaptation jointly to keep signal quality by reducing offset error, gain error, and clock-skew. Baseline wander effect will also be captured and compensated by using CTLE Vos compensation in continuous mode.

12.2 Rx Calibration Startup mode Sequence

The calibration sequence starts from Vos, ADC Vos, VGA Vos, and then CTLE Vos, and gain error calibration. The Vos calibration will be executed on TI-ADC with muting the line at ADC input by R-term, CTLE, VGA, and T/H squelch. Vos calibration on VGA Vos will be executed with muting the line at VGA input by R-term and CTLE squelch. Lastly, Vos calibration on CTLE Vos is then executed with muting the line at CTLE input by R-term squelch only. There are 2 gain gears settings for VGA Vos calibration, and there are 1 DGC settings for CTLE Vos calibration, where CTLE Vos calibration will be executed on current power mode only. Finally, ADC gain error calibration will be executed with muting the line at VGA output by R-term, CTLE, and VGA squelch. Meanwhile, the built-in DC signal should inject toggling DC signal for unbiased gain error estimation and compensation, where the toggling rate should be at us(micro second) order due to RC settling time around 2.5ns (complete settling <10ns). All the above calibration results should be stored in memory that we could use them as initial value for continuous mode or changing gears. The startup calibration sequence is shown as below:



12.3 Startup ADC Offset Calibration



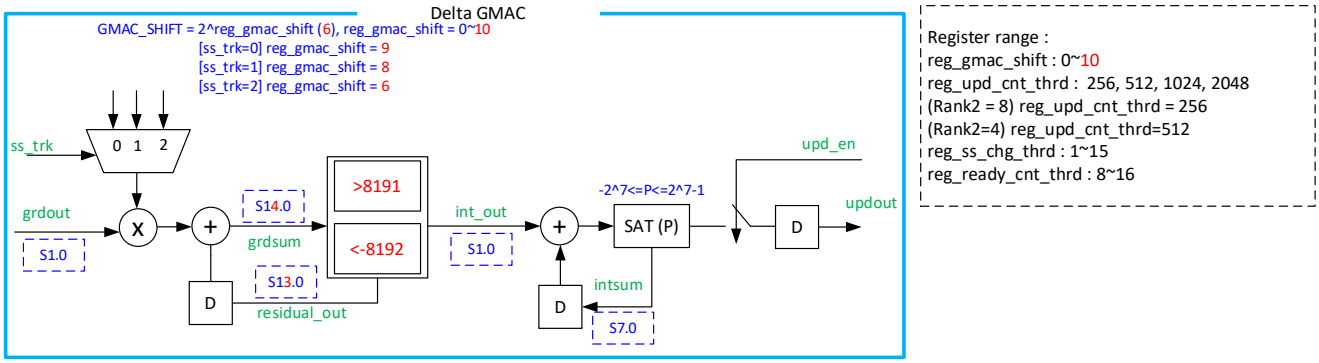
The offsets present in the individual time interleaved ADCs is estimated and cancelled by muting the line at ADC input by the squelch circuits at R-term., CTLE, VGA and TAH. The mean of the ADC samples in each of interleave ADC represents its offset voltage. Those mean ADC LSBs for each of the 64 interleaves will be converted to analog offset correction voltages using DACs and will be cancelled from the 64 time-interleaved ADC circuits. This concept can be implemented using the delta GMAC. There are fractional part and integer part in delta GMAC that once the accumulated value reaches threshold in fractional part, integer part will increase/decrease one LSB. The gradients for offset correction for the 64 T/Hs (i=0, 1, 2, …63) are given by with the following update equation for startup mode.

$$grdsum\ [I,n+1] = grdsum\ [I,n] + 2^{gmac\_shift}\text{sign(adc[I,n])}$$

The 2-level sign function is defined as

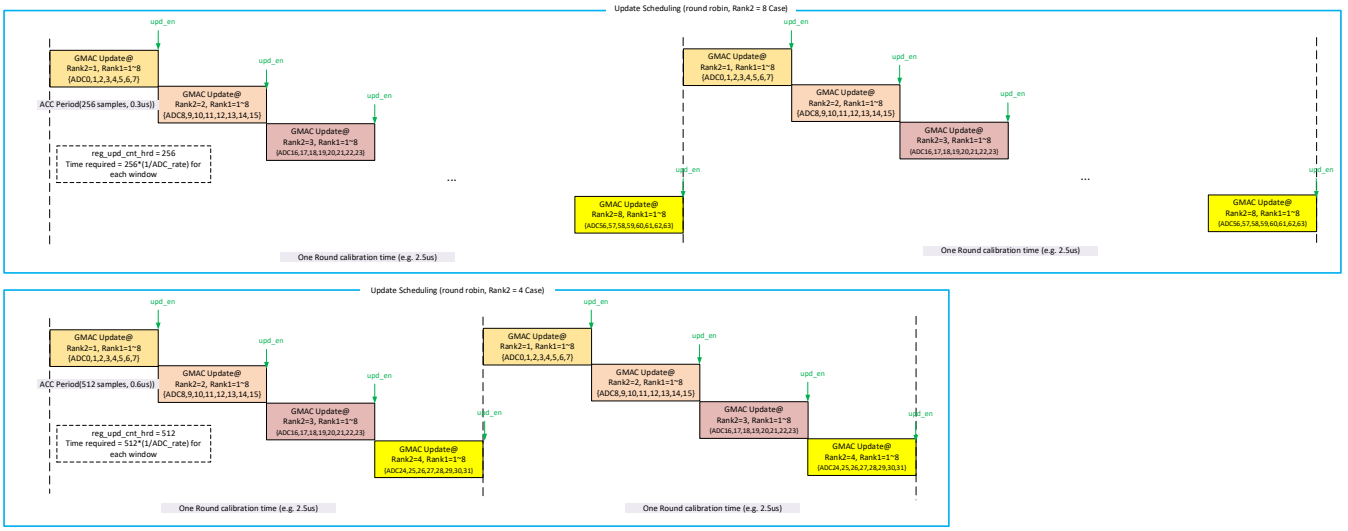
$$\text{sign(x)}=\begin{cases} 1, & x \geq 0 \\ -1, & x < 0 \end{cases}$$

As long as grdsum[I,n+1] touches the fractional threshold +8191/-8192, it will increase/decrease one digital code to integer loop. The integer loop will send out compensation code to compensation DAC with 0.7mV step for each 64 TI-ADC. The GMAC_SHIFT of fractional loop can be varied for purpose of fast acquisition and stable tracking. There are default values proposed and possible value range attached

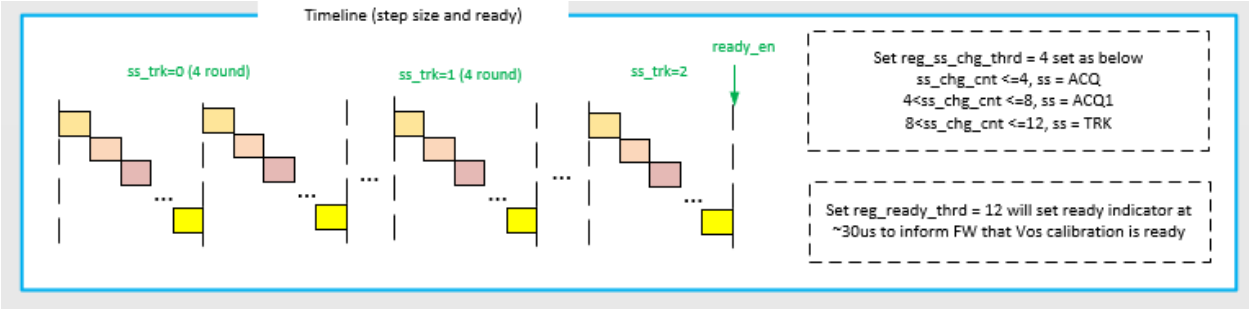


There are RC effect (~1us@startup and continuous mode) on ADC Vos DAC, so the update period of one TI-ADC should not be too fast; otherwise, there will be damping due to not enough settling time. So, round robin update is adopted to avoid damping. There are two scenarios, Rank2 is 4 and 8, which is the number of update set. To have fast convergence and small damping, we choose 2.5us (90% settling of 1us RC) update period as trade-off. For 8 GMAC scenario, each set will accumulate 256 samples and have one update opportunity while 512 samples for 4 GMAC scenario. The update period is number of samples per set multiply with number of update set. For example, 0.3us period per update window by doing accumulating 256 samples on each TI-ADC, and 8 update set due to Rank2 number is 8. Thus, total calibration time for one round is 2.5us, which will be 90% settling of 1us RC time constant.

The control for upd_en, ADC update index, and number of samples accumulated are shown below for both Rank2 is 4 and 8 cases.



The variable step size is adopted to have fast acquisition at beginning and good stability at end. There is no convergence detection mechanism for startup calibration, but there is timeout counter for ready_en indicator instead. As long as ready_en sent to top level FW, top level will know that startup ADC Vos is done, and we are ready to go for VGA Vos.



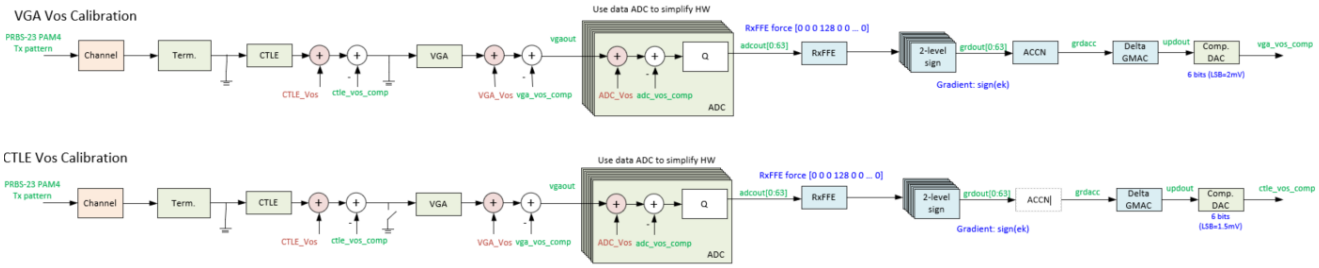
12.4 Startup CTLE/VGA Offset Calibration

The startup offsets present in the CTLE/VGA is estimated and cancelled by muting the line at CTLE/VGA input by the squelch circuits at R-term and CTLE. The mean of data TI-ADCs will be used for estimation and cancellation on CTLE/VGA Vos. Those mean ADC LSBs for CTLE/VGA Vos will be converted to analog offset correction voltages using DACs and will be cancelled in VGA circuits.

