JOINT, FRACTIONAL RESAMPLER WITH DELAY EQUALIZATION FOR HIGH SYNCHRONIZATION ACCURACY WITH A REDUCED NUMBER OF SAMPLES PER SYMBOL

Ghassan Maalouli, General Dynamics, Scottsdale, AZ.
Don R. Stephens, CommLargo Inc.

ABSTRACT

Software defined radios typically have asynchronous digital sampling relative to the received symbol rates. This architecture allows the same RF front end and digitization to be used for many waveforms and symbol rates, but requires the demodulator to generate symbol-based samples from the asynchronous input samples. In this paper, we propose an innovative method for resampling signals with a low number of samples per symbol. A method for jointly removing timing delay and timing drift is also provided. This structure is highly efficient, requires a low number of taps and achieves superior real-time performance.

1 Theoretical Derivation

Following the development in [1-2], we model the resampling problem as a conversion of the discrete input samples back into the analog domain and then resampling at the desired timing instants. In analog-to-digital conversion, a reconstruction filter interpolates the samples and is typically implemented as a *sinc function*. Our resampler presumes the existence of a hypothetical analog to digital converter that changes the signal into an analog signal then samples the analog signal at the desired sample rate. Assuming that the signal *x(n)* is sampled at a rate T_s. The analog signal at the output of an *A/D* is given by:

$$y(t) = \sum_m x(mT_s).h_I(t - mT_s) \qquad \textbf{Eq. 1}$$

Where h_I is the reconstruction filter with a *sinc* impulse response. If this signal is now sampled at a new sample rate T_i, the new signal can be represented as:

$$y(kT_i) = \sum_m x(mT_s).h_I(kT_i - mT_s) \qquad \textbf{Eq. 2}$$

To describe the indexing between input and output samples we must define the following variables:

$$m_k = floor\left(\frac{kT_I}{T_s}\right) \qquad \textbf{Eq. 3}$$

where m_k is the basepoint timing index.

$$\mu_k = \mathrm{mod}\left(\frac{kT_I}{T_s}\right)_1 \qquad \textbf{Eq. 4}$$

μ_k is the fractional timing index which is bounded to [0,1]

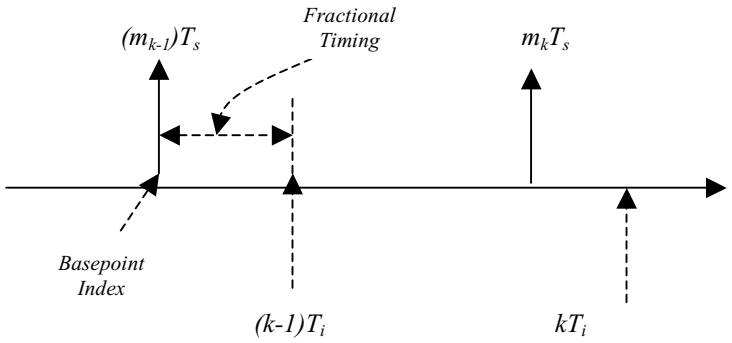
These variables allow us to write the input signal as:

$$y(kT_i) = y(m_k + \mu_k) = \sum_m x(mT_s).h_I(kT_i - mT_s) = \sum_i x\left[(m_k - i)T_s\right].h_I\left[(i + \mu_k)T_s\right] \qquad \textbf{Eq. 5}$$

Where $i = m_k - m$ is the filter index and runs over the number of taps in the filter.

The timing relationship between input and output can be illustrated as shown in Figure 1.

