Digital Receivers and Transmitters Using Polyphase Filter Banks for Wireless Communications

fred harris, Fellow, IEEE, Chris Dick, Member, IEEE, Michael Rice, Senior Member IEEE

Abstract: This paper provides a tutorial overview of multichannel wireless digital receivers and the relationships between channel bandwidth, channel separation, and channel sample rate. The overview makes liberal use of figures to support the underlying mathematics. A multichannel digital receiver simultaneously down-convert a set of frequency division multiplexed (FDM) channels residing in a single sampled data signal stream. In a similar way, a multichannel digital transmitter simultaneously up-converts a number of baseband signals to assemble a set of FDM channels in a single sampled data signal stream. The polyphase filter bank [1] has become the architecture of choice to efficiently accomplish these tasks. This architecture uses three [2, 3] interacting processes to assemble or to disassemble the channelized signal set. In a receiver these processes are an input commutator to effect spectral folding or aliasing due to a reduction in sample rate, a polyphase M-path filter to time align the partitioned and resampled time series in each path, and a discrete Fourier transform to phase align and separate the multiple base-band aliases. In a transmitter these same processes operate in a related manner to alias baseband signals to high order Nyquist zones while increasing the sample rate with the output commutator.

This paper presents a sequence of simple modifications to sampled data structures based on analog prototype systems to obtain the basic polyphase structure. We further discuss ways to incorporate small modifications in the operation of the polyphase system to accommodate secondary performance requirements. MATLAB simulations of a 10, a 40, and a 50 channel resampling receiver are included in the electronic version of this paper. An animated version of the 10-channel resampling receiver illustrates the time and frequency response of the filter bank when driven by a slowly varying linear FM sweep.

Manuscript received 2-May, 2002

fred harris is with the Department of Electrical Engineering, San Diego State University,
Chris Dick is with Xilinx, San Jose, CA.
Michael Rice is with the Department of Electrical Engineering, Brigham Young University

Motivation: Radio receivers and transmitters perform a sequence of invertible signal transformations in order to communicate through imperfect band limited channels. The transformations applied to waveforms are associated with disjoint frequency spans classically called baseband, intermediate frequency (IF), and radio frequency (RF). Early radios performed the desired transformations using appropriate linear and non-linear lumped and distributed analog circuit elements.

The confluence of three technology areas has had profound effect on the way we manipulate baseband and low IF signals. Two of these areas, enabled by the transistor and later by integrated circuits (ICs), are the analog-to-digital and digital-to-analog converter (ADC & DAC) and the programmable microprocessor. The third technology area is algorithm development by the digital signal processing (DSP) community. These technologies coupled with an educated and motivated work force led inexorably to insertion of DSP in the signal-processing path of radio receiver and transmitter systems.

Intel's former CEO Gordon Moore [4], observed that the cost of performing a specified processing task on an IC drops by a factor of two every 18-months or equivalently, the amount of processing that can be performed at a fixed cost doubles every 18-months. This relationship, known as Moore's law appears to be unique to the semiconductor industry. A similar cost-performance curve does not exist for general circuit components. A consequence of Moore's Law is the migration from designs that assemble and integrate sub-system to designs that are full systems on a Chip (SOC).

An important participant in the semiconductor arena is the Field Programmable Gate Array (FPGA) [5]. The FPGA consists of a vast array of configurable logic tiles, multipliers, and memory. This technology provides the signal-processing engineer with the ability to construct a custom data path that is tailored to the application at hand. FPGAs offer the flexibility of instruction set digital signal processors while providing the processing power and flexibility of an ASIC. The FPGA enables significant design cycle compression and time-to-market advantages, an important consideration in an economic climate with ever decreasing market windows and short product life cycles.

DSP based processing of baseband and low IF signals offer cost and performance advantages related to manufacturability, insensitivity to environment, ability to absorb design changes, and ease of feature insertion for product evolution and differentiability. The DSP segment of a radio enhances the radio while reducing its cost thus enabling larger market penetration as well as new market formation. DSP and RF and microwave communication systems are tightly coupled.

The authors have written this paper to help the RF and microwave engineer acquire an understanding of the key work performed by their DSP partners in pursuit of their common goal, the design and production of competitive, high quality, RF communication and RF monitoring systems. We start the paper with a review of a standard architecture for analog transmitters and receivers. Here the interface between continuous and sampled data is located at the end of the signal-processing path and operates at the highest signal to noise ratio with the lowest sample rate. We then present variants of the standard architectures in which the operating conditions change to process signals at higher dynamic range and at higher sample rates. A common variant of the high sample rate option is IF sampling.

High sample rate converters offer the option in a receiver to acquire large segments of input bandwidth and absorb much of the signal processing tasks and functions in DSP algorithms. The dual task of assembling large segments of bandwidth in a transmitter is implied but is not addressed here. The receiver processing includes, partitioning, filtering, translation, and demodulated. The remainder of the paper is restricted to description of various techniques to accomplish single or multiple channel extraction of signal bands from the bandwidth collected by high bandwidth converters.

Introduction: Base stations for cellular mobile communication systems [6] offer an example of a radio receiver that must down-convert and demodulate multiple simultaneous narrowband RF channels. The traditional architecture of a radio receiver that performs this task is shown in figure 1. This architecture contains N sets of dual-conversion sub-receivers. Each receiver amplifies and down-converts a selected radio-frequency (RF) channel to an intermediate frequency (IF) filter that performs initial bandwidth limiting.

The output of each IF filter is again down converted to baseband by matched quadrature mixers that are followed by matched base-band filters that perform final bandwidth control. Each quadrature down converted signal is then converted to their digital representation by a pair of matched analog-to-digital converters (ADC). The output of the ADCs is processed by digital signal processing (DSP) engines that perform the required synchronization, equalization, demodulation, detection, and channel decoding.