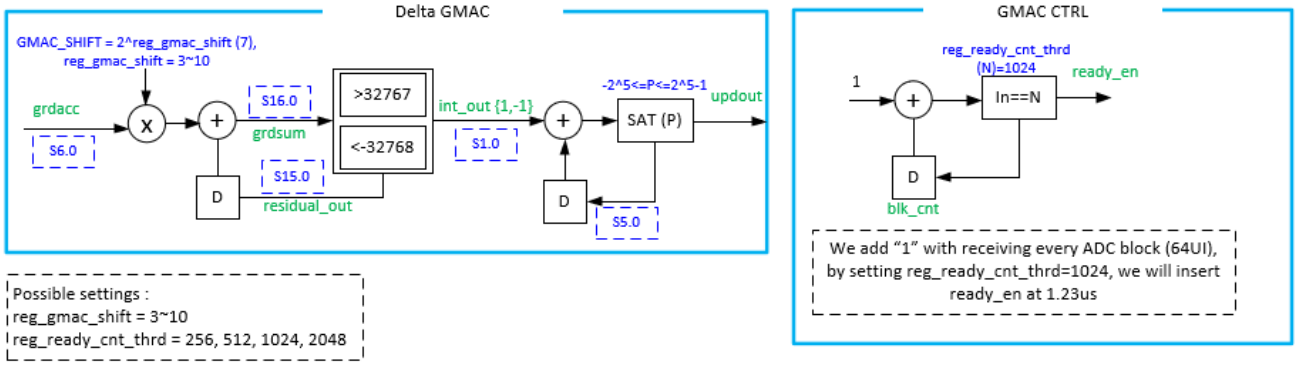
The gradient calculation for Vos calibration is defined as

$$grdsum\,[n+1] = grdsum\,[n] + 2^{GMAC\_SHIFT} \sum_{i=1}^{N} sign(adc[i,n]),$$

whereN is number of data ADC, i stands for i-th path of TI-ADCs at time instance n.

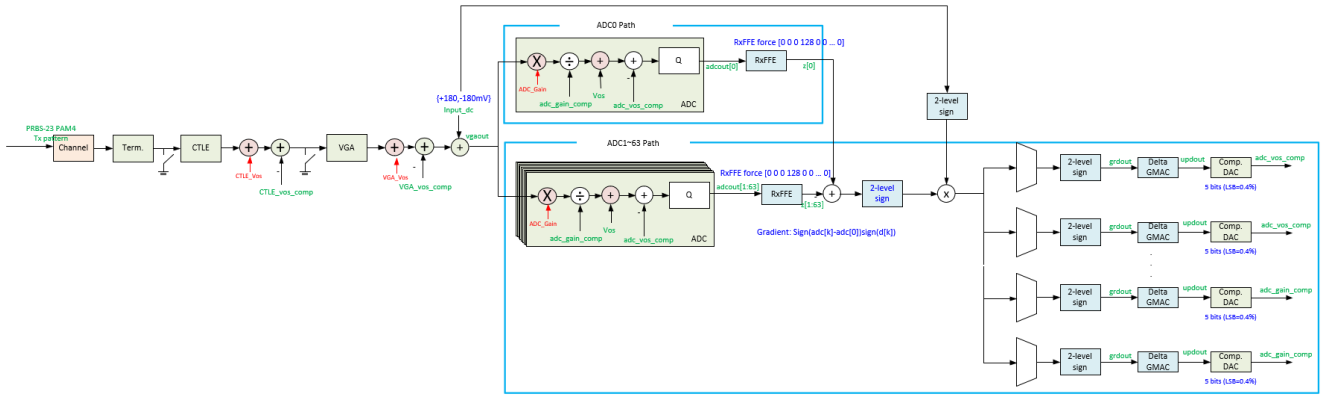


Since there is no GMAC sharing due to there is only one CTLE/VGA Vos and the RC effect of CTLE/VGA DAC is very small, so there is no update control for CTLE/VGA Vos. It simply accumulates the gradients from one Rx operation block (different speed will have different number of samples per operation block) in fractional loop, and it increases/decreases one code to integer loop for compensation DAC update.



There is no convergence detection mechanism for startup calibration, but there is timeout counter for ready_en indicator instead. As long as ready_en sent to top level FW, top level will know that startup CTLE/VGA Vos is done, and we are ready to go for ADC Gain Offset.

12.5 Startup ADC Gain Offset Calibration



The startup Gain offsets present in the individual time interleaved ADCs is estimated and cancelled by injecting a pre-defined DC signal while muting the line at ADC input by the squelch circuits at R-term., CTLE and VGA. The pre-defined DC signal should be large enough to estimate the gain mismatch, and the polarity should be

toggled to have unbiased estimation under residual ADC Vos from startup calibration. Instead of estimating ADC mean, the gain difference must be estimated and cancelled referring to one target, which is ADC0. Those gain difference of ADC LSBs for 1~63 ADCs will be converted to ADC reference voltage and will be cancelled from the 64 ADC circuits.

In BBPD-CDR case, data of edge ADC come from even number ADC, which is ADC[0], ADC[2], …, ADC[62]; and, data of data ADC come from odd number ADC, which is ADC[1], ADC[3], …, ADC[63]. Since the edge samples will have value close to "zero", and small magnitude is affected by gain mismatch less, so ADC gain calibration is skipped for edge ADC for BBPD-CDR case.

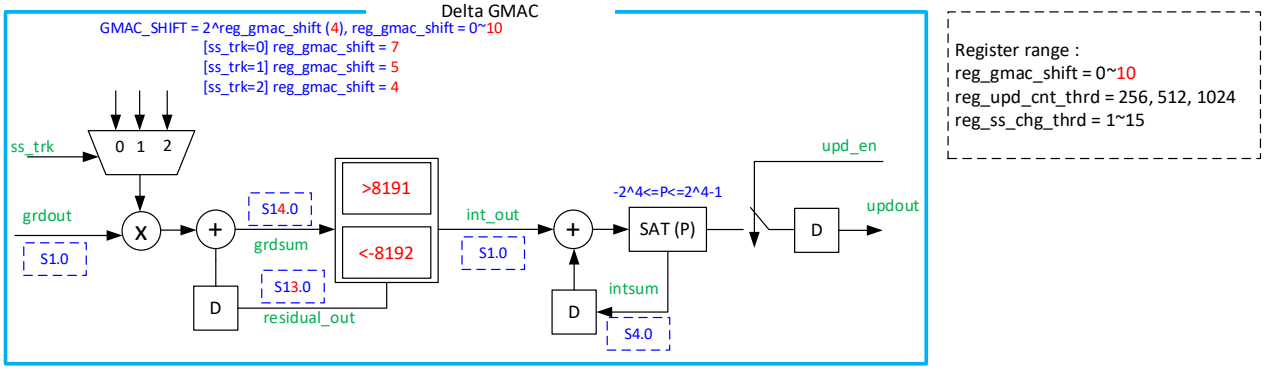
In addition, for lower rate protocols, the ADC resolution of 5bits or 4bits are 17.1875mV and 34.375mV respectively. The signal difference of 2% gain mismatch is much smaller than ADC resolution. The performance impact due to ADC gain mismatch might be minor under 5b or 4b ADC quantization noise, so ADC gain calibration is optional for lower rate protocols

This concept can be implemented using the delta GMAC. There are fractional part and integer part in delta GMAC that once the accumulated value reaches threshold in fractional part, integer part will increase/decrease one LSB. The gradient for GOS correction for the 64 T/Hs (i=0, 1, 2, …63) are given by with the following update equation for startup mode.

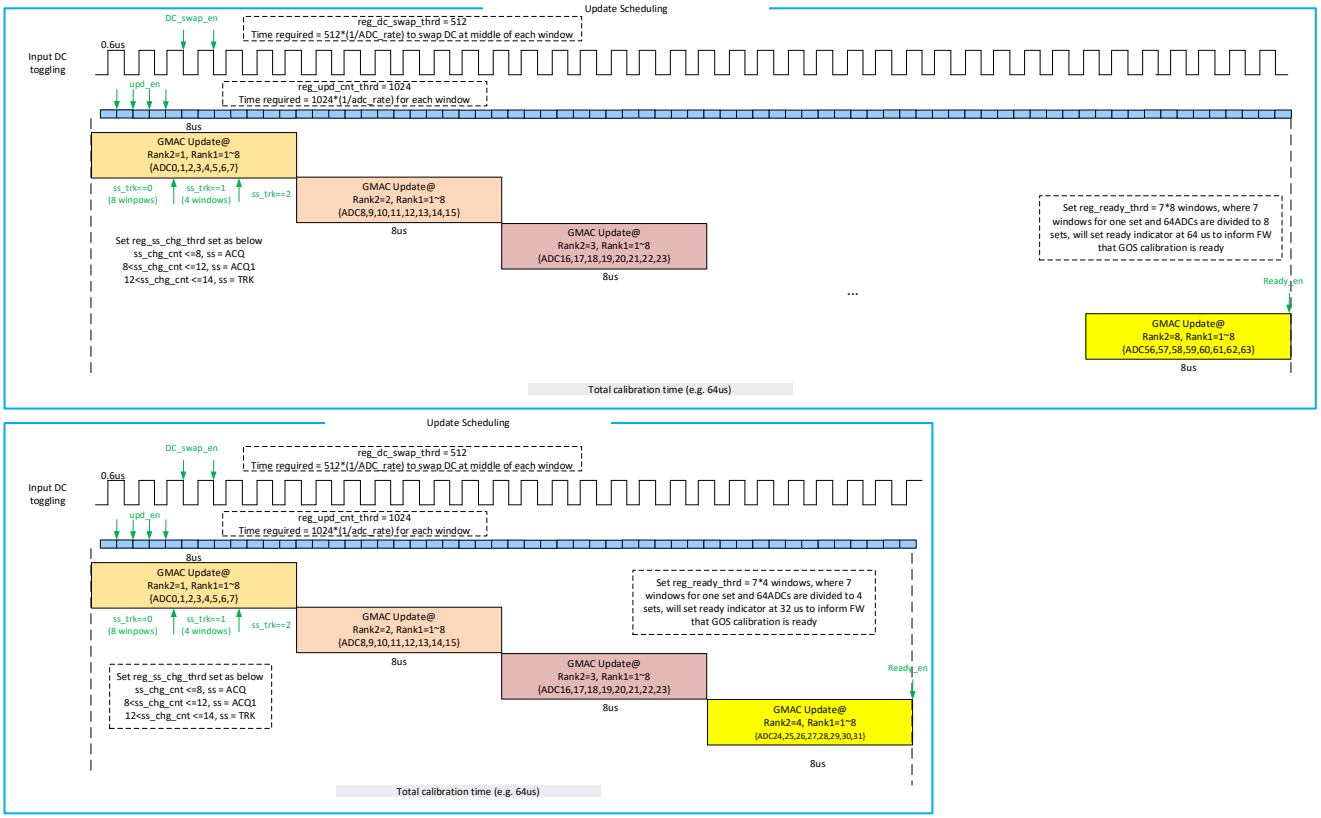
$$grdsum\ [i,n+1] = grdsum\ [i,n] + 2^{GMAC\_SHIFT}\text{sign}(adc[i,n]-adc[0,n])$$

The definition of sign function is the same as startup ADC Vos calibration.