Figure 1: Resampler Input/Output Timing Relationship



1.1 Delay & Timing Drift Equalization

Equation 5 does not consider delay equalization. We extend the capability of the resampler with delay equalization which provides joint resampling and symbol timing correction. A timing shift in the signal can be considered a shift in the filter impulse response provided the system is LTI which is the underlying assumption for the preceding derivation. Consider a received signal, *r(t)*. Taking propagation delay into account, the signal can be represented as:

$$r(t) = \sum x(t-\tau).e^{j\omega_c(t-\tau)}.h(t-mT) \quad \textbf{(5)}$$

Let $u = t-\tau$. We rewrite (5) as

$$r(t) = \sum x(u).e^{j\omega_c(u)}.h(u+\tau-mT) \quad \textbf{(6)}$$

Equation (6) indicates the delay in the channel can be expressed in terms of a delayed filter impulse response. The missing step is the estimation of the delay factor to use in order to equalize the channel delay. However, with the signal model that we have developed, not only we can track fixed channel delay but we can additionally track timing drift between the input sample clock and the output sample clock. We model the fixed delay as an addition of a fixed quantity to the fractional timing and is described as the zero order error. The drift between the input and output clock is modeled by the addition of a fixed quantity to the ratio Ts/Ti. Estimating the delay and drift can be done via the use of a generic timing error detector [3]. A maximum likelihood approximation of a timing detector is an early gate late gate with a loop filter. In a classical design, the early/late gate circuit provides a zero order estimate for timing error by advancing or retarding the perceived correct starting point of the symbol. In a sampled system, this is done by referencing the sample index of the symbol's leading edge. The error estimate from the early/late gate is filtered with the loop filter. As a state space model, the filter provides both a zero order estimate representing fixed delay and a first order estimate representing clock drift. These quantities are then used to compute an adjusted basepoint timing reference, m_k, and an adjusted fractional timing reference, μ. μ provides the correction into the interpolation filter to compute the output samples. Sinc filter coefficients must be specifically computed to interpolate the input samples at the desired timing offsets. Lookup tables can provide the coefficients in real-time implementations..

To facilitate the understanding of this design, we introduce the system block diagram.

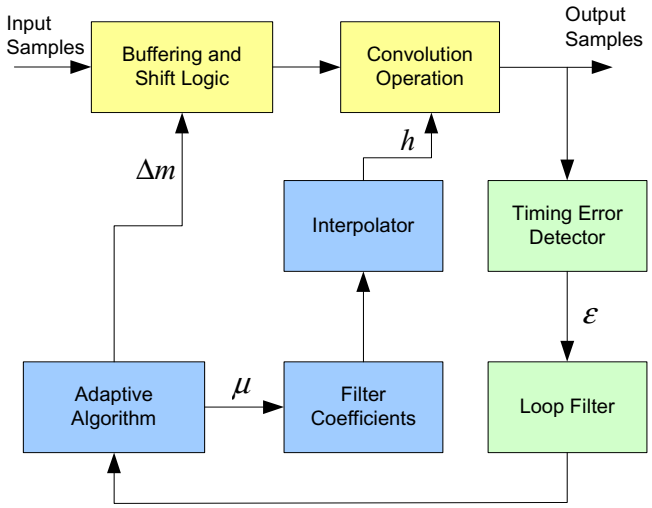


Figure 2: Resampler with Delay Equalization Block Diagram

The buffering and shift logic (BSL) block receives complex envelope samples at the input sample rate. A control input is Δm, which specifies the number of samples to shift into the interpolator filter. There are only two possibilities for Δm to control the sample shift; either no shift if Δm is zero or a Δm shift if Δm is greater than zero. Δm and μ are computed by the adaptive algorithm. If Δm equals zero, then the BSL will not shift any new samples into the filter structure before computing a new sample. If Δm is greater than zero, the shift logic will shift Δm samples into the filter structure before an output is computed. The details of the computation of Δm &μ are listed in the Adaptive algorithm description. Computation of the interpolator output is listed in the convolution block.

1.2 Adaptive Algorithm

The adaptive algorithm generates updates for Δm &μ to control input samples and delay adjustments. Δm controls samples shifted into the filter structure as described earlier. μ however, functions as a delay to derive the filter coefficients. The adaptive algorithm is divided into two steps: initialization and run-time operation.

1.3 Initialization

Parameters for the adaptive algorithm are initialized in the following fashion.

