Link modeling with 5-tap FFE

January 27, 2016

Contents

1 Scope

This report documents the link response and noise equivalent factor calculations for a five-tap feed forward equalizer (FFE). It follows closely the methods detailed by David Cunningham in his preliminary studies for 32GFC link performance.

2 References

1. ANSI/INCITS TR-60-2015, FC-MSQS-2, Fibre Channel Methodologies for Signal Quality Specification - 2, Annex B "Extending the Link Budget Spreadsheet Model."

2. David Cunningham, T11/11-502v0 "Optical Link Model Modifications Required to Include a Short Equalizer."

3. David Cunningham, T11/12-042v0 "Modifications to the Mode Partition Noise Penalty Calculation for Equalized 32GFC Links."

4. David Cunningham and Richard Johnson, T11/12-123v0 "Expressions for Tap Weights and Noise Equivalent Factor for a FFE(3,T)."

5. David Cunningham, T11/513v11 "32GFC 3T FFE Link Budget Spreadsheet."

6. Richard Johnson, T11/15-425v0 "PAM4 link model studies."

7. Richard Johnson, T11/16-013v1 "Python Script for IEEE GEPBud3_1_16a" (mislabeled as PAM4 link model studies).

3 Parameter normalization

A key parameter for defining link response is the composite link response time (defined from 10% to 90% points) Tc scaled to the unit interval time T. The unit interval time is the inverse of the symbol rate Sr.

Let us calculate typical values for this dimensionless scaling parameter for recent versions of Fibre Channel multimode links:

	16GFC	32GFC	128 GFC	64/256 GFC	Units
Source	T11/12-043v0	T11/12-376v0	T11/15-242v0	proposed	
Chromatic dispersion	24.00	15.86	16.70		picoseconds
Modal dispersion	16.28	10.67	10.91		picoseconds
Transmitter response time	51.23	31.57	30.36		picoseconds
Receiver response time	29.91	16.45	17.50		picoseconds
Composite response time	66.03	40.41	40.32	40.5	picoseconds
Symbol rate	14.025	28.05	28.05	29.45	gigabaud
Pulse width shrinkage	0.12	0.00	0.05	0.05	UI
$Sr \cdot Tc/(1 - PWS)$	1.05	1.13	1.19	1.27	dimensionless

We have increased the effective symbol rate accounting for duty cycle distortion component of deterministic jitter, also known as pulse width shrinkage (PWS), as commonly done in the Excel link budget spreadsheets. For 64GFC we will round up the $Sr \cdot Tc$ product to be 1.30.

In this report, the time axis for eye diagrams is scaled to the unit interval time T, and we choose to put the center of the time coordinate at the center of the eye. Thus the left crossing will be at $t = -\frac{1}{2}$ and the right crossing will be at $t = \frac{1}{2}$.

4 Link response in the absence of receiver equalization

4.1 Single pulse response

By 'unit pulse', we mean a signal stream consisting of an isolated '1' surrounded by a large number of zeros. Quoting equation B16 from Annex B.2 of FC-MSQS-2, the unit pulse response $h(t)$ is given by

$$h(t) = \frac{1}{2} \cdot erf\left[\frac{1.812}{Tc}\left(t + \frac{1}{2}\right)\right] - \frac{1}{2} \cdot erf\left[\frac{1.812}{Tc}\left(t - \frac{1}{2}\right)\right]$$

This differs from equation B.16 because I am assuming that time t and composite response time Tc have been scaled to the unit interval time T.

In the python scripts below I equate the factor 1.812 with $2 \cdot erfinv(0.8)$, as given in equation B.14 of FC-MSQS-2.

$h(t)$ appears in the link budget spreadsheets identified as 'oio' in column AF, rows 40-71.

4.2 NRZ eye in the absence of equalization

A typical NRZ eye is hereby calculated:

```
# Typical NRZ eye in absence of receiver equalization

import numpy as np
import matplotlib.pyplot as plt
from scipy.special import erf,erfinv

def h(t,Tc):
    """Calculates unit pulse response in absence of any equalization"""
    arg = 2.0*erfinv(0.8)/Tc
    return 0.5*erf(arg*(t+0.5))-0.5*erf(arg*(t-0.5))

def NRZ_eye(t,Tc):
    """Routine to plot a typical NRZ eye in the absence of receiver equalization"""
    nUI = 3
    nrz_eye = np.zeros(len(t))
    h_array = np.zeros((len(t),nUI))
    for i in range(nUI):
        h_array[:,i] = h(t+(i-nUI//2)*np.ones(len(t)),Tc)
    nrz_signal = np.zeros((nUI),dtype=np.float64)
    for j in range(2**nUI):
        for k in range(nUI):
            nrz_signal[k] = (j >> (2-k)) & 0x01
        nrz_eye[:] = np.dot(h_array,nrz_signal)
        plt.plot(t,nrz_eye)

t = np.linspace(-0.75,0.75,101)
Tc = 1.134
NRZ_eye(t,Tc)
plt.xlabel('Time (UI)')
plt.ylabel('Amplitude (normalized)')
plt.text(0.0,0.5,'Sr*Tc = 1.134\n(32GFC)',ha='center',va='center')
plt.axis([-0.75,0.75,-0.2,1.2])
plt.savefig('Fig_1_NRZ_no_FFE.pdf')
plt.show()
```

Results of running this script are shown in figure 1 on page 4.

4.3 PAM4 eye in the absence of equalization

A typical PAM4 eye is hereby calculated:

```
# Typical PAM4 eye in absence of receiver equalization

import numpy as np
import matplotlib.pyplot as plt
from scipy.special import erf,erfinv

def h(t,Tc):
    """Calculates unit pulse response in absence of any equalization"""
    arg = 2.0*erfinv(0.8)/Tc
    return 0.5*erf(arg*(t+0.5))-0.5*erf(arg*(t-0.5))

def PAM4_eye(t,Tc):
    """Routine to plot a typical PAM4 eye in the absence of receiver equalization"""
    nUI = 3
```