As long as grdsum[I,n] touches the fractional threshold +8191/-8192, it will increase/decrease one digital code to integer loop. The integer loop will send out compensation code to compensation Gain C-DAC with 0.27% step for each 64 TI-ADC. The GMAC_SHIFT of fractional loop can be varied for purpose of fast acquisition and stable tracking. There are default values proposed and possible value range attached



There is no fractional memory of delta GMAC, so the update scheme cannot be round robin. Although all ADCs are sharing 8 GMAC, but each GMAC will calibrate entirely on one ADC and switch to another ADC. There are two scenarios of update scheme, which are number of rank2 is 4 and 8. The control for upd_en, ADC_idx, and variable step size are shown below. There is no convergence detection mechanism for startup calibration, but there is timeout counter for ready_en indicator instead. As long as ready_en sent to top level FW, top level will know that startup ADC Vos is done, and we are ready to go for VGA Vos.



To reduce the compensation range, the value of startup gain calibration on each ADC will have a common shift with the mean value of gain calibration results.

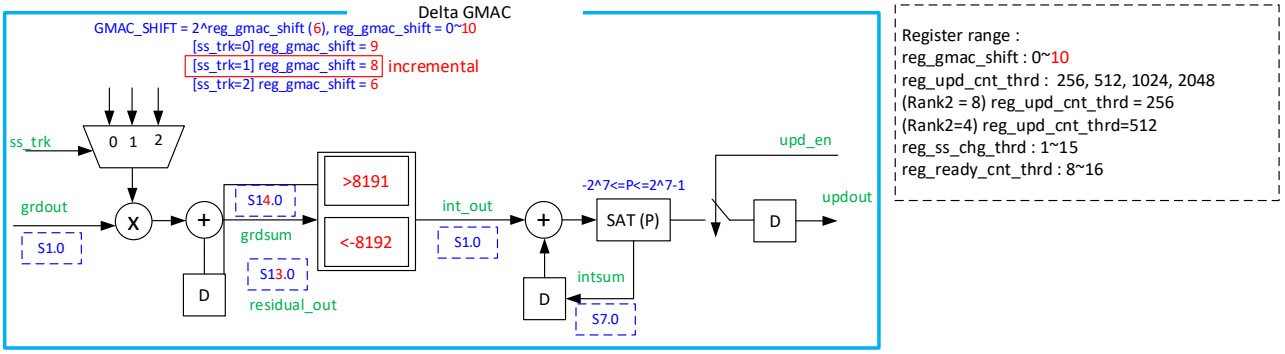
Initial_ADC_GOS[0]= (max(Startup_ADC_GOS[1:63])-min(Startup_ADC_GOS[1:63]))/2, where k=0~63

Initial_ADC_GOS[k]=Startup_ADC_GOS[k]-(max(Startup_ADC_GOS[1:63])+min(Startup_ADC_GOS[1:63]))/2, where k=1~63

12.6 Incremental Calibration

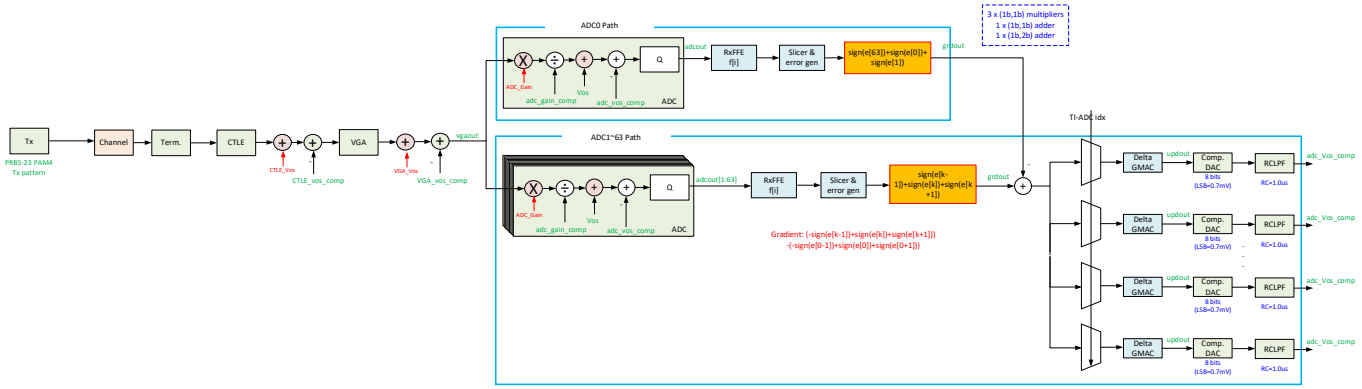
Receiver will go to power saving mode or sleep mode, and we will need incremental calibration before we resume data traffic. The purpose of incremental calibration is to calibrate VOS in a very short time.

For ADC VOS, we will set the GMAC_SHIFT to 2^8, which is small gain to have stable VOS calibration results. Also, we set the ready timeout to 2, so that the incremental ADC VOS calibration will be done in two iterations



For VGA and CTLE VOS calibration, we will have default GMAC_SHIFT settings with ready timeout count set to 256, which is the shortest period to reduce calibration time.

12.7 Continuous ADC Vos Calibration (MM-CDR Case)



In normal mode, also known as the continuous calibration mode, the change in ADC offsets due to temperature changes need to be tracked and cancelled out also for optimal performance. With max GPU activity, we can expect temperature delta of +80 DegC which will induce offset of ~6.25mV assuming exponential settling with time-constant of 10ms, where ~14.3mV offset change under 165 DegC temperature change. We need to make the offset calibration loops fast enough to keep vos drift in normal mode.

Unlike startup mode, continuous calibration needs to work with real traffic data. Furthermore, if TxFIR is not present in certain applications to provide the necessary PAM4 PR1 equalization at ADC output, this continuous calibration algorithm must work at the output of RxFFE, which provides the necessary equalization. However, RxFFE also complicates the estimation of ADC voltage offsets as the RxFFE output for each of the interleaves has voltage offset error contributions from adjacent ADC interleaves due to weighting from pre and post cursor RxFFE coefficients. However, based on many simulations under a variety of conditions, the same simple gradient of the sign of the

error at the corresponding i-th RxFFE interleave can still be used for cancelling the offset at the i-th ADC interleave.

Instead of calibrating each interleave ADC to absolute “0V”, we align i-th ADC (i=1, 2, …63) to ADC0 to save range of compensation DAC. The common Vos of all ADCs are compensated through CTLE Vos DAC by baseline wander estimator, which will be addressed in later section.

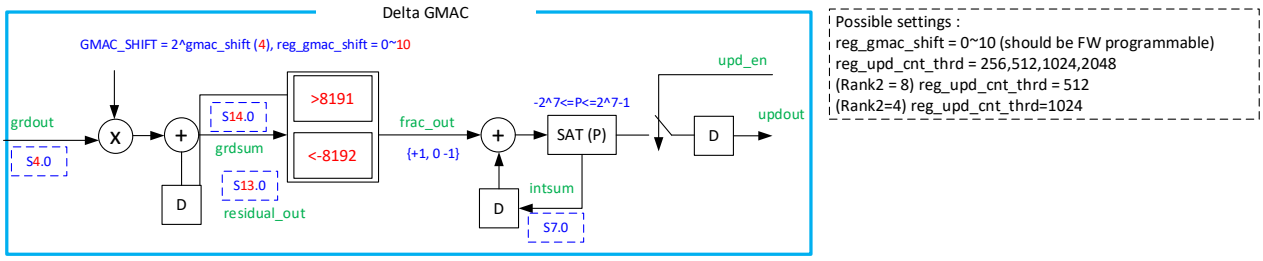
The gradient equations for offset correction for the 64 T/Hs (i=0, 1, 2, …63) are given by with the

following update equation for the periodic calibration mode.

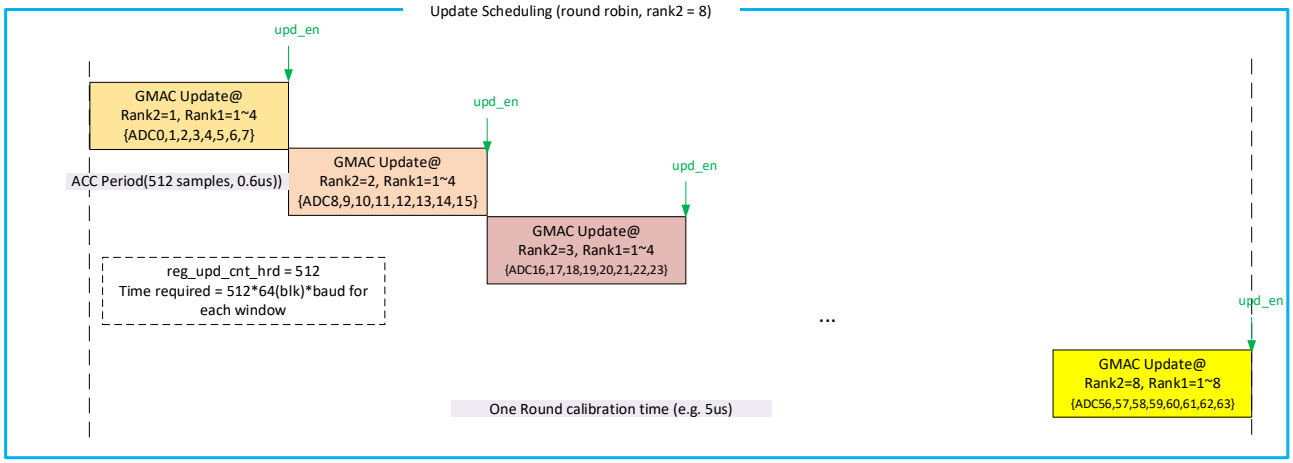
$$grdsum(i,n+1) = grdsum(i,n) + 2^{GMAC_{SHIFT}}((-sign\{e(i-1,n)\} + sign\{e(i,n)\} + sign\{e(i+1,n)\}) - (-sign\{e(-1,n)\} + sign\{e(0,n)\} + sign\{e(1,n)\})),$$

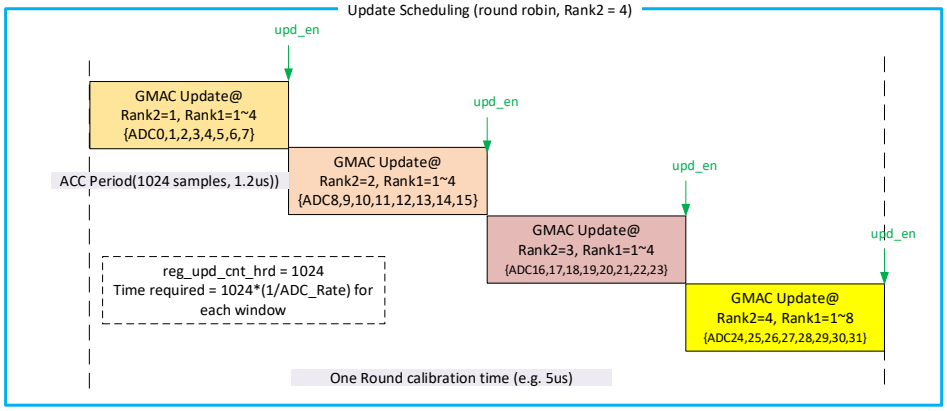
where $e(i,n) = z_{rxffe}(i,n) - y(i,n)$, where $z_{rxffe}(i,n)$ is the soft value at RxFFE output and $y(i,n)$ is the data slicer data, so e(I,n) is the error for i-th ADC interleave based on i-th interleave RxFFE output. The definition of sign function is the same as startup ADC Vos calibration.