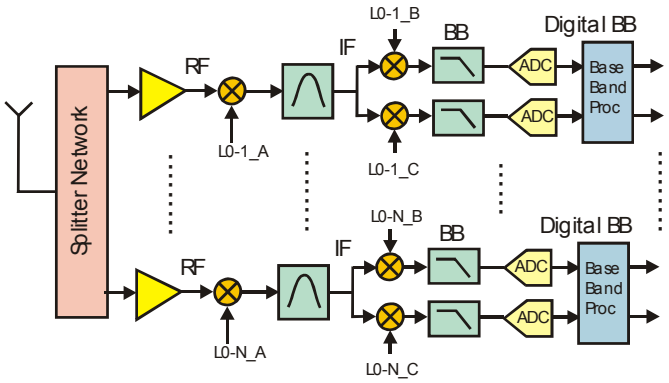

Figure 1. First Generation RF Architecture of N-Channel Receiver

Figure 2 shows a base station companion radio transmitter formed by N sets of dual conversion sub-transmitters that modulate and up-convert multiple simultaneous narrowband RF channels. Note that the signal flow for the transmitter chain is simply a reversal of the signal flow of the receiver chain.

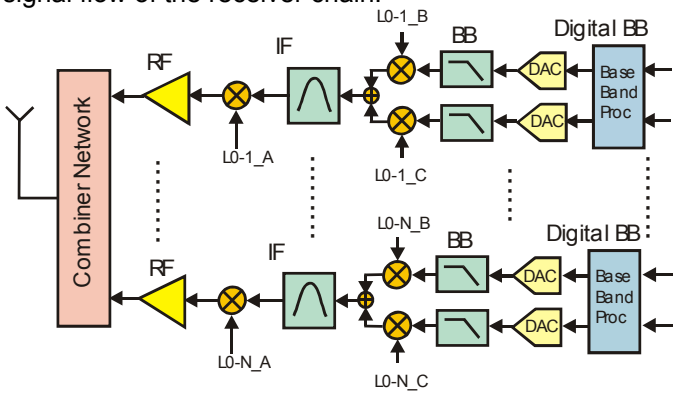

Figure 2. First Generation RF Architecture of N-Channel Transmitter

Gain and phase imbalance between the two paths containing the quadrature mixers, the analog baseband filters, and the ADC in an N-Channel receiver or N-channel transmitter is the cause of cross talk between the in-phase and quadrature (I/Q) components [7]. This in turn results in coupling between the many narrowband channels sometimes called ghosts or images. This spectral coupling can be described compactly by examining the model shown in figure 3. Here the composite I-Q gain and phase imbalances have been assigned to the quadrature term as the amplitude and phase shift of the sinusoid.

We can examine the unbalanced complex sinusoid presented to the mixer pair and compare its spectrum to that of the balanced spectrum. The complex sinusoid shown in eq-1 is expanded in eq-2 to explicitly show the positive and negative frequency components. Equation 3

uses the small signal approximation to obtain a simple estimate of the effects of gain and phase imbalance on the positive and negative frequency components of the quadrature mixer signal. Figure 4 presents a graphical visualization of these same spectral components.

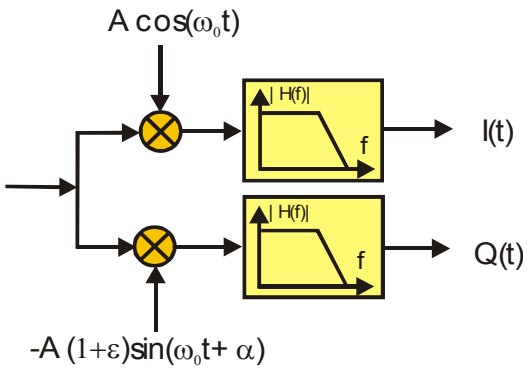

Figure 3. Quadrature Down Converter With Gain and Phase Imbalance

$$g(t) = A[\cos(\omega_0 t) - j\,(1+\varepsilon)\sin(\omega_0 t + \alpha)] \qquad (1)$$

$$\begin{aligned} g(t) = &\left\{\left[\frac{A}{2} - \frac{A}{2}(1+\varepsilon)\cos(\alpha)\right] - \right. \\ &\left. j\left[\frac{A}{2}(1+\varepsilon)\sin(\alpha)\right]\right\} e^{+j\omega_0 t} + \\ &\left\{\left[\frac{A}{2} + \frac{A}{2}(1+\varepsilon)\cos(\alpha)\right] - \right. \\ &\left. j\left[\frac{A}{2}(1+\varepsilon)\sin(\alpha)\right]\right\} e^{-j\omega_0 t} \end{aligned} \qquad (2)$$

$$\begin{aligned} g(t) \cong &A\left[\frac{\varepsilon}{2} - j\frac{\alpha}{2}\right] e^{+j\omega_0 t} + \\ &A\left[(1+\frac{\varepsilon}{2}) - j\frac{\alpha}{2}\right] e^{-j\omega_0 t} \end{aligned} \qquad (3)$$

Besides the obvious coupling between the quadrature components at the same frequency due to phase imbalance, we see a coupling between positive and negative frequencies due to both amplitude and phase imbalance. To achieve an imbalance related spectral image 40 dB below the desired spectral term, each imbalance term must be less than 1% of the desired term. It is difficult to sustain, over time and temperature, gain and phase balance of analog components to better than 1%. Third generation wireless systems impose severe requirements on level of I/Q balance. The need to achieve extreme levels of I/Q balance motivates us perform the complex conversion process in the DSP domain.