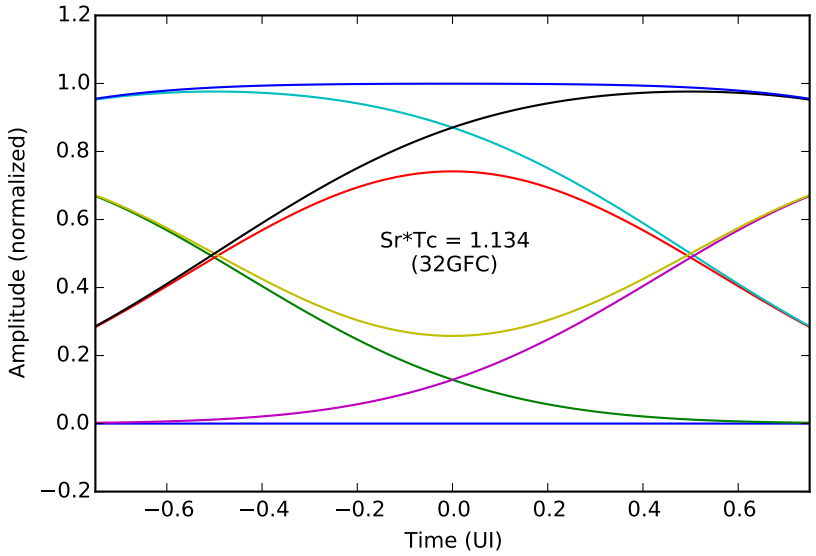

Figure 1: NRZ eye with no receiver equalization

```
        pam4_eye = np.zeros(len(t))
        h_array = np.zeros((len(t),nUI))
        for i in range(nUI):
            h_array[:,i] = h(t+(i-nUI//2)*np.ones(len(t)),Tc)
        pam4_signal = np.zeros((nUI),dtype=np.float64)
        for j in range(4**nUI):
            for k in range(nUI):
                pam4_signal[k] = ((j >> (4-2*k)) & 0x03) / 3.0
            pam4_eye[:] = np.dot(h_array,pam4_signal)
            plt.plot(t,pam4_eye)

t = np.linspace(-0.75,0.75,101)
Tc = 0.9
PAM4_eye(t,Tc)
plt.xlabel('Time (UI)')
plt.ylabel('Amplitude (normalized)')
plt.text(0.0,0.5,'Sr*Tc = 0.9',ha='center',va='center')
plt.axis([-0.75,0.75,-0.2,1.2])
plt.savefig('Fig_2_PAM4_no_eq.pdf')
plt.show()
```

Results of running this script are shown in figure 2 on page 5.

4.4 Power penalty for NRZ and PAM4 in the absence of receiver equalization

The eye opening ISI for NRZ signal is given by equation B.18 in FC-MSQS-2:

$$ISI_{NRZ} = 2 \cdot erf\left[\frac{erfinv(0.8)}{Tc}\right] - 1$$

As sketched in T11/15-425v0, similar expressions can be derived for PAM4:

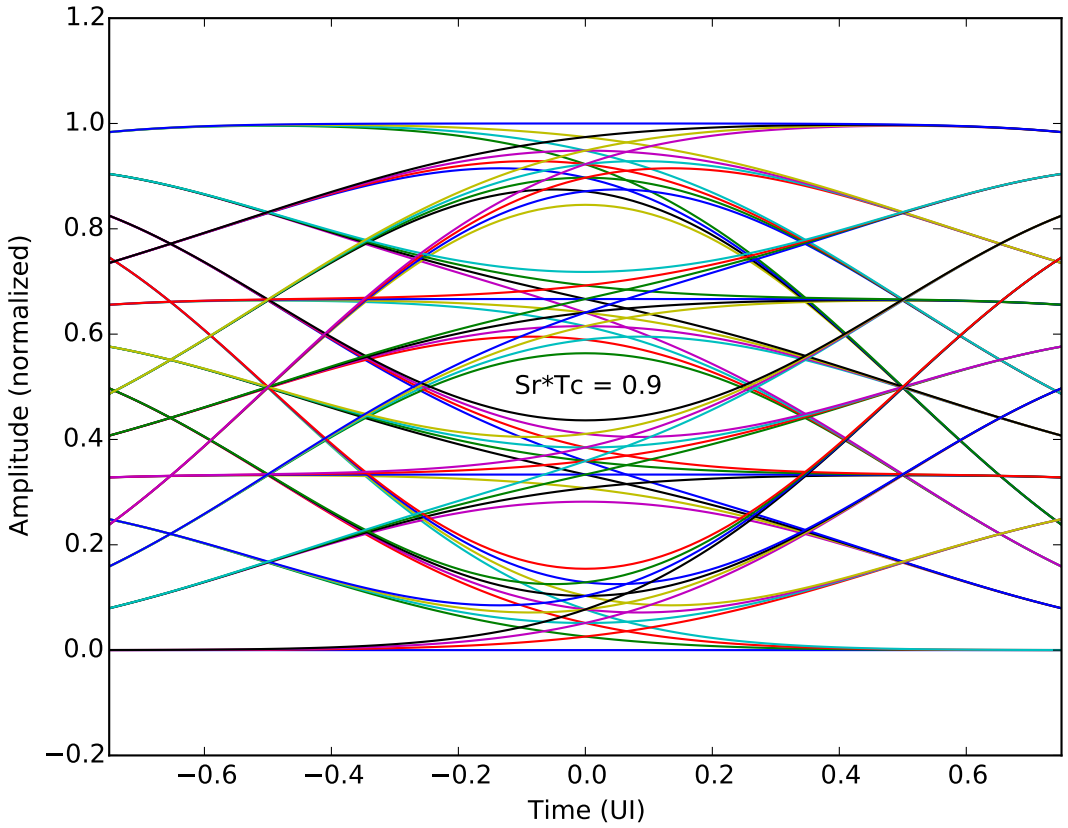

Figure 2: PAM4 eye with no receiver equalization

$$ISI_{PAM4} = \frac{4}{3} \cdot erf\left[\frac{erfinv(0.8)}{Tc}\right] - 1$$

GIven these, we can calculate the power penalties in dB:

$$Penalty = -10 \cdot log_{10}(ISI)$$

Let's plot these two curves, and indicate on the plot the $Sr \cdot Tc$ values for the various recent Fibre Channel links:

```
# plot eye closing penalties for NRZ and PAM4, no receiver equalization

import numpy as np
import matplotlib.pyplot as plt
from scipy.special import erf, erfinv

tpam4 = np.linspace(0.01,1.10,110)
heye = erf(erfinv(0.8)/tpam4)
isi_pam4 = 1.3333333*heye - np.ones(110)
p_pam4 = -10.0*np.log10(isi_pam4)

tnrz = np.linspace(0.01,1.88,188)
```

```
heye = erf(erfinv(0.8)/tnrz)
isi_nrz = 2.0*heye - np.ones(188)
p_nrz = -10.0*np.log10(isi_nrz)

plt.ylim(0.0,20.0)
plt.plot(tpam4,p_pam4,'r-',label='PAM4',lw=2)
plt.plot(tnrz,p_nrz,'b-',label='NRZ',lw=2)
plt.plot([1.134,1.134],[0.0,20.0],'k--')
plt.xlabel('Composite response time scaled to bit period')
plt.ylabel('Power penalty (dB)')
#plt.legend(loc='upper left')
plt.text(1.3,2,"NRZ,\nno equalization")
plt.text(0.15,7,"PAM4,\nno equalization")
plt.text(1.15,8,"32GFC")
plt.savefig('Fig_3_Power_penalties_no_FFE_with_tics.pdf')
plt.show()
```

Results of running this script are shown in figure 3, below. Note from figure 3 that PAM4 eyes are closed for typical link response $Sr{\cdot}Tc$ associated with 64GFC/128GFC links, in the absence of any receiver equalization. As a consequence, we need to include equalization in our link model.