As long as grdsum[I,n] touches the fractional threshold +8191/-8192, it will increase/decrease one digital code to integer loop. The integer loop will send out compensation code to compensation DAC with 0.7mV step for each 64 TI-ADC. The GMAC_SHIFT of fractional loop can be varied for purpose of fast acquisition and stable tracking. There is default value proposed and possible value range attached

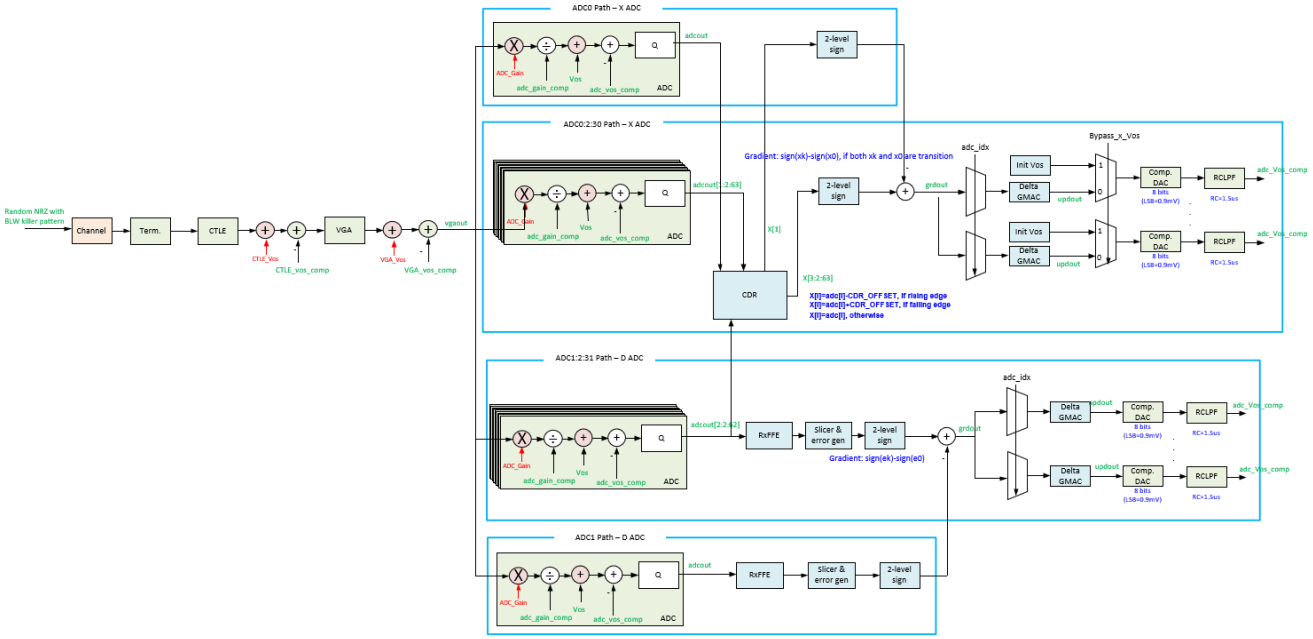


Continuous mode follows the same update scheme of startup Vos calibration to share GMAC and avoid damping due to compensation DAC RC effect. The control for upd_en and input_idx are shown below.





12.8 Continuous ADC Vos Calibration (BB-CDR Case)



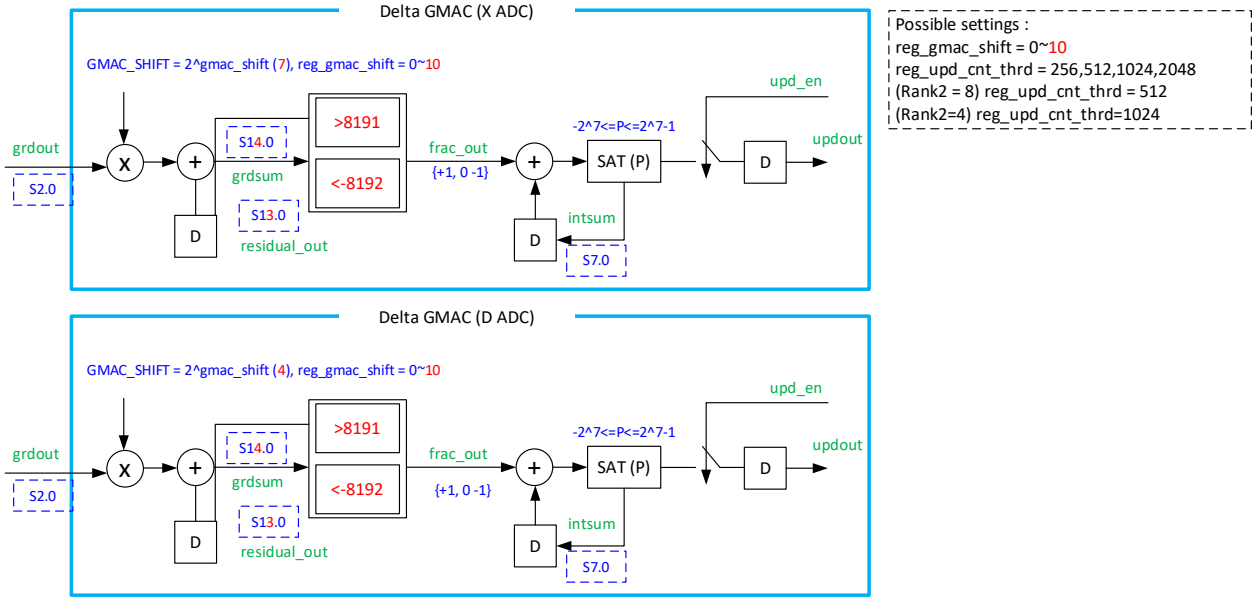
There are D ADC and X ADC for data samples and edge samples respectively. The continuous Vos calibration on D ADC follows the continuous Vos calibration of MM-CDR case in section 10.7; however, the continuous Vos calibration on X ADC is optional and the gradient equation is different to D ADC. Since X ADC samples will not pass through RxFFE, so we will calibrate Vos of X ADC by using CDR data. The CDR will calculate X value based on following equation

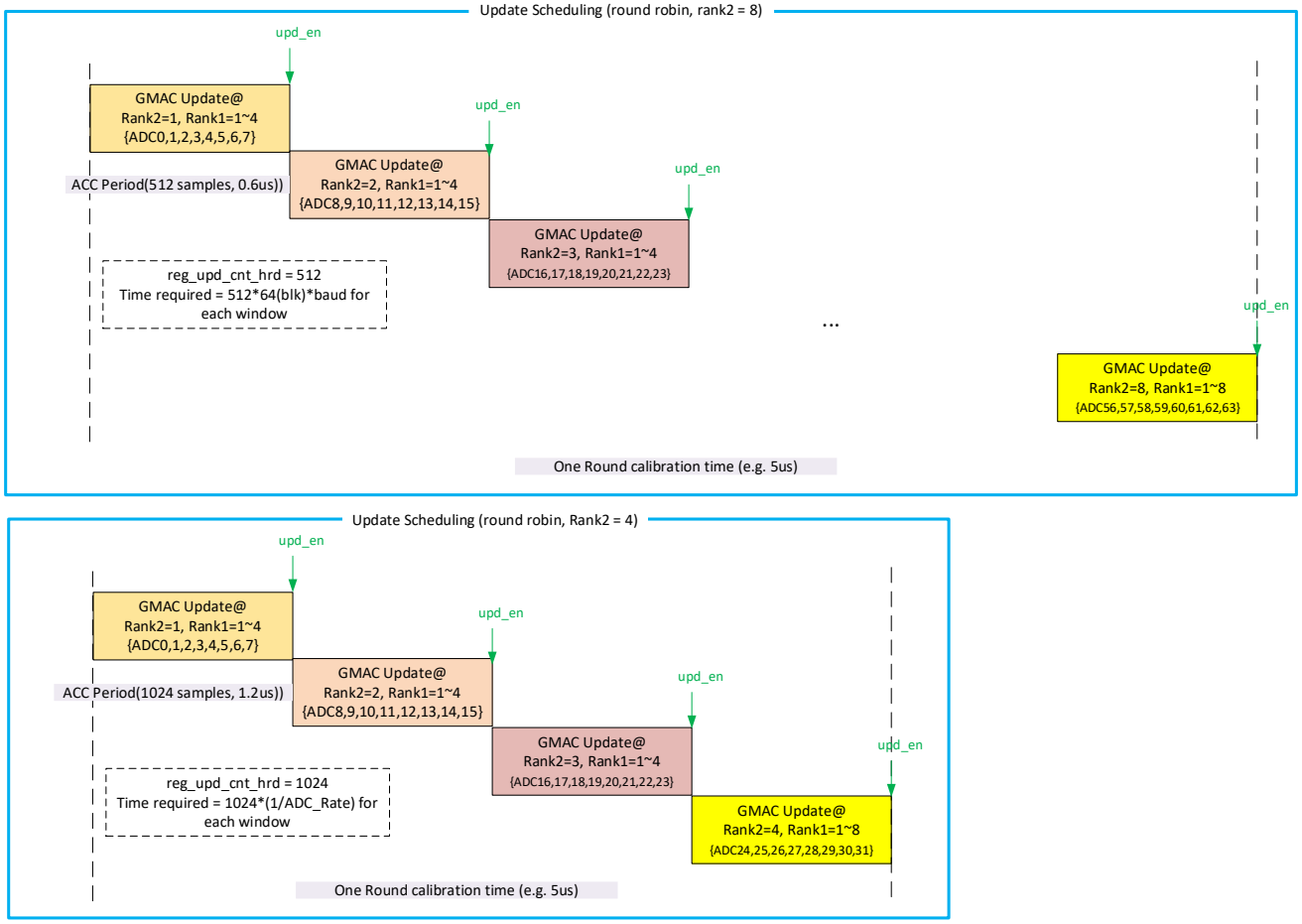
$$\begin{cases} X[i] = adc[i] - CDR\_OFFSET, \text{if rising edge} \\ X[i] = adc[i] + CDR\_OFFSET, \text{if falling edge} \\ X[i] = adc[i], \text{otherwise} \end{cases}$$