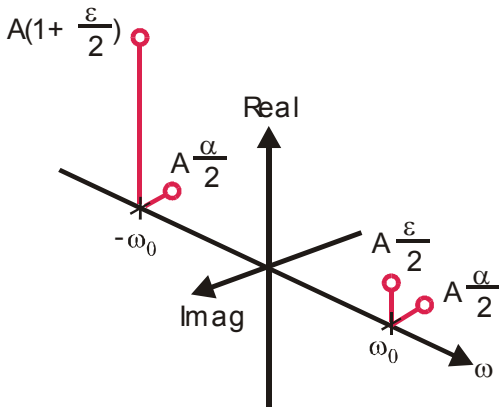

Figure 4. Spectral Components of Unbalanced Complex Sinusoid

Figures 5 and 6 present block diagrams of a second-generation multichannel receiver and transmitter in which the conversion from analog to digital (or digital to analog) occurs at IF rather than at baseband. Examining the receiver, we see that the down conversion of the separate channels is performed by a set of digital down converters and digital low pass filters. The digital process can realize arbitrarily small levels of imbalance by controlling the number of bits involved in the arithmetic operations. Precision of coefficients used in the filtering process sets an upper bound to spectral artifact levels at –5 dB/bit so that 12-bit arithmetic can achieve image levels below –60 dB. Thus the DSP based complex down conversion does not introduce significant imbalance related spectral terms. Similar comments apply to the DSP based up-conversions in the digital transmitter. The rule of thumb here is that the levels of spectral images are controlled to be below the quantizing noise floor of the ADC or DAC involved in the conversion process. A second advantage of digital translation process is that the digital filters following or preceding the mixers are designed to have linear phase characteristics, a characteristic trivially simple to realize in digital non-recursive filters [18].

The dynamic range and conversion speed of the ADC and the DAC becomes the limiting factor in the application of the architectures shown in figures 5 and 6. The dynamic range of the converter is determined to first order, by the number of bits in the converter with each bit contributing 6-dB [8]. The Nyquist criterion [9] establishes the minimum sample rate to obtain an alias free representation of the sampled signal. The Nyquist crite-

rion directs us to select the sample rate to exceed the two-sided bandwidth of the signal. Converters have the property that the product of sample rate and number of conversion levels is a constant [10]. This relationship is shown in eq-4 where b is the number of bits in the converter. Equation 5, a rearrangement of eq-4, shows how the number of bits varies inversely with the sample rate.

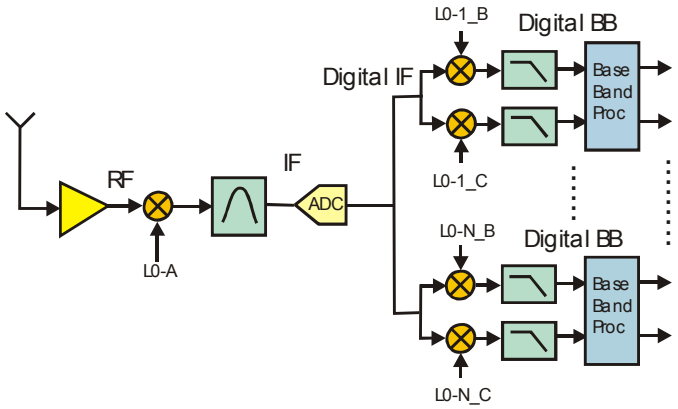

Figure 5. Second Generation RF Architecture of N-Channel Receiver

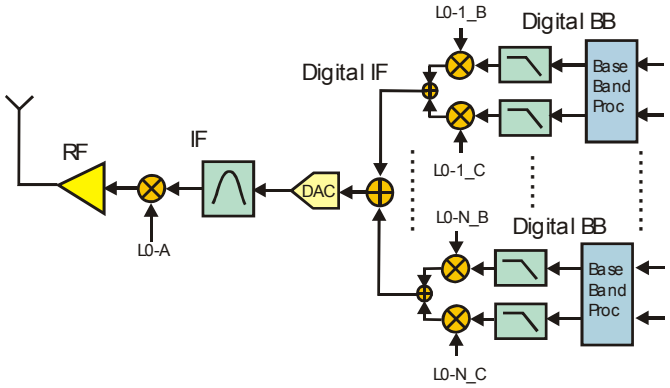

Figure 6. Second Generation RF Architecture of N-Channel Transmitter

Figure 7 is a graphical presentation of this relationship along with a scattering of data points showing the conversion speed versus precision performance exhibited by a number of current (mid-year 2002) ADCs. A useful rule of thumb is that a converter operating at 10-MHz sample rate can deliver 16-bit performance and that for every doubling of the sample rate results in a 1-bit (or 6-dB) reduction in conversion precision. The sloped line in figure 7 matches this rule. The intercept of this performance line is related to the aperture uncertainty of the conversion process, a parameter that improves slowly in response to advances in semiconductor technology.

$$\log_2(2^b \ f_{SAMPLE}) = k \qquad (4)$$

$$b = \log_2\{k\} - \log_2(f_{SAMPLE}) \qquad (5)$$

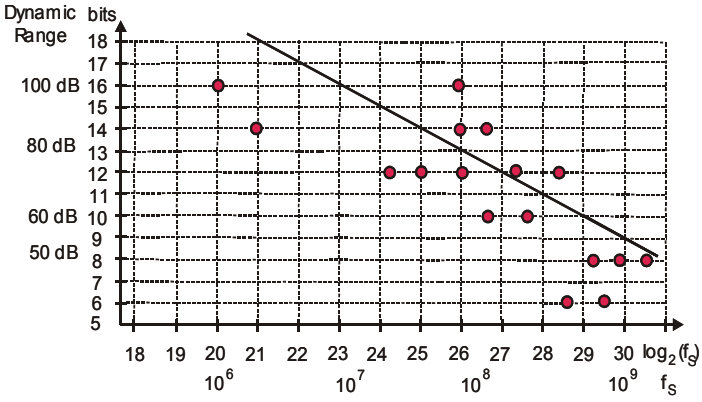

Figure 7. Scatter Diagram Showing Speed-Precision Performance of ADCs

A final comment on ADCs is that the spurious terms generated by converter non-linearities often exceed the quantizing noise levels described by the –6dB per bit rule. The true performance measure of the ADC is the full bandwidth, full-scale spurious free dynamic range (SPDR) [11].