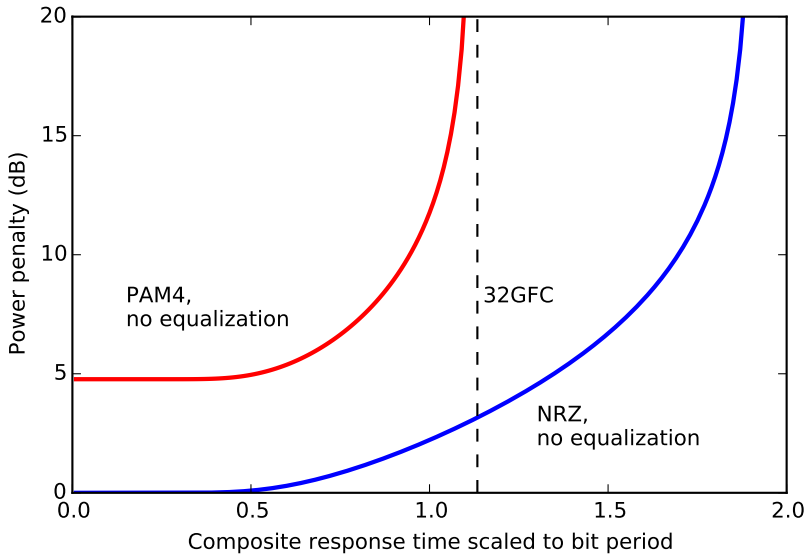

Figure 3: Power penalty, NRZ and PAM4, no receiver equalization

5 Link response with FFE

Our focus in this report is in 5-tap FFE with T/2 tap spacing. However, it will prove to be instructive to review first the 3-tap T-spacing model given by David Cunningham in T11/11-502v0.

5.1 3-tap FFE

Let us consider a 3-tap equalizer with tap spacing T. Let us represent the equalized unit pulse response by $\hat{\mathbf{h}}(t)$:

$$\hat{\mathbf{h}}(t) = \tau_{-1} \cdot h(t-1) + \tau_0 \cdot h(t) + \tau 1 \cdot h(t+1)$$

τ_{-1}, τ_0, and τ_1 are the tap weights, yet to be defined, and I have assumed that time t has been normalized to the unit interval time T.

We wish to choose tap weights such that the equalized unit response is

$$\begin{bmatrix} \hat{\mathbf{h}}(-2) \\ \hat{\mathbf{h}}(-1) \\ \hat{\mathbf{h}}(0) \\ \hat{\mathbf{h}}(1) \\ \hat{\mathbf{h}}(2) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

(Recall that we have normalized our time parameter t such that $h(1)$ means h evaluated at time t = T, the unit interval.)

We only have 3 taps and 5 boundary constraints, so this is an over-constrained problem. We can only approximately achieve the desired boundary conditions.

Let us define a matrix H such that

$$\begin{bmatrix} \hat{h}(-2) \\ \hat{h}(-1) \\ \hat{h}(0) \\ \hat{h}(1) \\ \hat{h}(2) \end{bmatrix} = \begin{bmatrix} h(-1) & h(-2) & h(-3) \\ h(0) & h(-1) & h(-2) \\ h(1) & h(0) & h(-1) \\ h(2) & h(1) & h(0) \\ h(3) & h(2) & h(1) \end{bmatrix} \cdot \begin{bmatrix} \tau_{-1} \\ \tau 0 \\ \tau 1 \end{bmatrix}$$

We identify the 5 row by 3 column matrix as H. The optimum tap weights, as defined by a minimum mean square error (MMSE) metric, are given by

$$\begin{bmatrix} \tau_{-1} \\ \tau 0 \\ \tau 1 \end{bmatrix} = \left(H^T \cdot H\right)^{-1} \cdot \begin{bmatrix} h(1) \\ h(0) \\ h(-1) \end{bmatrix}$$

Because the tap selection is over-constrained, our resulting tap weights can and will change OMA levels, necessitating incorporation of a gain chosen to re-establish correct OMA.

This concludes our summary of the 3-tap analysis.

5.2 5-tap FFE

Let us next consider a 5-tap equalizer with tap spacing $T/2$. The equalized unit pulse response $\hat{h}(t)$ is given by

$$\hat{\mathbf{h}}(t) = \tau_{-2} \cdot h(t-1) + \tau_{-1} \cdot h(t - \frac{1}{2}) + \tau_0 \cdot h(t) + \tau 1 \cdot h(t + \frac{1}{2}) + \tau_2 \cdot h(t+1)$$

We again choose tap weights such that

$$\begin{bmatrix} \hat{\mathbf{h}}(-2) \\ \hat{\mathbf{h}}(-1) \\ \hat{\mathbf{h}}(0) \\ \hat{\mathbf{h}}(1) \\ \hat{\mathbf{h}}(2) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

Note that now we have just enough tap weights to match our desired boundary conditions. This is no longer an over-constrained problem. In particular, this means that we no longer need to define a gain to maintain correct OMA.

Following the same process as the 3-tap analysis, we define a 5 row by 5 column matrix H such that

$$\begin{bmatrix} \hat{h}(-2) \\ \hat{h}(-1) \\ \hat{h}(0) \\ \hat{h}(1) \\ \hat{h}(2) \end{bmatrix} = \begin{bmatrix} h(-1) & h(-\frac{3}{2}) & h(-2) & h(-\frac{5}{2}) & h(-3) \\ h(0) & h(-\frac{1}{2}) & h(-1) & h(-\frac{3}{2}) & h(-2) \\ h(1) & h(\frac{1}{2}) & h(0) & h(-\frac{1}{2}) & h(-1) \\ h(2) & h(\frac{3}{2}) & h(1) & h(\frac{1}{2}) & h(0) \\ h(3) & h(\frac{5}{2}) & h(2) & h(\frac{3}{2}) & h(1) \end{bmatrix} \cdot \begin{bmatrix} \tau_{-2} \\ \tau_{-1} \\ \tau 0 \\ \tau_1 \\ \tau 2 \end{bmatrix}$$