The gradient equations for offset correction for the edge T/Hs (i=0,2,4, …, 30) are given by with the following update equation for the periodic calibration mode.

$$grdsum(i, n+1) = grdsum(i, n) + 2^{GMAC\_SHIFT}(sign\{X(i, n)\} - sign\{X(0, n)\}),$$

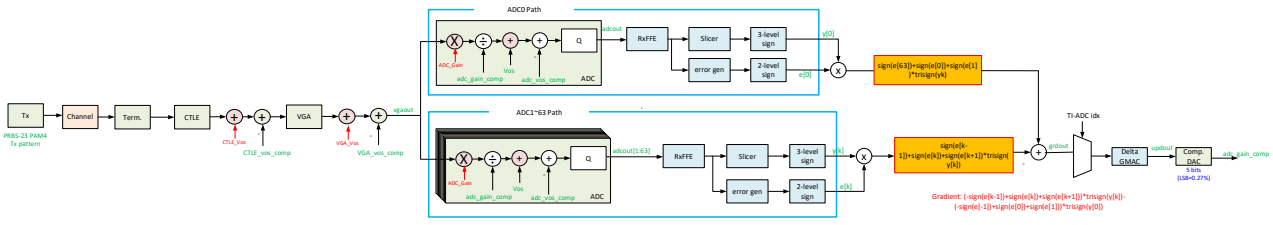
, and the gradient is calculated if both X(i,n) and X(0,n) are either rising or falling edge. The definition of sign function is the same as startup ADC Vos calibration.





The delta GMAC and control signal follow the same definition of continuous Vos calibration of MM-CDR case.

12.9 Continuous ADC Gain Calibration



In normal mode, also known as the continuous calibration mode, the change in ADC gain offset due to temperature changes need to be tracked and cancelled out also for optimal performance. With max GPU activity, we can expect temperature delta of +80 DegC which will induce gain offset change of ~1% assuming linear change with time-constant of 10ms, where ~2% gain offset change under 165 DegC temperature change. We need to make the offset calibration loops fast enough to keep gos drift in normal mode.

Unlike startup mode, continuous calibration needs to work with real traffic data. Furthermore, if TxFIR is not present in certain applications to provide the necessary PAM4 PR1 equalization at ADC output, this continuous calibration algorithm must work at the output of RxFFE, which provides the necessary equalization. Since the gradient driving force shows up at RxFFE output, so simple gradient of the sign of the error at the corresponding i-th RxFFE interleave can still be used for cancelling the gain offset at the i-th ADC interleave. We need a reference signal for ADC Gain calibration, so ADC0 is used to align i-th ADC (i=1, 2, …63) for ADC Gain offset.

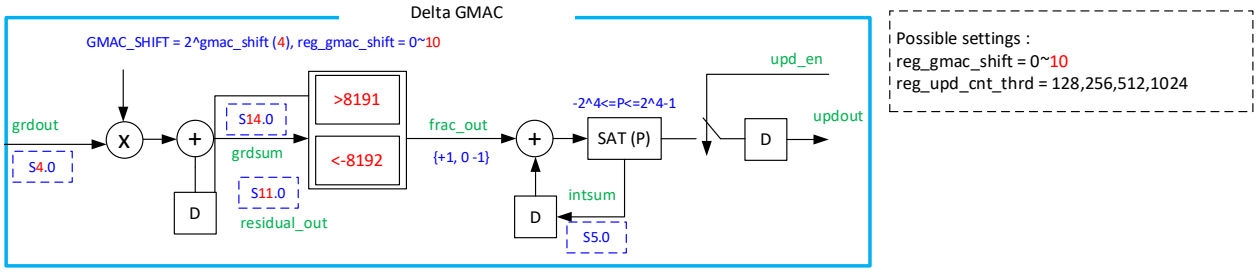
The gradient equations for offset correction for the 64 T/Hs (i=0, 1, 2, …63) are given by with the following update equation for the periodic calibration mode.

$$grdsum(i,n+1) = grdsum(i,n) + 2^{GMAC_{SHIFT}}((-sign\{e(i-1,n)\} + sign\{e(i,n)\} + sign\{e(i+1,n)\})trisign\{y(i,n)\} - ((-sign\{e(-1,n)\} + sign\{e(0,n)\} + sign\{e(1,n)\})trisign\{y(0,n)\})),$$

where $e(i,n) = z_{rxffe}(i,n) - y(i,n)$, where $z_{rxffe}(i,n)$ is the soft value at RxFFE output and $y(i,n)$ is the data slicer data, so e(I,n) is the error for i-th ADC interleave based on i-th interleave RxFFE output. The definition of sign function is the same as startup ADC Vos calibration. Since the probability of y(i,n) to be zero is high, so we need trisign function to avoid inducing bias value. The definition of trisign function is defined as

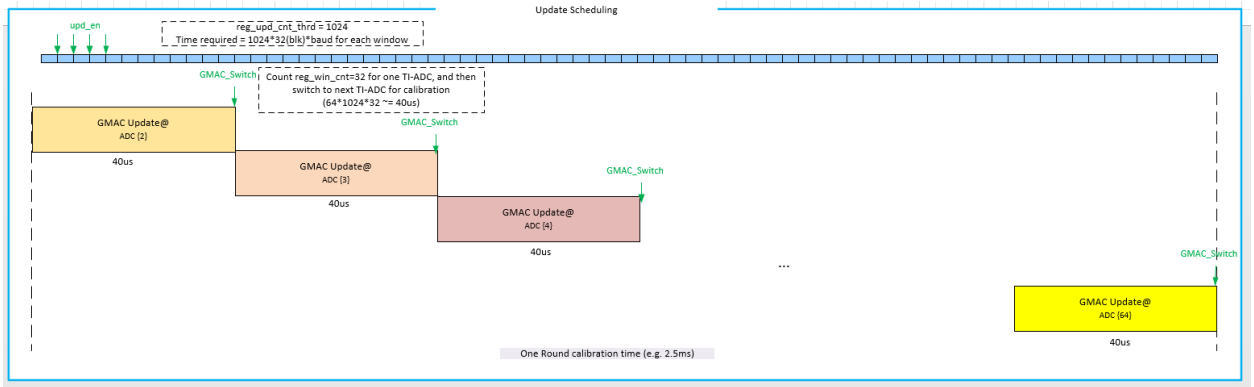
$$\text{trisign(x)}=\begin{cases} 1, & x > 0 \\ 0, & x = 0 \\ -1, & x < 0 \end{cases}$$

As long as grdsum[i,n] touches the fractional threshold +8191/-8192, it will increase/decrease one digital code to integer loop. The integer loop will send out compensation code to compensation DAC with 0.27% step for each 64 TI-ADC. The GMAC_SHIFT of fractional loop can be varied for purpose of fast acquisition and stable tracking. There is default value proposed and possible value range attached

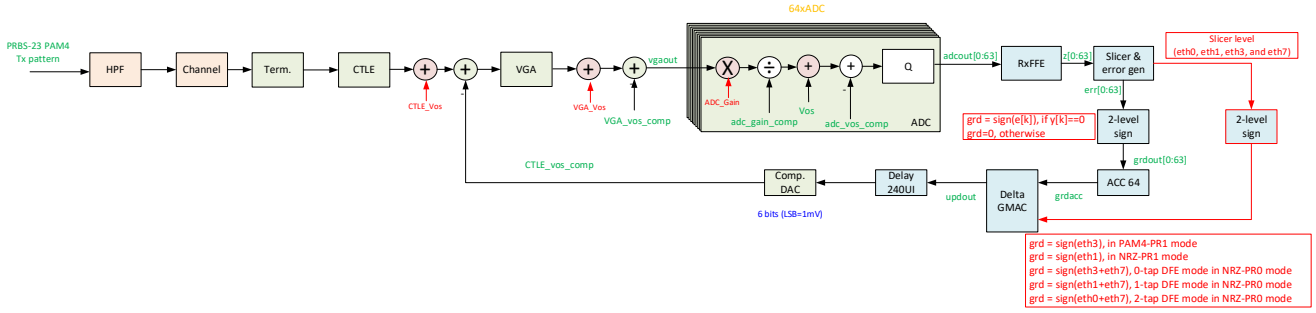


Since we do not have fractional memory, so GMAC will calibrate on one TI-ADC entirely, and then move on to next TI-ADC. At each period of updates, we have 32 accumulation windows, and each accumulation windows accumulates 1024 samples to have one opportunity to update ADC Gain compensation DAC. The initial drift of continuous GOS might be large (2%),

so multiple update opportunities provide capability of large acquisition range. One period of updates cost around 40us, to update 64 TI-ADCs, it takes around 2.5ms.



12.10 Baseline Wander Cancellation



There is high-pass filter effect of on-board or on-die capacitor, so the signal will have bias after a long same polarity symbol stream. Such bias will be present as common offset for all time-interleave ADCs and can be processed and cancelled through CTLE Vos DAC by using RxFFE output error.

In continuous mode, Vos of i-th ADC will align to ADC0 so that all time-interleaved ADC will have same Vos as ADC0. This common Vos could be estimate and compensate through CTLE Vos DAC by using RxFFE output error as well.

The gradient equations for baseline wander correction and common offset compensation is given by with the following update equation for the periodic calibration mode.

$$grdsum(n+1) = grdsum(n) + 2^{GMAC\_SHIFT} \sum_{i=0}^{63} sign\{e(i,n)\}, \text{ if y[k]==0}$$

$$grdsum(n+1) = grdsum(n) + 2^{GMAC_{SHIFT}} sign(eth3), \text{ if PAM4-PR0 mode}$$

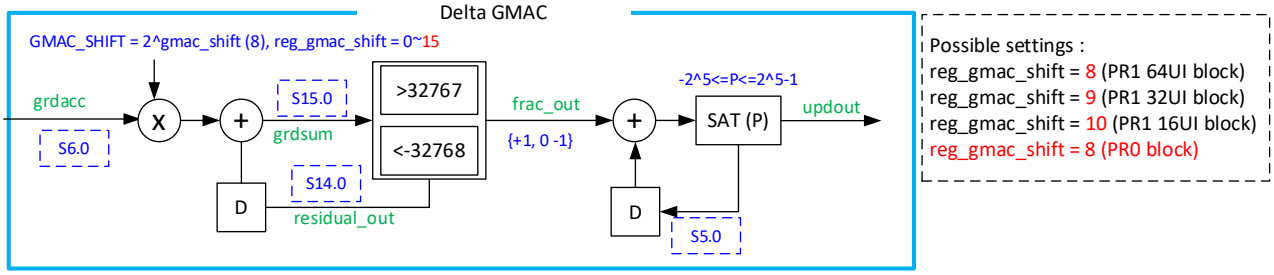
$$grdsum(n+1) = grdsum(n) + 2^{GMAC_{SHIFT}} sign(eth1), \text{ if NRZ-PR0 mode}$$

$$grdsum(n+1) = grdsum(n) + 2^{GMAC_{SHIFT}} sign(eth3 + eth7), \text{ if 0-tap DFE in NRZ-PR0 mode}$$

$$grdsum(n+1) = grdsum(n) + 2^{GMAC_{SHIFT}} sign(eth1 + eth7), \text{ if 1-tap DFE in NRZ-PR0 mode}$$

$grdsum(n+1) = grdsum(n) + 2^{GMAC_{SHIFT}} sign(eth0 + eth7)$, if 2-tap DFE in NRZ-PR0 mode

where $e(i,n) = z_{rxffe}(i,n) - y(i,n)$, where $z_{rxffe}(i,n)$ is the soft value at RxFFE output and $y(i,n)$ is the data slicer data, so e(I,n) is the error for i-th ADC interleave based on i-th interleave RxFFE output. The slicer levels used for PR1 mode are level 0, and the slicer levels used for PR0 mode are symmetric levels. The definition of sign function is the same as startup ADC Vos calibration.