The limited dynamic range available from high speed ADCs restricts the range of applications for the architectures presented in figures 5 and 6 to IF center frequencies to the low to mid 100's of MHz. To extend the application range of digital N-channel receivers and digital N-channels transmitters we often use a hybrid scheme in which the initial complex down conversion is performed with analog I-Q mixers and the channelization is performed digitally after the ADC. The first conversion can be considered a block conversion to baseband that delivers the frequency band of interest to the DSP arena for subsequent channelization. The hybrid forms of the digital N-channel receiver and the digital N-channel transmitter are shown in figures 8 and 9 respectively. DSP techniques are applied to the digitized I-Q data to balance the gain and phase offsets in the analog ADC and DAC. DSP based I-Q balance correction is a standard signal conditioning task in high-end as well as consumer based receivers and transmitters.

Digital Down Conversion: In the previous section we described the process of sampling an analog IF signal or complex analog baseband signal containing the set of N-frequency division multiplexed channels to be further processed or channelized by DSP techniques. We consider the input signal to be composed of many equal-bandwidth, equally spaced, frequency division multi-

plexed (FDM) channels as shown in figure 10. These many channels are digitally down-converted to base-band, bandwidth constrained by digital filters, and subjected to a sample rate reduction commensurate with the bandwidth reduction.

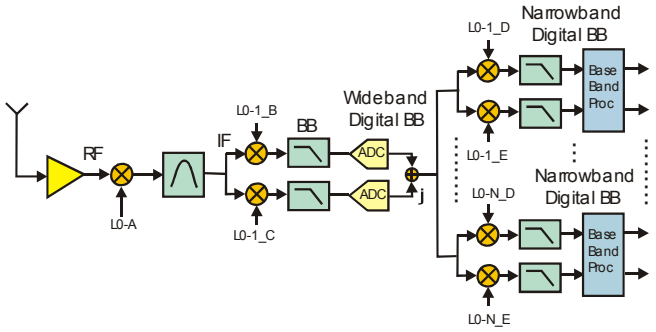

Figure 8. Second Generation Hybrid RF Digital N-Channel Receiver

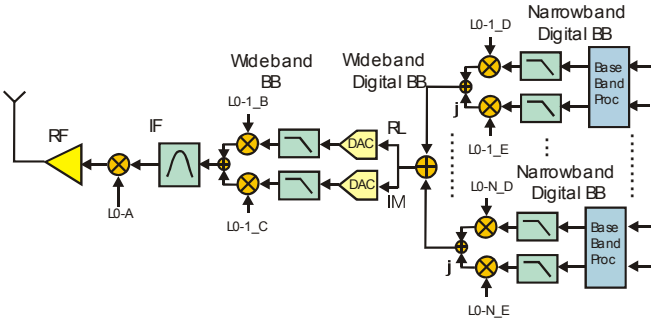

Figure 9. Second Generation Hybrid RF Digital N-Channel Transmitter

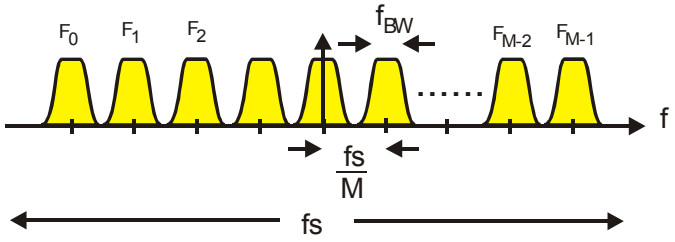

Figure 10. Input Spectrum of Frequency Division Multiplexed Signal to be Channelized

The signal processing task can be performed as a replica of the analog prototype solution by a DSP based set of independent down-conversion processes as indicated in figure 11. For clarity of presentation, we describe how digital frequency denoted by the angle θ_k is derived from analog frequency f_k. This change of variables is shown on equations 6 through 8. Equation 6 presents a complex sinusoid of frequency $2\pi f_k$. We note that frequency is the time derivative of the time evolving phase angle $\theta(t)$ and has units of radians/second. The sampled data sinusoid is obtained by replacing the time variable "t" with the sampled time variable "nT" as shown in eq-7. Note that the units of the sample time variable are samples and seconds/sample respectively. The angle formed by the product $2\pi f_k$ and T or by the equivalent term $2\pi f_k/f_S$, where $f_S=1/T$, is shown in eq-8. Here the product term $2\pi f_k$, denoted by θ_k, has units of radians/second by seconds/sample or radians/sample.