The desired tap solution is then given by

$$\begin{bmatrix} \tau_{-2} \\ \tau_{-1} \\ \tau_0 \\ \tau_1 \\ \tau_2 \end{bmatrix} = \left(H^T \cdot H\right)^{-1} \cdot \begin{bmatrix} h(1) \\ h(\frac{1}{2}) \\ h(0) \\ h(-\frac{1}{2}) \\ h(-1) \end{bmatrix}$$

5.3 Unit pulse profile with 5-tap FFE

Let us plot the resulting unit pulse response after passing through the 5-tap FFE:

```
# Plotting the unit pulse response after processing by a 5-tap filter
#  Tc = 1.3 time unit pulse period

import numpy as np
import matplotlib.pyplot as plt
from scipy.special import erf, erfinv
def h(t,Tc):
    """Calculates unit pulse response in absence of any equalization"""
    arg = 2.0*erfinv(0.8)/Tc
    return 0.5*erf(arg*(t+0.5))-0.5*erf(arg*(t-0.5))

def tap_calc(Tc):
    """Calculates optimum tap weights for 5-tap FFE"""
    tau = np.zeros(5)
    rowstore = np.zeros(13,dtype=np.float64)
    for i in range(13):
        rowstore[i] = h(0.5*(i-6),Tc)
    H5 = np.zeros((5,5),dtype=np.float64)
    for row in range(5):
        H5[row,:]=rowstore[2*row:2*row+5]
    tau = np.dot(np.linalg.inv(np.dot(H5.T,H5)),rowstore[4:9])
    return tau

def heq(t,Tc):
    """Calculates unit pulse response with 5-tap equalization"""
    tau = tap_calc(Tc)
    hhat = np.zeros(len(t))
    for i in range(5):
        tshift = 0.5*float(i-2)*np.ones(len(t))
        hhat += tau[i]*h(t-tshift,Tc)
    return hhat

t = np.linspace(-4.0,4.0,101)
Tc = 1.3
plt.plot(t,heq(t,Tc))
plt.grid()
plt.xlabel('Time (UI)')
plt.ylabel('Amplitude (normalized)')
plt.text(0.0,0.1,'Sr*Tc = 1.3',ha='center',va='center')
plt.axis([-4.0,4.0,-0.4,1.2])
plt.savefig('Fig_4_Unit profile equalized 1p2.pdf')
plt.show()
```

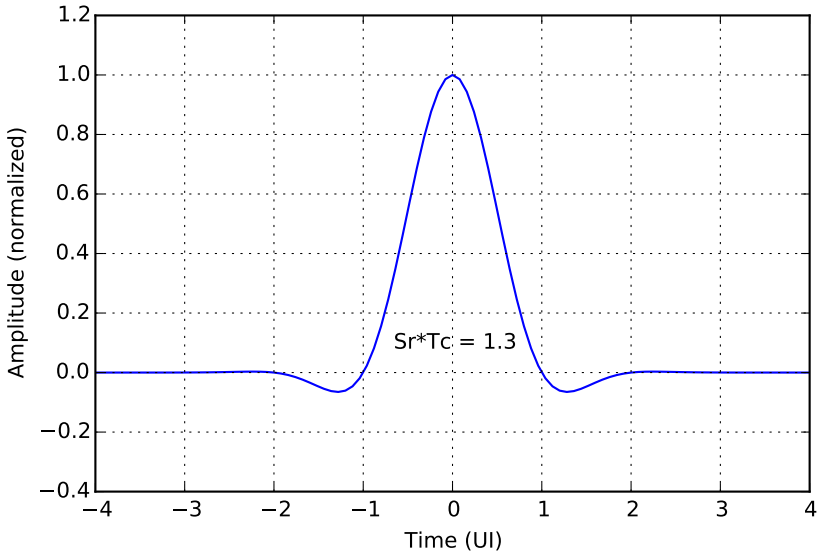

Figure 4: Power penalty, NRZ and PAM4, no receiver equalization

Results of running this script are shown in figure 4, above.

Let's repeat this analysis, but this time using a very much more sluggish composite link response time $Sr \cdot Tc = 3$. This is a much more sluggish response than we would ever consider for a reasonable link, but it will more clearly exhibit two interesting aspects of the FFE model.

```
# Plotting the unit pulse response after processing by a 5-tap filter
#  Tc = 3 time unit pulse period

import numpy as np
import matplotlib.pyplot as plt
from scipy.special import erf, erfinv
import math

def h(t,Tc):
    """Calculates unit pulse response in absence of any equalization"""
    arg = 2.0*erfinv(0.8)/Tc
    return 0.5*erf(arg*(t+0.5))-0.5*erf(arg*(t-0.5))

def tap_calc(Tc):
    """Calculates optimum tap weights for 5-tap FFE"""
    tau = np.zeros(5)
    rowstore = np.zeros(13,dtype=np.float64)
    for i in range(13):
        rowstore[i] = h(0.5*(i-6),Tc)
    H5 = np.zeros((5,5),dtype=np.float64)
    for row in range(5):
        H5[row,:]=rowstore[2*row:2*row+5]
    tau = np.dot(np.linalg.inv(np.dot(H5.T,H5)),rowstore[4:9])
    return tau

def heq(t,Tc):
    """Calculates unit pulse response with 5-tap equalization"""
```

```
    tau = tap_calc(Tc)
    hhat = np.zeros(len(t))
    for i in range(5):
        tshift = 0.5*float(i-2)*np.ones(len(t))
        hhat += tau[i]*h(t-tshift,Tc)
    return hhat

t = np.linspace(-4.0,4.0,101)
Tc = 2.5
plt.plot(t,heq(t,Tc))
plt.grid()
plt.show()
```

Results of running this script are shown in figure 5, below. Notice how this profile goes through zero for $\hat{\mathbf{h}}(-2)$, $\hat{\mathbf{h}}(-1)$, $\hat{\mathbf{h}}(1)$, and $\hat{\mathbf{h}}(2)$, and goes to one for $\hat{\mathbf{h}}(0)$, as required by our boundary conditions, but $\hat{\mathbf{h}}(-3)$ and $\hat{\mathbf{h}}(3)$ do not go to zero. Thus to model eye closing we need to study data sequences that extend from -3 UI through 3 UI inclusive, a sequence of 7 symbols. For PAM4 with 4 levels per UI, this corresponds to $4^7 = 16384$ discrete data patterns.

The second observation to be drawn from figure 5 is that the boundary conditions are achieved only at the center of the eye. As we move away from the center, the unit pulse profile deviates ever more from our desried ideal response. This helps to explain the unique shape of the eye, the next topic that we consider in this report.