1) Compute the ratio of the input sample rate to the desired output sample rate.

$$W = \frac{T_i}{T_s} = \frac{F_s}{F_i}$$

Where T_s is the input sample rate and T_i is the output sample rate. Note that these are sample rates and not symbol rates.

2) Initialize: $m(0) = \mu(0) = 0$.

1.4 Resampling Operation

To produce output samples, we must compute μ and Δm which include corrections for timing delay and timing drift. The timing delay estimate and the timing drift estimate are derived from the filtered error estimate. The adaptation proceeds as follows.

Update the basepoint timing index. Include the Timing drift and the timing delay estimates from the timing error detector.

$$m(k+1) = m(k) + \left\lfloor \mu(k) + \frac{T_i}{T_s} + n.\dot{\tau} + \tau \right\rfloor \text{ where}$$

$\lfloor \bullet \rfloor$ indicates the largest integer less than $\bullet$

Update the fractional timing index

$$\mu(k+1) = \left(\mu(k) + \frac{T_i}{T_s} + n.\dot{\tau} + \tau \right)_{\text{mod 1}}$$

Compute the delta. If equals 0, do not shift any samples into the filter. Otherwise, shift Δm samples into the filter.

$$\Delta m = m(k+1) - m(k)$$

2 Table Driven Implementation

For real-time operation, it may be necessary to use look-up tables for the filter coefficients. The table size is determined by system considerations. For example, if an upper bound of 2.5% timing accuracy is desired and the system memory can handle the presence of a table of Lx20 where L is the number of taps, then interpolation of coefficients is not required. Otherwise, a first or second order polynomial is used to interpolate the filter coefficients. For coefficient interpolation, μ is quantized to the number of entries in the table. The residue parameterizes the interpolating polynomial.

$$\mu = \mu_q + \mu_r = Q[\mu, N] + \mu_r \qquad \textbf{Eq. 6}$$

Q[] is the quantization operator

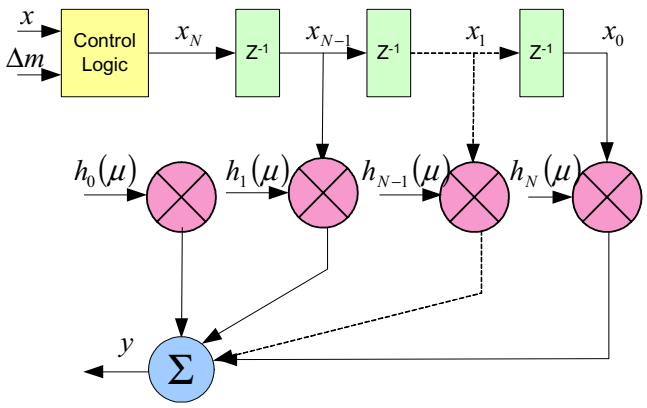
$$h(\mu) = \alpha_0 + \alpha_1.\mu_r + \alpha_2.\mu_r^2 \qquad \textbf{Eq. 7}$$

N is the number of delay steps in the table and α_j are polynomial coefficients computed using Lagrange's or divided difference methods. The data used in computing the coefficients are the table values that bound the value μ.

2.1 Interpolator

The interpolator of Figure 2 is a linear transversal filter structure with a mechanism that controls data transfer into the filter's memory. This is illustrated in Figure 4.

Figure 3: Interpolator Filter Structure



Where x represents the input samples and Δm represents the number of samples to be shifted into the filter.

2.2 Advantages

This design offers a significant reduction in the computational complexity with table lookup. The example presented in the introduction section which requires 1235 taps to achieve requires significantly more memory and more operations per output than a design with a table 11x20 in size which requires only eleven operations per output.

This design allows the incorporation of symbol timing correction which yields results highly accurate results at a significant less cost than the classical early/late gate designs. As discussed earlier, this design is not limited to an Early/Late gate but any timing error detector can be used. This design can correct both first order and second order timing errors.

This algorithm was successfully implemented on the Navy's software defined radio's Digital Modular Radio (DMR).