$$g(t) = \exp(j\,2\pi\,f_k\,t) \tag{6}$$

$$g(n) = g(t)\big|_{t=nT} = \exp(j\,2\pi\,f_k\,nT) \tag{7}$$

$$g(n) = \exp(j\,2\pi\,\frac{f_k}{f_S}n) = \exp(j\,\theta_k n) \tag{8}$$

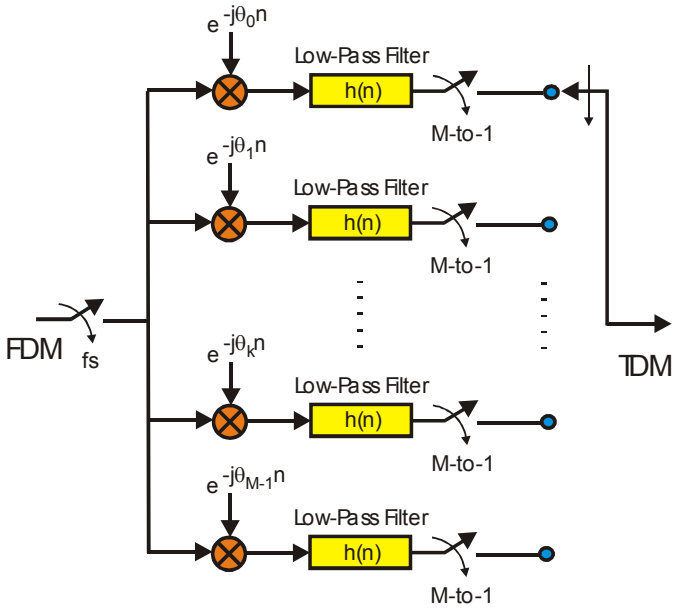

Figure 11. Conventional Channelizer as a Replica of Analog Prototype: Down-Converters, Base-Band Filters, and Resamplers

An alternate implementation performs the channelization as a single merged process called a polyphase N-Path filter bank [12] as shown in figure 12. The polyphase filter bank partition offers a number of significant advantages relative to the set of individual down conversion receivers. The primary advantage is reduced cost due to major reduction in system resources required to perform the multichannel processing.

The first sector in the communications community to make wide use of this form of the transmultiplxer was the Bell System network that used this structure in the early

1980's to modulate and demodulate analog single side band (SSB) FDM supergroup containing 60 4-kHz channels [13]. We now present a tutorial review to describe how the conventional channelizer is converted to the standard polyphase channelizers [14, 15]. This review contains simple equations and informative block diagrams representing the sequence of modifications that affect the transformation. We then extend the tutorial to incorporate a number of variations to perform secondary processing tasks along with the basic channelization task.

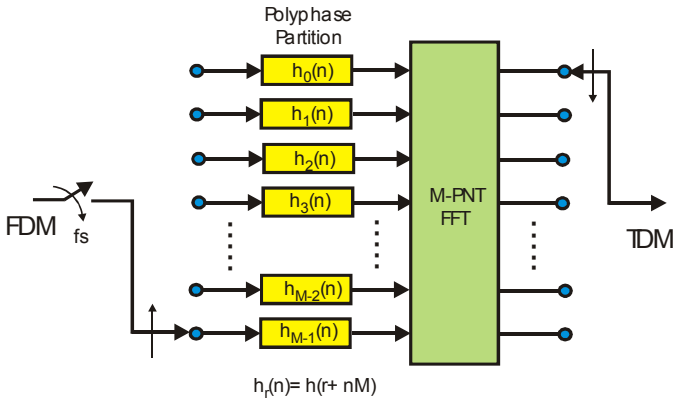

Figure 12. Polyphase Channelizer: Resampler, All-Pass Partition, and FFT Phase Shifters

Transforming the Channelizer, First Step: The block diagram of a single channel of a conventional channelizer is shown in figure 13. This structure performs the standard operations of down conversion of the selected channel with a complex heterodyne, low-pass filtering to reduce bandwidth to the channel bandwidth, and down sampling to a reduced rate commensurate with the reduced bandwidth. We mention that the down sampler is commonly referred to as a decimator, a term which means to destroy every tenth one. Since nothing is destroyed, and nothing happens in tenths, we prefer, and will continue to use the more descriptive name, down sampler.

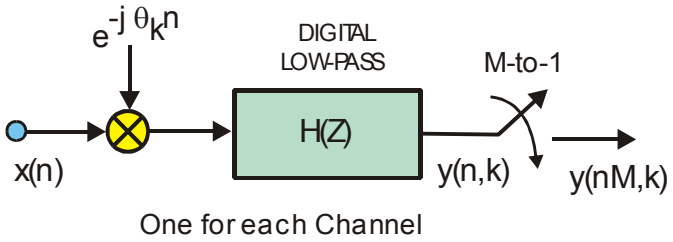

Figure 13. k-th Channel of Conventional Channelizer

The expression for y(n,k), the time series output from the k-th channel, prior to resampling, is a simple convolution as shown in eq-9.

$$y(n,k) = [x(n)e^{-j\theta_k n}] * h(n)$$
$$= \sum_{r=0}^{N-1} x(n-r)\, e^{-j\theta_k (n-r)}\, h(r) \qquad (9)$$

The output data from the complex mixer is complex hence is represented by two time series, I(n) and Q(n). The filter with real impulse response h(n) is implemented as two identical filters, each processing one of the quadrature time series. The convolution process is performed by a simple digital filter that performs the multiply and add operations between data samples and filter coefficients extracted from two sets of addressed memory registers. One register set contains the data samples while the other contains the coefficients that define the filter impulse response. This structure is shown in figure 14.