As long as grdsum[I,n] touches the fractional threshold +32767/-32768, it will increase/decrease one digital code to integer loop. The integer loop will send out compensation code to compensation DAC with 1.0mV step for each 64 TI-ADC. The GMAC_SHIFT of fractional loop can be varied for purpose of fast acquisition and stable tracking. There is default value proposed and possible value range attached

The step size of different CTLE LFG/MFG gears are different, and some examples are shown as below

Systematic Error (mV)	5

Srikanth Sundaram: To account for systematic errors from layout

Srikanth Sundaram: Nsigma = (Vos_range - 5mV)/offset_sigma

5mV is subtracted to account for systematic offset

Srikanth Sundaram: VGA/TAH/ADC offsets are divided by VGA gain and referred to CTLE output

Srikanth Sundaram: Baseline Wander is referred to the CTLE output by TERM + CTLE dc gain

100G MODE

Temp = 125C	NO LTE - LFG/MFG=7		vos_range=0			vos_range=1			Temp Drift								
Corner	CTLE offset (mVrms)	Other offset (mVrms)	CTLE vos range (mV)	vos_lsb (mV)	N sigma	CTLE vos range (mV)	vos_lsb (mV)	N sigma	VGA (mVrms)	TAH0 (mVrms)	ADC0 (mVrms)	TAH_buf gain(dB)	VGA dc_gain(dB)	BLW (mVrms)	CTLE dc_gain (dB)	TERM attn (dB)	Other offset (mVrms)
FFG	4.11	2.3	41.6	1.38	7.8	50.9	1.70	9.8	2.2	0.58	1.6	-1.11	2.32	0.6	-0.13	-4	2.3
FSG	4.13	2.1	38.9	1.25	7.3	47.7	1.62	9.2	2	0.57	1.6	-1.12	2.59	0.6	0.42	-4	2.1
SFG	4.14	2.0	41.2	1.35	7.8	50.2	1.68	9.8	1.9	0.58	1.6	-1.01	2.56	0.6	0.42	-4	2.0
SSG	3.96	1.9	37.2	1.25	7.3	45.3	1.54	9.1	1.7	0.58	1.6	-1.02	2.57	0.6	0.6	-4	1.9
TTG	4.15	1.9	40.2	1.32	7.7	49.0	1.65	9.7	2	0.56	1.2	-1.06	2.74	0.6	0.45	-4	1.9

Temp = 125C	MID LTE - LFG/MFG=3		vos_range=0			vos_range=1			Temp Drift								
Corner	CTLE offset (mVrms)	Other offset (mVrms)	CTLE vos range (mV)	vos_lsb (mV)	N sigma	CTLE vos range (mV)	vos_lsb (mV)	N sigma	VGA (mVrms)	TAH0 (mVrms)	ADC0 (mVrms)	TAH_buf gain(dB)	VGA dc_gain(dB)	BLW (mVrms)	CTLE dc_gain (dB)	TERM attn (dB)	Other offset (mVrms)
FFG	3.27	2.3	32.6	1.07	6.9	40.0	1.35	8.8	2.2	0.58	1.6	-1.11	2.32	0.6	-2.71	-4	2.3
FSG	3.28	2.1	30.4	1.00	6.5	37.2	1.25	8.3	2	0.57	1.6	-1.12	2.59	0.6	-2.1	-4	2.1
SFG	3.26	2.0	31.9	1.05	7.0	38.9	1.3	8.8	1.9	0.58	1.6	-1.01	2.56	0.6	-2.17	-4	2.0
SSG	3.15	1.9	29.0	0.95	6.5	35.3	1.18	8.2	1.7	0.58	1.6	-1.02	2.57	0.6	-1.89	-4	1.9
TTG	3.28	1.8	31.2	1.02	7.0	38.1	1.27	8.8	2	0.56	1.2	-1.06	2.74	0.6	-2.11	-4	1.8

Temp = 125C	MAX LTE - LFG/MFG=0		vos_range=0			vos_range=1			Temp Drift								
Corner	CTLE offset (mVrms)	Other offset (mVrms)	CTLE vos range (mV)	vos_lsb (mV)	N sigma	CTLE vos range (mV)	vos_lsb (mV)	N sigma	VGA (mVrms)	TAH0 (mVrms)	ADC0 (mVrms)	TAH_buf gain(dB)	VGA dc_gain(dB)	BLW (mVrms)	CTLE dc_gain (dB)	TERM attn (dB)	Other offset (mVrms)
FFG	2.75	2.2	32.5	1.08	7.7	32.5	1.08	7.7	2.2	0.58	1.6	-1.11	2.32	0.6	-6.42	-4	2.2
FSG	2.74	2.1	29.8	0.99	7.2	29.8	0.99	7.2	2	0.57	1.6	-1.12	2.59	0.6	-5.67	-4	2.1
SFG	2.71	2.0	31.0	1.04	7.7	31.0	1.04	7.7	1.9	0.58	1.6	-1.01	2.56	0.6	-5.84	-4	2.0
SSG	2.62	1.9	27.9	0.94	7.1	27.9	0.94	7.1	1.7	0.58	1.6	-1.02	2.57	0.6	-5.35	-4	1.9
TTG	2.73	1.8	30.4	1.02	7.7	30.4	1.02	7.7	2	0.56	1.2	-1.06	2.74	0.6	-5.74	-4	1.8

Temp = 125C	Min vos_range - LFG=3/MFG=0		vos_range=0			vos_range=1			Temp Drift								
Corner	CTLE offset (mVrms)	Other offset (mVrms)	CTLE vos range (mV)	vos_lsb (mV)	N sigma	CTLE vos range (mV)	vos_lsb (mV)	N sigma	VGA (mVrms)	TAH0 (mVrms)	ADC0 (mVrms)	TAH_buf gain(dB)	VGA dc_gain(dB)	BLW (mVrms)	CTLE dc_gain (dB)	TERM attn (dB)	Other offset (mVrms)
FFG	2.96	2.3	28.7	0.94	6.4	35.2	1.18	8.1	2.2	0.58	1.6	-1.11	2.32	0.6	-4.6	-4	2.3
FSG	2.95	2.1	26.6	0.87	6.0	32.6	1.10	7.6	2	0.57	1.6	-1.12	2.59	0.6	-3.96	-4	2.1
SFG	2.93	2.0	27.7	0.9	6.4	33.8	1.13	8.1	1.9	0.58	1.6	-1.01	2.56	0.6	-4.07	-4	2.0
SSG	2.83	1.9	25.2	0.83	5.9	30.8	1.03	7.5	1.7	0.58	1.6	-1.02	2.57	0.6	-3.71	-4	1.9
TTG	2.95	1.8	27.2	0.89	6.4	33.4	1.1	8.2	2	0.56	1.2	-1.06	2.74	0.6	-3.99	-4	1.8

The PR1 mode CTLE VOS continuous calibration will use sign{e(I,n)} for gradient calculation in ACQ mode. In TRK mode, the gradient will change to use sign(eth3) with eth3 adaptation to track on common voltage offset and baseline wander

12.11 Continuous Calibration RxFFE Constraint

Rx calibration takes the error data of RxFFE, and the ADC mismatches are mixed by RxFFE. The targeting ADC that we are calibrating using RxFFE error data is the ADC aligned with main-tap, and main-tap is the 4th tap of RxFFE, which is located on even tap. If the even tap summation is smaller than odd tap summation, the odd tap ADC mismatch term will be dominant. Following is an example of even tap sum (82) smaller than odd tap sum(99), and a specific zig-zag ADC mismatch pattern, and we can see the polarity of VOS at RxFFE is different to ADC VOS

	F[-3]	F[-2]	F[-1]	F[0]	F[1]	F[2]	F[3]	F[4]	F[5]
RxFFE	-5	12	-45	128	117	-46	29	-12	3
ADC VOS	-5	5	-5	5	-5	5	-5	5	-5
RxFFE out				-85					

The calibration gradient polarity will also be different to ADC VOS, so it makes the calibration go divergence

ADCNo.	ADC0	ADC1	ADC2	ADC3	ADC4	ADC5	ADC6	ADC7	ADC8
ADC VOS	-5	5	-5	5	-5	5	-5	5	-5
RxFFE out	85	-85	85	-85	85	-85	85	-85	85
GRD (sign-error)	1	-1	1	-1	1	-1	1	-1	1
GRD (simplified full grd)	1	-1	1	-1	1	-1	1	-1	1

Since mathematically the gradient will have different polarity to ADC VOS, so we can only avoid such situation from happening, If the calibration divergence happens fast, the FoM will be small and we can avoid FEQ procedure to select such configuration; however, there are some cases that the divergence speed is not fast enough, so we need to identify such condition. We have FW method to avoid divergence that we check the RxFFE coefficients while sweeping CTLE and f1 during FEQ period. As long as we found that the RxFFE coefficient even tap sum is smaller odd tap sum, then it is possible to be divergent, so we should avoid the FEQ procedure to choose such configuration by making the FoM to be very small. The FW action as below

For CTLE = CTLE Sweeping set

For f1 = f1 sweeping set

Adapt RxFFE and enable other loops (e.g. CDR, RxCAL)

Calculate sum(ffe even tap) and sum(ffe odd tap)

If sum(ffe even tap) > sum(ffe odd tap) + thresh

Record FoM, CTLE, and f1 settings

If sum(ffe even tap) <= sum(ffe odd tap) + thresh

Set FoM=0

End

End

The thresh parameters can be calculated by sum(ffe even tap) * ratio, where ratio is reserved as 0.125, 0.2, 0.25, 0.33. And we set default as 0.2.