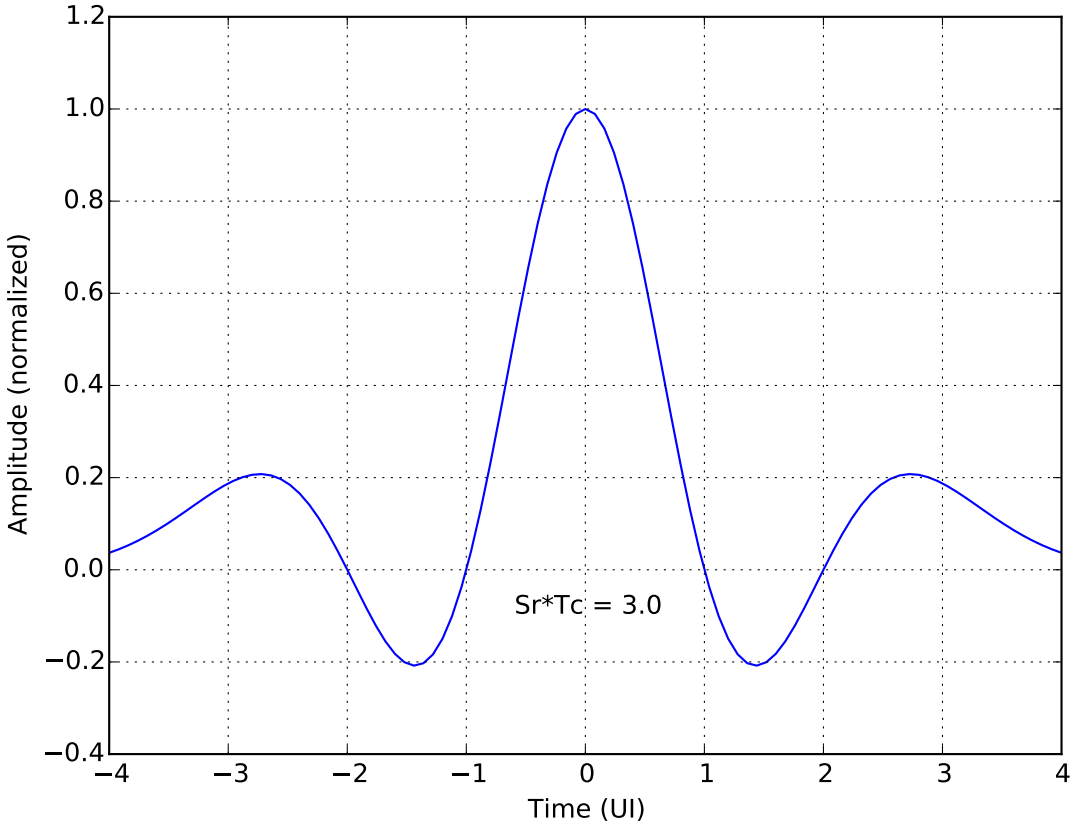

Figure 5: Power penalty, NRZ and PAM4, no receiver equalization

5.4 Typical PAM4 eye

Let us plot a typical PAM4 eye:

```
# Plot representative PAM4 eyes with 5-tap equalizer

import numpy as np
import matplotlib.pyplot as plt
from scipy.special import erf, erfinv

def h(t,Tc):
    """Calculates unit pulse response in absence of any equalization"""
    arg = 2.0*erfinv(0.8)/Tc
    return 0.5*erf(arg*(t+0.5))-0.5*erf(arg*(t-0.5))

def tap_calc(Tc):
    """Calculates optimum tap weights for 5-tap FFE"""
    tau = np.zeros(5)
    rowstore = np.zeros(13,dtype=np.float64)
    for i in range(13):
        rowstore[i] = h(0.5*(i-6),Tc)
    H5 = np.zeros((5,5),dtype=np.float64)
    for row in range(5):
        H5[row,:]=rowstore[2*row:2*row+5]
    tau = np.dot(np.linalg.inv(np.dot(H5.T,H5)),rowstore[4:9])
    return tau

def heq(t,Tc):
    """Calculates unit pulse response with 5-tap equalization"""
    tau = tap_calc(Tc)
    hhat = np.zeros(len(t))
    for i in range(5):
        tshift = 0.5*float(i-2)*np.ones(len(t))
        hhat += tau[i]*h(t-tshift,Tc)
    return hhat

def PAM4_eye(t,Tc):
    """Plot typical PAM4 eye"""
    nUI = 7
    hhat_array = np.zeros((len(t),nUI),dtype=np.float64)
    pam4_eye = np.zeros((len(t),1),dtype=np.float64)
    for i in range(nUI):
        hhat_array[:,i] = heq(t+(i-nUI//2)*np.ones(len(t)),Tc)
    PAM4_signal = np.zeros((nUI,1),dtype=np.float64)
    for j in range(4**nUI):
        for k in range(nUI):
            PAM4_signal[k,0] = ((j >> (12-2*k)) & 0x03) / 3.0
        pam4_eye[:] = np.dot(hhat_array,PAM4_signal)
        plt.plot(t,pam4_eye)

t = np.linspace(-0.75,0.75,101)
Tc = 1.3
PAM4_eye(t,Tc)
plt.xlabel('Time (UI)')
plt.ylabel('Amplitude (normalized)')
```

```
plt.text(0.0,0.5,'Sr*Tc = 1.3',ha='center',va='center')
plt.axis([-0.75,0.75,-0.2,1.2])
plt.savefig('Fig_6_PAM4 eyes FFE 1p2.png')
plt.show()
```

Results of running this script are shown in figure 6, below. (Caution! This python script will take several minutes to run.)

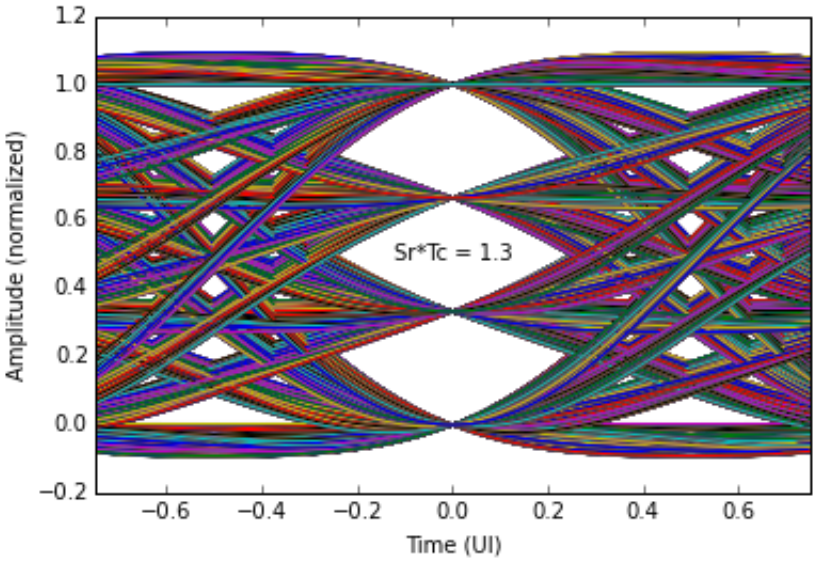

Figure 6: Power penalty, NRZ and PAM4, no receiver equalization

Note the typical diamond shape of the equalized eye, and the much faster eye closing with timing errors compared with an NRZ eye. For $Sr \cdot Tc = 1.3$, the horizontal tip-to-tip interval is on the order of 0.5 UI.