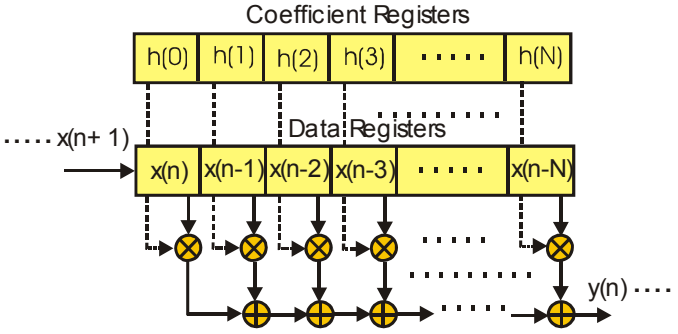

Figure 14. Conceptual Digital Filter: Coefficients and Data Registers, Multipliers, and Adders

We can rearrange the summation of eq-9 to obtain a related summation reflecting the *equivalency theorem* [16]. The equivalency theorem states that the operations of down conversion followed by a low-pass filter are totally equivalent to the operations of a band-pass filter followed by a down conversion. The block diagram demonstrating this relationship is shown in figure 15, while the rearranged version of eq-9 is shown in eq-10. Note here, that the up-converted filter, h(n) exp(jθ_kn), is complex and as such its spectrum resides only on the positive frequency axis without a negative frequency image. This is not a common structure for an analog prototype because of the difficulty of forming a pair of analog quadrature filters exhibiting a 90-degree phase difference across the filter bandwidth. The closest equivalent structure in the analog world is the filter pair used in image-reject mixers.

Applying the transformation suggested by the equivalency theorem to an analog prototype system does not make sense since it doubles the required

hardware. We would have to replace a complex scalar heterodyne (two mixers) and a pair of low-pass filters with a pair of band-pass filters, containing twice the number of reactive components, and a full complex heterodyne (four mixers). If it makes no sense to use this relationship in the analog domain, why does it make sense in the digital world? The answer is found in the fact that we define a digital filter as a set of weights stored in coefficient memory. Thus, in the digital world,

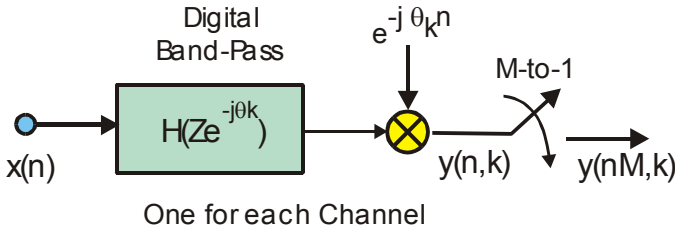

Figure 15. Band-Pass Filter, k-th Channel of Channelizer

$$y(n,k) = \sum_{r=0}^{N-1} x(n-r)\, e^{-j(n-r)\theta_k} h(r)$$
$$= \sum_{r=0}^{N-1} x(n-r)\, e^{-jn\theta_k} h(r) e^{jr\theta_k} \qquad (10)$$
$$= e^{-jn\theta_k} \sum_{r=0}^{N-1} x(n-r)\, h(r) e^{jr\theta_k}$$

we incur no cost in replacing the pair of low pass filters h(n) required in the first option with the pair of band pass filters h(n) $\cos(n\theta_k)$ and h(n) $\sin(n\theta_k)$ required for the second option. We accomplish this task by a simple download to the coefficient memory. The filter structures corresponding to the two sides of the equivalency theorem are shown in figure 16. Note the input signal interacts with the complex sinusoid as a product at the filter input or as a convolution in the filter weights.

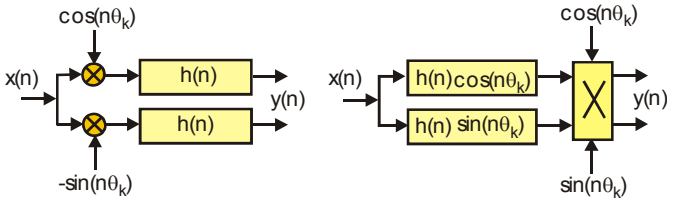

Figure 16. Block Diagrams Illustrating Equivalency Between Operations of Heterodyne and Baseband Filter With Band-Pass Filter and Heterodyne

We still have to address the matter of the full complex heterodyne required for the down conversion at the filter output rather than at the filter input. Examining figure 16, we note that following the output down conversion, we perform a sample rate reduction by retaining only one sample in every M-samples. Recognizing that there is no need to down convert the samples we discard in the down sample operation, we choose to down sample only the retained samples. This is shown in figure 17.

We note in figure 17, that when we bring the down converter to the low data rate side of the resampler, we are in fact also down sampling the time series of the complex sinusoid. The rotation rate of the sampled complex sinusoid is θ_k and $M\theta_k$ radians per sample at the input and output respectively of the M-to-1 resampler.

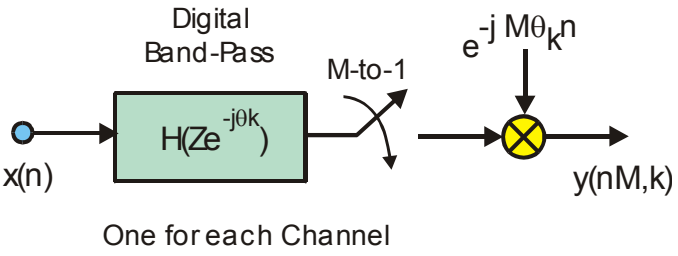

Figure 17. Down Sampled Down Converter, Band-Pass k-th Channel

This change in rotation rate is an aliasing affect, a sinusoid at one frequency or phase slope, appears at another phase slope when resampled. We now invoke a constraint on the sampled data center frequency of the down converted channel. We choose center frequencies θ_k which will alias to DC (zero frequency) as a result of the down sampling to $M\theta_k$. This condition is assured if $M\theta_k$ is congruent to 2π, which occurs when $M\theta_k = k\,2\pi$, or more specifically, when $\theta_k = k\,2\pi/M$. The modification to figure 17 to reflect this provision is seen in figure 18. The constraint, that the center frequencies be integer multiples of the output sample rate assures aliasing to base band by the sample rate change. When a channel aliases to base band by the resampling operation the resampled related heterodyne defaults to a unity-valued scalar, which consequently is removed from the signal-processing path. Frequency offsets of the channel center frequencies, due to oscillator drift or Doppler effects, are removed after the down conversion by a baseband phase locked loop (PLL) controlled mixer. This baseband mixer operates at the output sample rate rather than at the input sample rate for a conventional down converter. We consider this required final mixing operation a post conversion task and allocate it to the next processing block.