12.12 Rx Calibration Summary

To have optimal performance, ADC Vos, CTLE/VGA Vos, and ADC Gain Offset mismatch are needed to be estimated and compensated. There are two types of calibration, startup and continuous. All impairment is calibrated during startup following a specific sequence; however, to consider temperature variation, ADC Vos and ADC GOS should be executed in continuous mode. CTLE Vos, which is named as baseline wander cancellation, should be executed for continuous mode.

Startup Vos of ADC, VGA, and CTLE will be targeting absolute 0, and startup gain offset of ADC will be targeting ADC0. Continuous ADC Vos will be targeting ADC0 and baseline wander will cancel common offset including ADC0 Vos variation. The impairment range and compensation spec of current design and gradient equation and calibration target are summarized in following table.

Type	OSR	Cal	Yes/No	Methodology (gradient)	DAC RC time constant	Impairment Range	Compensation spec.
Startup	1X/2X	ADC Vos	Y	sign(zk) (zk: RxFFE out with [0 ... 0 128 0 ... 0])	1us	σ = 5mV Range: +/-30mV	8 bits (LSB = 0.7mV) Range: +/-34mV
		VGA Vos	Y	sign(zk) (zk: RxFFE out with [0 ... 0 128 0 ... 0])	<2.5ns	σ = 4.5mV Range: +/-27mV	6 bits (LSB = 1.5mV) Range: +/-32mV
		CTLE Vos	Y	sign(zk) (zk: RxFFE out with [0 ... 0 128 0 ... 0])	<2.5ns	σ = 4mV Range: +/-24mV	6 bits (LSB = 1.0mV) Range: +/-32mV
	1X	ADC Gain	Y	(sign(zk-z0)*sign(dk)), dk is input signal (zk is output of RxFFE-k - RxFFE_0 with [0 ... 0 128 0 ... 0])	<2.5ns (input DC toggling RC: 100ns)	σ = 0.5% Range: +/- 3%	5 bit (step = 0.27%) Range +/-4%
	2X	ADC Gain	N				
Type	OSR	Cal	Yes/No	Methodology (gradient)	DAC RC time constant	Residual after Startup Cal.	Compensation spec.
Continuous	1X	ADC Vos	Y	grd = (-sign(e[k-1])+sign(e[k])+sign(e[k+1])) - (-sign(e[-1])+sign(e[0])+sign(e[1])) (ek = RxFFE error for ADC-k)	1us	+/-5.3mV (RC ripple 1.3mV + 4mV)	temp. drifting: 15mV/165°C
	2X	ADC Vos	Y	[Data sample ADC] grd = sign(ek)-sign(e0) (ek = DFE error for ADC-k) [Edge sample ADC] grd = sign(xk)-sign(x0) (xk = CDR X value of ADC-k) xk = adc[k]-CDR_OFFSET, if rising edge xk = adc[k]+CDR_OFFSET, if falling edge grd=0 if xk or x0 is not tranisition			
	1X	ADC Gain	N	{-sign(e[k-1])+sign(e[k])+sign(e[k+1])} * trisign(y[k]) - {-sign(e[-1])+sign(e[0])+sign(e[+1])} * trisign(y[0]) (ek=RxFFE error for ADC-k, yk is PR1 symbol for ADC-K)	<2.5ns		temp. drifting: 2%/165°C
	1X	CTLE Vos	Y	CTLE_Vos = CTLE_Vos(no update) + Baseline Wander grd = sign(ek), if y[k]==0 (ek = RxFFE error) [ACQ] + ETH3 freeze -> default grd = sign(eth3) (eth3 = RxFFE error slicer, PAM4-PR1) +ETH3 update[TRK] grd = sign(eth1) (eth1 = RxFFE error slicer, NRZ-PR1) + ETH1 update [TRK]	<2.5ns		
	2X	CTLE Vos	Y	[0-tap DFE] grd = sign(eth7 + eth3), where eth7 is the slicer level for +1*h0, and eth3 is the slicer level for -1*h0 [1-tap DFE] grd = sign(eth7 + eth1), where eth7 is the slicer level for +1*h0+1*h1, and eth1 is the slicer level for -1*h0-1*h1 [2-tap DFE] grd = sign(eth7 + eth0), where eth7 is the slicer level for +1*h0+1*h1+1*h2, and eth0 is the slicer level for -1*h0-1*h1-1*h2			6 bits (LSB = 1.0mV) Range: +/-32mV

The calibration for each rate/protocols are also listed below.

Interface	Protocol	Baudrate (GBps)	Modulation	RX Core Interface (# Sym)	Baudperiod (pS)	RX Mode	EQ Target	CDR	RX Interleave (Rank1_ways)	DCO_freq	ADC resolution (#bits)	# Rank2 Interleave (Rank2_ways)	ADC conversion (GHz)	Compator speed (GHz)		Startup ADC VOS (D ADC)	Startup ADC VOS (E ADC)	Startup ADC GOS (D ADC)	Startup ADC GOS (E ADC)	Startup VGA Vos	Startup CTLE Vos	Incremental ADC VOS (D ADC)	Incremental ADC VOS (E ADC)	Continuous ADC VOS (D ADC)	Continuous ADC VOS (E ADC)	Continuous ADC GOS (D ADC)	Continuous ADC GOS (E ADC)	Continuous BLW CTLE VOS
NVHS	106Gbps	53.125	PAM4	32	18.82	DSP	PR-1	M-M	8	6.640625	7	8	0.830078	6.640625		Y	N/A	Y	N/A	Y	Y	Y	Y	Y	N/A	Y	N/A	Y
	106Gbps	53.125	PAM4	32	18.82	DSP	PR-1	M-M	8	6.640625	7	8	0.830078	6.640625		Y	N/A	Y	N/A	Y	Y	Y	Y	Y	N/A	Y	N/A	Y
	53Gbps	26.5625	PAM4	16	37.65	DSP	PR-1	M-M	4	6.640625	7	8	0.830078	6.640625		Y	N/A	Y	N/A	Y	Y	Y	Y	Y	N/A	Y	N/A	Y
	53Gbps	26.5625	PAM4	16	37.65	DSP	PR-1	M-M	4	6.640625	7	8	0.830078	6.640625		Y	N/A	Y	N/A	Y	Y	Y	Y	Y	N/A	Y	N/A	Y
	50Gbps	50	NRZ	32	20.00	DSP	PR-1	M-M	8	6.25	7	8	0.78125	6.25		Y	N/A	Y	N/A	Y	Y	Y	Y	Y	N/A	Y	N/A	Y
PCIE	Gen6 - 64Gbps	32	PAM4	32	31.25	DSP	PR-1	M-M	8	4	7	8	0.5	4		Y	N/A	Y	N/A	Y	Y	Y	Y	Y	N/A	Y	N/A	Y
	Gen5 - 32Gbps	32	NRZ	32	31.25	DSP	PR-1	M-M	8	4	5	8	0.5	3		Y	N/A	N	N/A	Y	Y	Y	Y	Y	N/A	N	N/A	Y
	Gen4 - 16Gbps	16	NRZ	16	62.50	2-tap DFE	PR-0	2X	4	4	4	8	0.5	2.5		Y	Y	N	N	Y	Y	Y	Y	Y	Optional	N	N	Y
	Gen3 - 8Gbps	8	NRZ	16	125.00	1-tap DFE	PR-0	2X	2	4	4	8	0.5	2.5		Y	Y	N	N	Y	Y	Y	Y	N	Optional	N	N	N
	Gen2 - 5Gbps	5	NRZ	16	200.00	VGA-Only	PR-0	2X	2	2.5	1	4	0.625	1.25		Y	Y	N	N	Y	Y	Y	Y	N	Optional	N	N	N
	Gen1 - 2.5Gbps	2.5	NRZ	16	400.00	VGA-Only	PR-0	2X	1	2.5	1	4	0.625	1.25		Y	Y	N	N	Y	Y	Y	Y	N	Optional	N	N	N
IEEE	106Gbps	53.125	PAM4	32	18.82	DSP	PR-1	M-M	8	6.640625	7	8	0.830078	6.640625		Y	N/A	Y	N/A	Y	Y	N/A	N/A	Y	N/A	Y	N/A	Y
	51.5625Gbps	25.78125	PAM4	16	38.79	DSP	PR-1	M-M	4	6.445313	7	8	0.805664	6.445313		Y	N/A	Y	N/A	Y	Y	N/A	N/A	Y	N/A	Y	N/A	Y
	53Gbps	26.5625	PAM4	16	37.65	DSP	PR-1	M-M	4	6.640625	7	8	0.830078	6.640625		Y	N/A	Y	N/A	Y	Y	N/A	N/A	Y	N/A	Y	N/A	Y
	25G - KR4	25.78125	NRZ	16	38.79	3-tap DFE	PR-0	2X	4	6.445313	4	8	0.805664	4.02832		Y	Y	N	N	Y	Y	N/A	N/A	Y	Optional	N	N	Y
	USXGMII_10G	10.3125	NRZ	16	96.97	2-tap DFE	PR-0	2X	2	5.15625	4	8	0.644531	3.222656		Y	Y	N	N	Y	Y	N/A	N/A	Y	Optional	N	N	Y
	USXGMII_10G	10.3125	NRZ	16	96.97	VGA-Only	PR-0	2X	2	5.15625	1	8	0.644531	1.289063		Y	Y	N	N	Y	Y	N/A	N/A	N	Optional	N	N	N
	USXGMII_5G	5.15625	NRZ	16	193.94	VGA-Only	PR-0	2X	2	2.578125	1	4	0.644531	1.289063		Y	Y	N	N	Y	Y	N/A	N/A	N	Optional	N	N	N
	SGMII_2.5G	3.125	NRZ	16	320.00	VGA-Only	PR-0	2X	1	3.125	1	4	0.78125	1.5625		Y	Y	N	N	Y	Y	N/A	N/A	N	Optional	N	N	N
	SGMII_1G	1.25	NRZ	16	800.00	VGA-Only	PR-0	2X	0.5	2.5	1	4	0.625	1.25		Y	Y	N	N	Y	Y	N/A	N/A	N	Optional	N	N	N
	SGMII_1G	1.25	NRZ	16	800.00	VGA-Only	PR-0	2X	0.5	2.5	1	4	0.625	1.25		Y	Y	N	N	Y	Y	N/A	N/A	N	Optional	N	N	N
IB	106Gbps - NDR	53.125	PAM4	32	18.82	DSP	PR-1	M-M	8	6.640625	7	8	0.830078	6.640625		Y	N/A	Y	N/A	Y	Y	Y	Y	Y	N/A	Y	N/A	Y
	53Gbps - HDR	26.5625	PAM4	16	37.65	DSP	PR-1	M-M	4	6.640625	7	8	0.830078	6.640625		Y	N/A	Y	N/A	Y	Y	Y	Y	Y	N/A	Y	N/A	Y
	25Gbps - EDR	25.78125	NRZ	16	38.79	2-tap DFE	PR-0	2X	4	6.445313	4	8	0.805664	4.02832		Y	Y	N	N	Y	Y	Y	Y	Y	Optional	N	N	Y
	14Gbps - FDR	14.0625	NRZ	16	71.11	1-tap DFE	PR-0	2X	4	3.515625	4	4	0.878906	4.394531		Y	Y	N	N	Y	Y	Y	Y	Y	Optional	N	N	Y
	10Gbps - QDR	10	NRZ	16	100.00	1-tap DFE	PR-0	2X	2	5	4	8	0.625	3.125		Y	Y	N	N	Y	Y	Y	Y	Y	Optional	N	N	Y
	5Gbps - DDR	5	NRZ	16	200.00	VGA-Only	PR-0	2X	2	2.5	1	4	0.625	1.25		Y	Y	N	N	Y	Y	Y	Y	N	Optional	N	N	N
	2.5Gbps - SDR	2.5	NRZ	16	400.00	VGA-Only	PR-0	2X	1	2.5	1	4	0.625	1.25		Y	Y	N	N	Y	Y	Y	Y	N	Optional	N	N	N
USB 3.1	Gen2	10	NRZ	16	100.00	2-tap DFE	PR-0	2X	2	5	4	8	0.625	3.125		Y	Y	N	N	Y	Y	Y	Y	Y	Optional	N	N	Y
	Gen1	5	NRZ	16	200.00	1-tap DFE	PR-0	2X	1	5	4	8	0.625	3.125		Y	Y	N	N	Y	Y	Y	Y	Y	Optional	N	N	Y

13 Illustrative Simulation Results

The following illustrate some example simulation results for the sequencing of adaptation loops described earlier. These are for Chid27 with the PRBS23 pattern.