5.5 Power penalty with PAM4 eye closing

Let us next calculate the power penalty associated with PAM4 eye closing as a function of increasing composite link response times Tc:

```
# Calculate power penalty due to eye closing for PAM4 eyes with 5-tap equalizer.

import numpy as np
import matplotlib.pyplot as plt
from scipy.special import erf, erfinv
import math

def h(t,Tc):
    """Calculates unit pulse response in absence of any equalization"""
    arg = 2.0*erfinv(0.8)/Tc
    return 0.5*erf(arg*(t+0.5))-0.5*erf(arg*(t-0.5))

def tap_calc(Tc):
    """Calculates optimum tap weights for 5-tap FFE"""
    tau = np.zeros(5)
    rowstore = np.zeros(13,dtype=np.float64)
    for i in range(13):
```

```
            rowstore[i] = h(0.5*(i-6),Tc)
        if Tc < 0.38:
            # only need gain adjust with single tap
            tau[2] = 1.0/rowstore[6]
        elif Tc < 0.7:
            # only need 3 taps
            H3 = np.zeros((3,3),dtype=np.float64)
            for row in range(3):
                H3[row,:]=rowstore[2*row+3:2*row+6]
            tau[1:4] = np.dot(np.linalg.inv(np.dot(H3.T,H3)),rowstore[5:8])
        else:
            # we will use all 5 taps
            H5 = np.zeros((5,5),dtype=np.float64)
            for row in range(5):
                H5[row,:]=rowstore[2*row:2*row+5]
            tau = np.dot(np.linalg.inv(np.dot(H5.T,H5)),rowstore[4:9])
        return tau

    def heq(t,Tc):
        """Calculates unit pulse response with 5-tap equalization"""
        tau = tap_calc(Tc)
        try:
            # t is a numpy vector containing several values
            n = len(t)
            hhat = np.zeros(n)
            for i in range(5):
                tshift = 0.5*float(i-2)*np.ones(n)
                hhat += tau[i]*h(t-tshift,Tc)
        except TypeError:
            # t is a single float number
            hhat = 0.0
            for i in range(5):
                tshift = 0.5*float(i-2)
                hhat += tau[i]*h(t-tshift,Tc)
        return hhat

    def PAM4_eye_crossings(tshift,Tc):
        """Calculate crossing levels for PAM4 with 5-tap equalizer for specific timing location
        tshift and for specific composite link response time Tc"""
        nUI = 7
        hhat_array = np.zeros(nUI,dtype=np.float64)
        pam4_eye_crossings = np.zeros(4**nUI,dtype=np.float64)
        for i in range(nUI):
            hhat_array[i] = heq(tshift+(i-nUI//2),Tc)
        PAM4_signal = np.zeros(nUI,dtype=np.float64)
        for j in range(4**nUI):
            for k in range(nUI):
                PAM4_signal[k] = ((j >> (12-2*k)) & 0x03) / 3.0
            pam4_eye_crossings[j] = np.dot(hhat_array,PAM4_signal)
        return pam4_eye_crossings

    def pam4_isi_calc(tshift,Tc):
        """Calculates the linear eye opening isi for PAM4 signals with 5-tap FFE"""
        pam4_eye_crossings = PAM4_eye_crossings(tshift,Tc)
```

```
    b = np.sort(pam4_eye_crossings)
    c = np.sort(b - np.roll(b,1))
    isi = c[len(c)-1]
    return isi

tc = np.linspace(0.024,2.4,101)
pp = np.zeros(101)
tshift = 0.0
for i in range(101):
    pp[i] = -10.0*math.log10(pam4_isi_calc(tshift,tc[i]))
plt.plot(tc,pp)
plt.plot([1.3,1.3],[4.0,6.0])
plt.text(1.2,7.0,'Proposed 64GFC & 256GFC link response',ha='center')
plt.xlabel('Sr*Tc product (dimensionless)')
plt.ylabel('Power penalty (dB)')
plt.savefig('Fig_7_Power penalty FFE.pdf')
plt.show()
```

Results of running this script are shown in figure 7, below. (Caution! This python script wiill take a couple of minutes to run.) As can bee seen, for $Sr \cdot Tc = 1.3$ we still have some margin, and if necessary can relax temporal response parameters without suffering from eye closure.

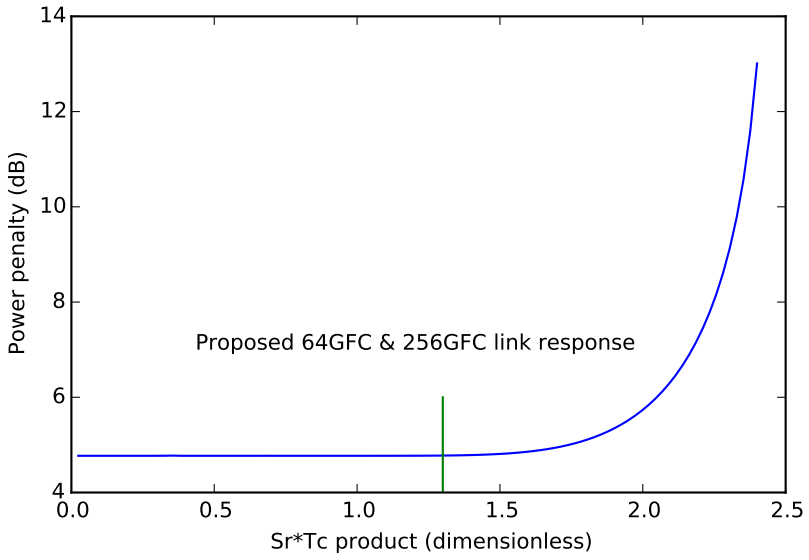

Figure 7: Power penalty, NRZ and PAM4, no receiver equalization

5.6 Noise equivalent factor (NEF)

Next we need to calculate how much the FFE extends the frequency spectrum, and the consequence on laser residual intensity noise (RIN). Let us quote from annex B.4 of FC-MSQS-2 some key equations:

$$\sigma_{rin} = \frac{n_{rin}}{s/2}$$

$$n_{rin}^2 = W_0 \int_{-\infty}^{\infty} df \, |I(f)|^2 \cdot |G(f)|^2$$

$I(f)$ is the frequency spectrum of the composite Gaussian link response in the absence of FFE, and $G(f)$ is the change in spectral response induced by the equalizer. From Table B.3,

$$I(f) = \exp\left(-3.005\, T_c^2 \cdot f^2\right)$$

Generalizing equation B.52 for our 5-tap filter,

$$G(f) = \tau_0 + 2\tau_1 \cos(\pi f T) + 2\tau_2 \cos(2\pi f T)$$

Let us compare plots of these two:

```
# calculating spectral response with and without equalizer

import numpy as np
from scipy.special import erf, erfinv
import matplotlib.pyplot as plt
from math import pi

def h(t,Tc):
    """Calculates unit pulse response in absence of any equalization"""
    arg = 2.0*erfinv(0.8)/Tc
    return 0.5*erf(arg*(t+0.5))-0.5*erf(arg*(t-0.5))

def tap_calc(Tc):
    """Calculates optimum tap weights using MMSE"""
    tau = np.zeros(5)
    rowstore = np.zeros(13,dtype=np.float64)
    for i in range(13):
        rowstore[i] = h(0.5*(i-6),Tc)
    H5 = np.zeros((5,5),dtype=np.float64)
    for row in range(5):
        H5[row,:]=rowstore[2*row:2*row+5]
    tau = np.dot(np.linalg.inv(np.dot(H5.T,H5)),rowstore[4:9])
    return tau

def I2(f,Tc):
    arg = 0.5*(pi*Tc*f/erfinv(0.8))**2
    return np.exp(-arg)

def G2(f,tau):
    return (tau[2]+2.0*tau[3]*np.cos(pi*f)+2.0*tau[4]*np.cos(2.0*pi*f))**2

def I2G2(f,Tc,tau):
    return I2(f,Tc)*G2(f,tau)

Tc = 1.3
tau = tap_calc(Tc)
freq = np.linspace(0.0,1.0,101)
plt.plot(freq,I2(freq,Tc))
plt.plot(freq,I2G2(freq,Tc,tau))
plt.xlabel('Frequency scaled to symbol rate')
plt.ylabel('Power spectrum')
plt.text(0.15,0.4,'No FFE',color='blue')
plt.text(0.6,0.6,'with 5-tap FFE',color='green')
plt.savefig('Fig_8_Power spectrum comparison.pdf')
plt.show()
```

Results of running this script are shown in figure 8, below.

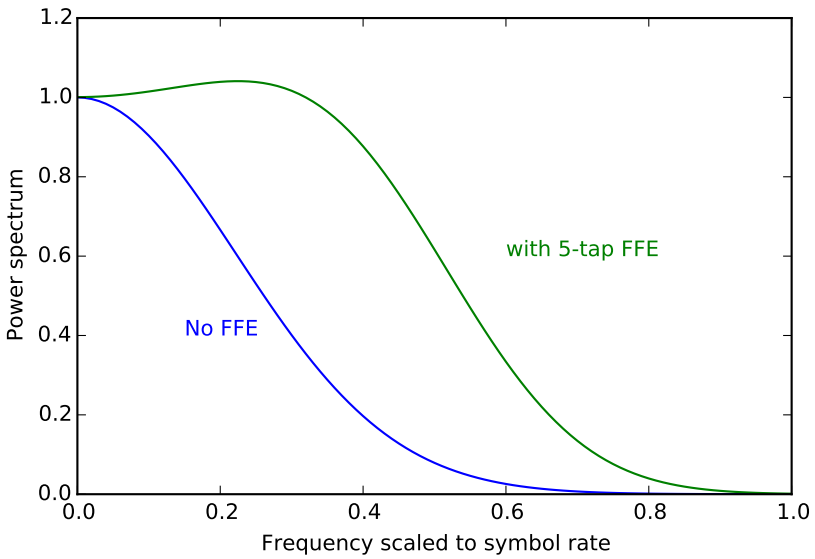

Figure 8: Power penalty, NRZ and PAM4, no receiver equalization

Next let us take the ratio of the area under the curves, which we will call the Noise Equivalent Factor (NEF):

```
import numpy as np
from scipy.special import erf, erfinv
from scipy.integrate import quad
from math import pi, exp, cos

def h(t,Tc):
    """Calculates unit pulse response in absence of any equalization"""
    arg = 2.0*erfinv(0.8)/Tc
    return 0.5*erf(arg*(t+0.5))-0.5*erf(arg*(t-0.5))

def tap_calc(Tc):
    """Calculates optimum tap weights using MMSE"""
    tau = np.zeros(5)
    rowstore = np.zeros(13,dtype=np.float64)
    for i in range(13):
        rowstore[i] = h(0.5*(i-6),Tc)
    H5 = np.zeros((5,5),dtype=np.float64)
    for row in range(5):
        H5[row,:]=rowstore[2*row:2*row+5]
    tau = np.dot(np.linalg.inv(np.dot(H5.T,H5)),rowstore[4:9])
    return tau

def I2(f,Tc):
    arg = 0.5*(pi*Tc*f/erfinv(0.8))**2
    return exp(-arg)
```

```
def G2(f,tau):
    return (tau[2]+2.0*tau[3]*cos(pi*f)+2.0*tau[4]*cos(2.0*pi*f))**2

def I2G2(f,Tc,tau):
    return I2(f,Tc)*G2(f,tau)

Tc = 1.3
tau = tap_calc(Tc)
area1 = quad(I2,-np.inf,np.inf,args=(Tc))       # absencee of equalization
area2 = quad(I2G2,-np.inf,np.inf,args=(Tc,tau)) # with 5-tap FFE
NEF = area2[0] / area1[0]
print('{0:.3f}'.format(NEF))
```

This script when run will produce no plots, but rather will display in a text terminal the following NEF: 2.007.

6 Combining python and excel link models

The final link model needs to be in the form of an Excel spreadsheet to facilitate sharing with the greatest number of users. In this section we explore proposals for combining the strength of python with the convenience of excel.