The operations invoked by applying the equivalency theorem to the down conversion process guided us to the following sequence of maneuvers: i) slide the input heterodyne through the low pass filters to their outputs,

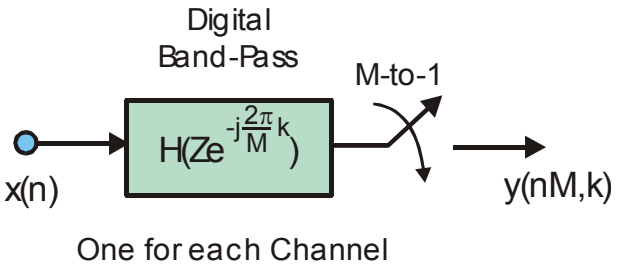

Figure 18. Alias to Base Band, Down Sampled Down Converter, Band-Pass k-th Channel

ii) doing so converts the low pass filters to a complex band pass filter, iii) slide the output heterodyne to the downside of the down sampler, iv) doing so aliases the center frequency of the oscillator, v) restrict the center frequency of the band pass to be a multiple of the output sample rate, vi) doing so assures alias of the selected pass band to base band by the resampling operation, and finally, vii) discard the now unnecessary heterodyne. The spectral effect of these operations is shown in figure 19. The savings realized by this form of the down conversion is due to the fact we no longer require a quadrature oscillator nor the pair of input mixers to effect the frequency translation.

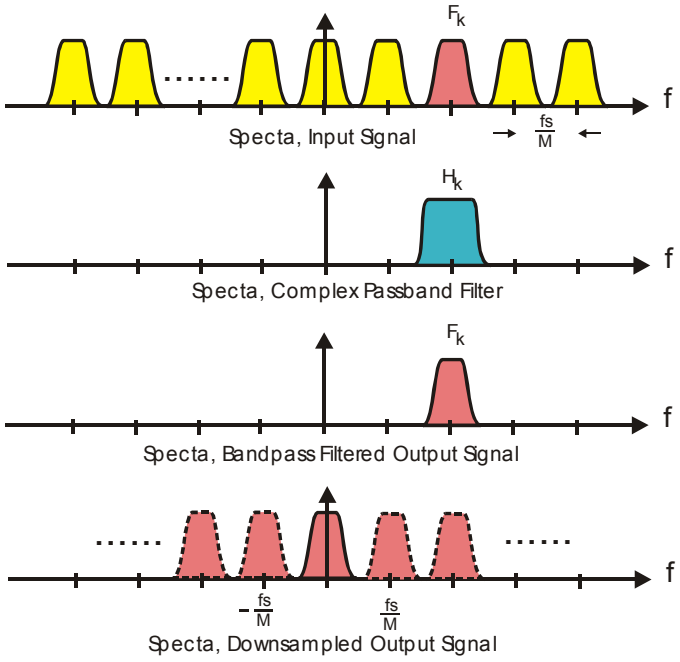

Figure 19. Spectral Description of Down Conversion Realized by a Complex Band Pass Filter at a Multiple of Output Sample Rate, Aliased to Baseband by Output Resampling

Transforming the Channelizer, Second Step: Examining figure 18, we note that the current configuration of the single channel down converter involves a band pass filtering operation followed by a down sampling of the filtered data to alias the output spectrum to baseband. Following the idea developed in the previous section that led us to down-convert only those samples retained by the down sampler, we similarly conclude that there is no need to compute the output samples from the pass band filter that will be discarded by the down sampler. We now interchange the operations of filter and down sample with the operations of down sample and filter. The process that accomplishes this interchange is known as the *Noble Identity* [17], which we now review.

The noble identity is compactly presented in figure 20 which we describe with similar conciseness by "The output from a filter $H(Z^M)$ followed by an M-to-1 down sampler is identical to an M-to-1 down sampler followed by the filter H(Z)". The Z^M in the filter impulse response tell us that the coefficients in the filter are separated M-samples rather than the more conventional one sample delay between coefficients in the filter H(Z). We must take care to properly interpret the operation of the M-to-1 down sampler. The interpretation is that the M-to1 down sampled time series from a filter processing every M-th input sample presents the same output by first down sampling the input by M-to-1 to discard the samples not used by the filter to compute the retained output samples and then operating the filter on the retained input samples. The noble identity works because M-samples of delay at the input clock rate is the same interval as one-sample delay at the output clock rate.

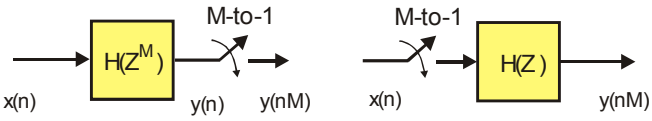

Figure 20. *Noble Identity*: A Filter Processing Every M-th Input Sample Followed by an Output M-to-1 Down Sampler is the same as an Input M-to-1 Down Sampler Followed by a Filter Processing Every M-th Input Sample.