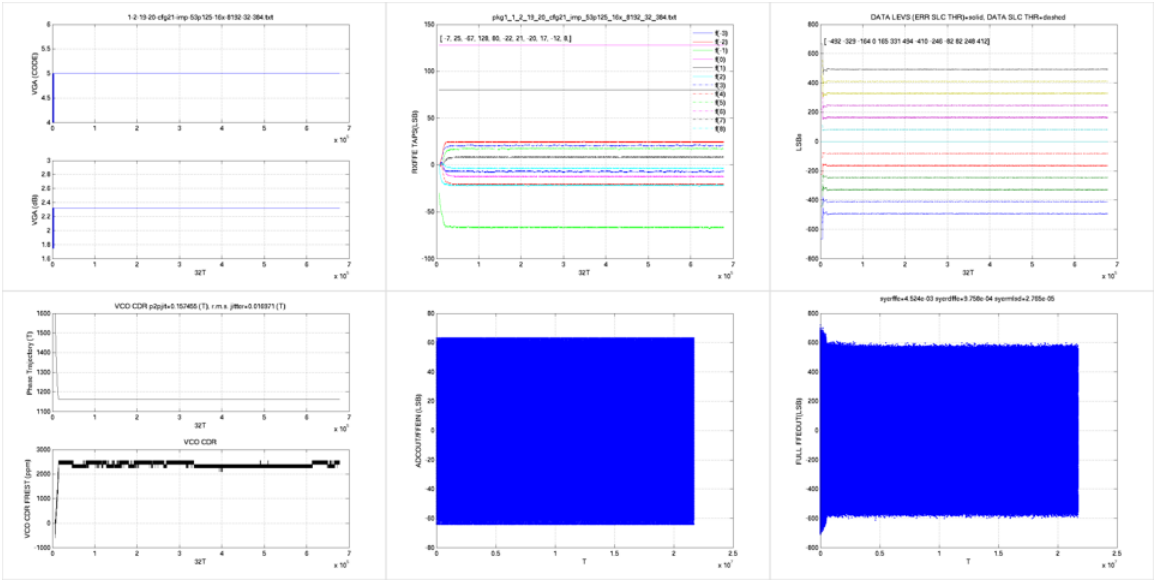


Figure 23: Example simulation results from BASES transient simulator

The panels are as follows. Top row from left to right: vga adaptation in code and dB, RXFFE adaptation in signed LSBs/code, *yl[p,m]#,yl0* in LSBs/code. Bottom row: phase trajectory, frest (ppm), ADC output (LSBs), RXFFE output (LSBS).

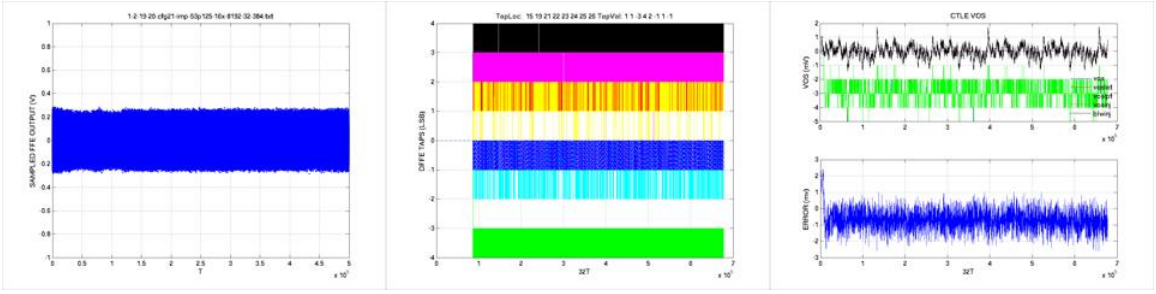


Figure 24: Example simulation results from BASES transient simulator.

In the simulations above the left most panel is the analog voltage at the ADC input. The middle panel shows the convergence of DFFE tap values for fixed (aprior programmed) DFFE tap positions which are noted in the title of the panel. The last plot shows the CTLEVOS loop convergence trajectory (top) and residual error with a 100kHz AC coupling high-pass corner and PRBS23 pattern. The CTLEVOS also compensates for the ADC bias based ADC asymmetric input/output transfer curve.

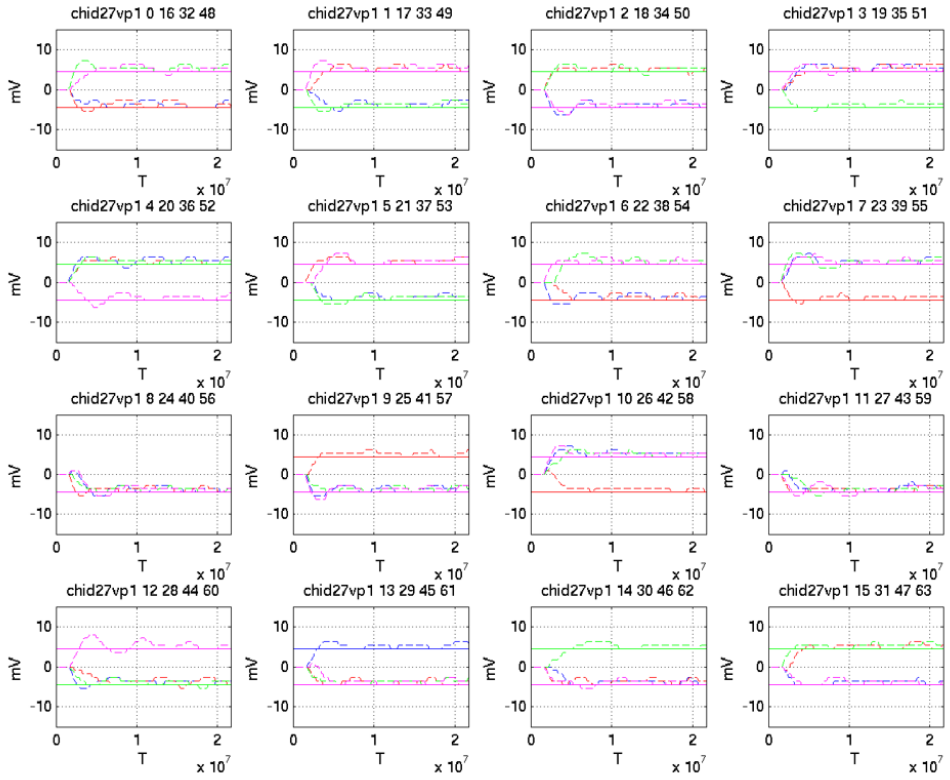


Figure 25: Example ADCVOS convergence results from the BASES transient simulator

The above results show example ADCVOS convergence trajectories across the 64 interleaves for randomly injected static +/- 4.5mV offset into each interleave. Each panel is for the 4 labeled interleaves.

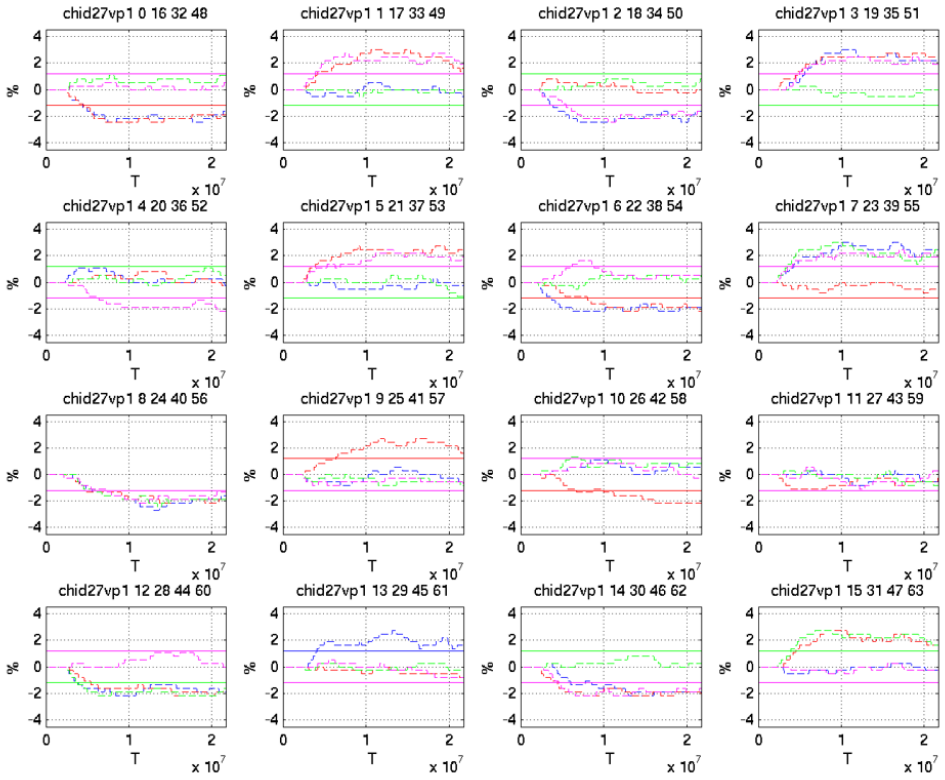


Figure 26: Example ADCGOS convergence results from the BASES transient simulator

The above results show example ADCGOS convergence trajectories across the 64 interleaves for randomly injected static +/- 1.2% offset into each interleave. Each panel is for the 4 labeled interleaves.