3 Simulations

Three examples are demonstrated: using the algorithm as a resampler only, resampler with first order timing correction and resampler with second order timing correction.

Case 1:
For this case, we illustrate the operation of the algorithm as an interpolator with a factor of 2. No timing correction.

Figure 4: Algorithm Output for an Interpolation Factor of 2 with no Timing Errors

m	μ	Δm	*Comment*
0	0	1	Δm = 1. Shift one sample. Interpolate with μ=0
0	.5	0	Δm = 0. For the same sample set, interpolate with μ=0.5
1	0	1	Δm = 1. Shift one sample. Interpolate with μ=0
1	.5	0	Δm = 0. For the same sample set, interpolate with μ=0.5
2	0	1	Δm = 1. Shift one sample. Interpolate with μ=0
2	.5	0	Δm = 0. For the same sample set, interpolate with μ=0.5

For each input sample, two samples are produced, one at a delay of zero and one at a delay of 0.5.

For a second example, we demonstrate decimation by a factor of 4.

Figure 5: Algorithm Output for a Decimation Factor of 4 with no Timing Errors

m	μ	Δm	*Comment*
0	0	1	Δm = 1. Shift one sample. Interpolate with μ=0
4	0	4	Δm = 4. Shift 4 samples. Interpolate with μ=0
8	0	4	Δm = 4. Shift 4 samples. Interpolate with μ=0
12	0	4	Δm = 4. Shift 4 samples. Interpolate with μ=0
16	0	4	Δm = 4. Shift 4 samples. Interpolate with μ=0
20	0	4	Δm = 4. Shift 4 samples. Interpolate with μ=0
24	0	4	Δm = 4. Shift 4 samples. Interpolate with μ=0

For every 4 input samples we compute one output sample thus achieving a 4 to 1 decimation factor.

Case 2:

For this example we demonstrate the operation of the algorithm when a fixed delay correction is required. We assume that a 20% symbol delay is correction is required. This would be the correction that is received by the algorithm from the timing error detector.

In this example, we have an interpolation factor of two. With no delay, we would expect an output similar to case 1 where the samples are being interpolated with a zero delay and 0.5 delay. However, since we have a fixed delay adjustment of 0.2 symbols, we expect the delta between the samples to remain at 0.5 and the delay to be constant at 0.2. As we can observe from the computations in Figure 7, the difference between the μ values is 0.5 indicating that the difference between the samples is fixed at 0.5 as expected. However, the location of the samples is now offset by 0.2 symbols because the first sample appears at 0.7 rather than 0.5. The next sample appears at 1.2 rather than 1.0 with the offset continuing.

Figure 6: Algorithm Output for an Interpolation Factor of 2 with 0.2 Symbol Delay Error

m	μ	Δm	*Comment*
0	0	0	Δm = 0. Shift 0 samples. Interpolate with μ=0
0	0.7	0	Δm = 0. Shift 0 samples. Interpolate with μ=0.7
1	0.2	1	Δm = 1. Shift 1 sample. Interpolate with μ=0.2
1	0.7	0	Δm = 0. Shift 3 samples. Interpolate with μ=0.7
2	0.2	1	Δm = 1. Shift 1 sample. Interpolate with μ=0.2
2	0.7	0	Δm = 0. Shift 0 samples. Interpolate with μ=0.7
3	0.2	1	Δm = 1. Shift 1 sample. Interpolate with μ=0.2
3	0.7	0	Δm = 0. Shift 0 samples. Interpolate with μ=0.7

4 References

[1] Gardner F., "Interpolation in Digital Modems – Part I: Fundamentals", IEEE Transactions on Communications, VOL. 41, No. 3, March 1993

[2] Erup L., Gardner F., Harris R. "Interpolation in Digital Modems – Part II: Implementation and Performance", IEEE Transactions on Communications, Vol. 41, No. 6, June 1992

[3] "Digital Communications Receivers", H. Meyer, M. Monocalaey and S.A. Fechtel, Wiley Books