The first question to ask is whether a 5-tap FFE is needed? We have a 3-tap spreadsheet in T11/11-502v0. This can be modified to model PAM4 signals. However, the following eye diagram shows that a 3 tap FFE is not quite sufficient when considering PAM4 signals and $Sr \cdot Tc = 1.3$: David Cunningham was

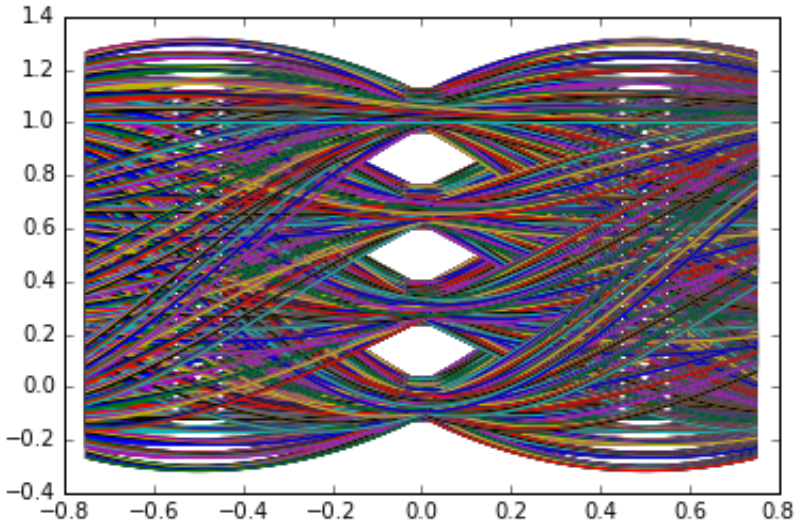

Figure 9: Eye diagram for PAM4 modulation with 3-tap FFE

successful in getting good performance from a 3-tap FFE, so why is this less than perfect? One reason is that PAM4 signals appear to be more prone to eye closing than NRZ. A second reason is that we assume a higher $Sr \cdot Tc$ value because of different FEC and we have included a small amount of pulse width shrinkage.

If a 3-tap FFE is not an option, then how can we best combine the analytical strength and flexibility of python with the convenience of excel? The next proposal is to have python calculate the most tedious and time-consuming aspects of the PAM4 - 5-tap FFE model and output them as an Excel table which can

then be imported into the link budget spreadsheet. The following program listing does just that. It uses the openpyxl library module which teaches python how to read and write excel spreadsheets.

```
# start of file Excel_link_model_inputs.py

from openpyxl import Workbook
import numpy as np
from scipy.special import erf, erfinv
from math import exp

def h(t,Tc):
    """Calculates unit pulse response in absence of any equalization"""
    arg = 2.0*erfinv(0.8)/Tc
    return 0.5*erf(arg*(t+0.5))-0.5*erf(arg*(t-0.5))

def tap_calc(Tc):
    """Calculates optimum tap weights using MMSE"""
    tau = np.zeros(5)
    rowstore = np.zeros(13,dtype=np.float64)
    for i in range(13):
        rowstore[i] = h(0.5*(i-6),Tc)
    H5 = np.zeros((5,5),dtype=np.float64)
    for row in range(5):
        H5[row,:]=rowstore[2*row:2*row+5]
    tau = np.dot(np.linalg.inv(np.dot(H5.T,H5)),rowstore[4:9])
    return tau

def NEF(Tc):
    """Calculate the Noise Equivalent Factor, the area under the power
    spectrum"""
    tau = tap_calc(Tc)
    b = 0.5*(erfinv(0.8)/Tc)**2
    p0 = (tau[2]**2 + 2.0*tau[3]**2 + 2.0*tau[4]**2)
    p1 = 4.0*tau[3]*(tau[2]+tau[4])*exp(-b)
    p2 = (4.0*tau[2]*tau[4]+2.0*tau[3]**2)*exp(-4.0*b)
    p3 = 4.0*tau[3]*tau[4]*exp(-9.0*b)
    p4 = 2.0*tau[4]**2*exp(-16.0*b)
    return p0+p1+p2+p3+p4

rootdir = '/home/rvj/Projects/Python/PAM4/Excel transfer/'
wb = Workbook()
ws = wb.active
ws['A1'] = 'Sr*Tc'
ws['B1'] = 'Tap 0'
ws['C1'] = 'Tap 1'
ws['D1'] = 'Tap 2'
ws['E1'] = 'NEF'

for iTc in range(61):
    Tc = 0.9 + 0.01*float(iTc)
    tau = tap_calc(Tc)
    nef = NEF(Tc)
    ws.append([Tc,tau[2],tau[3],tau[4],nef])
```

```
wb.save(rootdir+'Excel_Python_inputs.xlsx')
```

This script produces no plots, but rather generates an excel spreadsheet. See the zip file that is a companion to this document for this spreadsheet.

The final topic in this subsection is to identify the data patterns that correspond to the innermost eyes in the PAM4 eye diagram. This is given in figure 10.

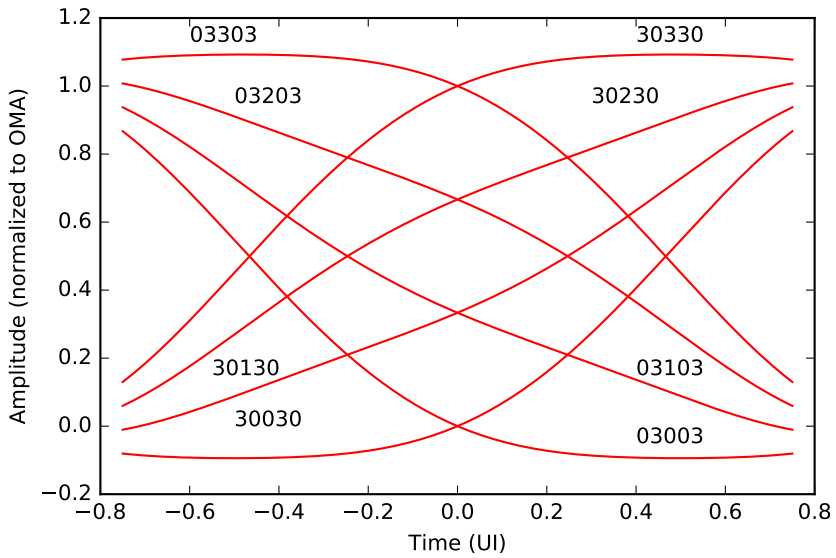

Figure 10: Data patterns corresponding to the innermost eyes for PAM4 with 5-tap FFE

7 Concluding remarks

- A detailed mathematical model of PAM4 link response with a 5-tap Feed Forward Equalizer (FFE) has been presented.
- Python scripts analyzing various aspects of the link model have been listed.
- Plots generated by the python scripts have been displayed as figures 1-9.
- PAM4 eyes with 5-tap FFE exhibit a diamond shape typical of FFE links.
- The tip-to-tip horizontal eye opening is on the order of 0.5 UI for the response times under consideration for 64GFC/256GFC.
- A 3-tap FFE has been found to be inadequate for PAM4 signals in a 64GFC/256GFC link.
- Alternatives for incorporating the python analysis into an Excel link budget spreadsheet have been reviewed. It is recommended to have python do the most intensive calculations and output the results to an Excel file which can then be copied and pasted into the link budget spreadsheet.