We might ask, "Under what condition does a filter manage to operate on every M-th input sample?" We answer this query by rearranging the description of the filter to establish this condition so that we can invoke the noble identity. This rearrangement starts with an initial partition of the filter into M-parallel filter paths. The Z-transform description of this partition is presented in equations 11 through 14, which we interpret in figures 21 through 23. For ease of notation, we first examine the base-band version of the noble identity and then trivially extend it to the pass band version.

$$H(Z) = \sum_{n=0}^{N-1} h(n)\, Z^{-n}$$
$$= h(0) + h(1)\, Z^{-1} + h(2)\, Z^{-2} + \quad (11)$$
$$h(3)\, Z^{-3} + \cdots + h(N-1)\, Z^{-(N-1)}$$

Anticipating the M-to-1 resampling, we partition the sum shown in eq-11 to a sum of sums as shown in eq-12. This partition maps a one-dimensional array of weights (and index markers Z^{-n}) to a two dimensional array. This mapping is sometimes called lexicographic, for natural order, a mapping that occurs in the Cooley-Tukey fast Fourier transform. In this mapping we load an array by columns but process the array by rows. In our example, the partition forms columns of length M containing M successive terms in the original sum, and continues to form adjacent M-length columns till we account for all the elements of the original one-dimensional array.

$$\begin{aligned}
H(Z) = \; & h(0) + h(M+0)\,Z^{-M} + h(2M+0)Z^{-(2M+0)} + \cdots \\
& h(1)Z^{-1} + h(M+1)Z^{-(M+1)} + h(2M+1)Z^{-(2M+1)} + \cdots \\
& h(2)Z^{-2} + h(M+2)Z^{-(M+2)} + h(2M+2)Z^{-(2M+2)} + \cdots \\
& h(3)Z^{-3} + h(M+3)Z^{-(M+3)} + h(2M+3)Z^{-(2M+3)} + \cdots \\
& \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \\
& h(M-1)Z^{-(M-1)} + h(2M-1)Z^{-(2M-1)} + h(3M-1)Z^{-(3M-1)} + \cdots
\end{aligned} \quad (12)$$

We note that the first row of the two dimensional array is a polynomial in Z^M, which we will denote $H_0(Z^M)$ a notation to be interpreted as an addressing scheme to start at index 0 and increment in stride of length M. The second row of the same array, while not a polynomial in Z^M, is made into one by factoring the common Z^{-1} term and then identifying this row as $Z^{-1}\, H_1(Z^M)$. It is easy to see that each row of eq-12 can be described as $Z^{-r}\, H_r(Z^M)$ so that eq-12 can be re-written in a compact form as shown in eq-13.

$$H(Z) = H_0(Z^M) + Z^{-1} H_1(Z^M) +$$
$$Z^{-2} H_2(Z^M) + \cdots\cdots + \quad (13)$$
$$Z^{-(M-1)} H_{(M-1)}(Z^M)$$

We rewrite eq-13 in the traditional summation form as shown in eq-14, which describes the original polynomial as a sum of delayed polynomials in Z^M.

The block diagram reflecting this M-path partition of a resampled digital filter is shown in figure 21. The output of the filter is the resampled sum of the output of the M separate filter stages along the M-paths. We pull the resampler through the output summation element and

$$H(Z) = \sum_{r=0}^{M-1} Z^{-r} H_r(Z^M)$$
$$= \sum_{r=0}^{M-1} Z^{-r} \sum_{n=0}^{(N/M)-1} h(r+nM) Z^{-M\,n} \quad (14)$$

down sample the separate outputs, only performing the output sum for the retained output sample points. With the resamplers at the output of each filter, which operates on every M-th input sample, we are prepared to invoke the noble identity and pull the resampler to the input side of each filter stage. This is shown in figure 22. The input resamplers operate synchronously, all closing at the same clock cycle. When the switches close, the signal delivered to the filter on the top path is the current input sample. The signal delivered to the filter one path down is the content of the one stage delay line, which of course is the previous input sample. Similarly, as we traverse the successive paths of the M-path partition, we find upon switch closure, that the k-th path receives a data sample delivered k-samples ago. We conclude that the interaction of the delay lines in each path with the set of synchronous switches can be likened to an input commutator that delivers successive samples to successive legs of the M-path filter. This interpretation is shown in figure 23.

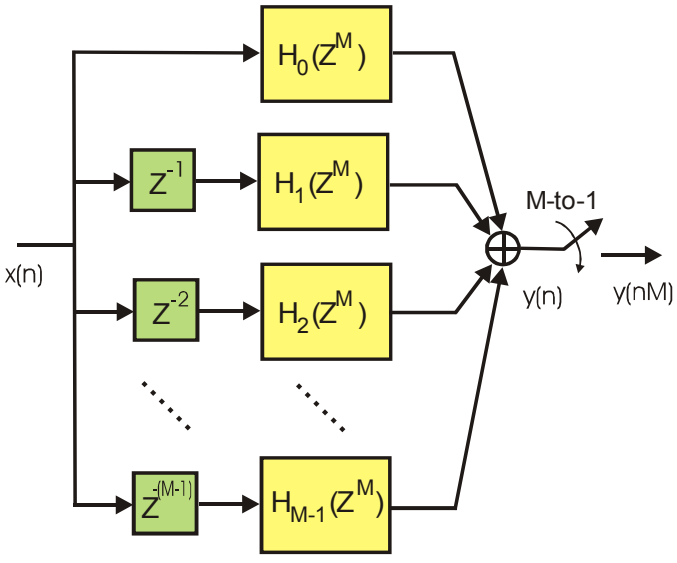

Figure 21. M-Path Partition of Prototype Low-Pass Filter with Output Resampler

We now complete the final steps of the transform that changes a standard mixer down converter to a resampling M-Path down converter. We note and apply the

frequency translation property of the Z-Transform [18]. This property is illustrated and stated in eq-15. Interpreting the relationship presented in eq-15, we note that if h(n), the impulse response of a base band filter, has a Z-transform H(Z), then the sequence $h(n)e^{+j\theta n}$, the impulse response of a pass band filter, has a Z-transform $H(Z\ e^{-j\theta n})$. Simply stated, we can convert a low pass filter to a band pass filter by associating the complex heterodyne terms of the modulation process either with the filter weights or with the delay elements storing the filter weights.

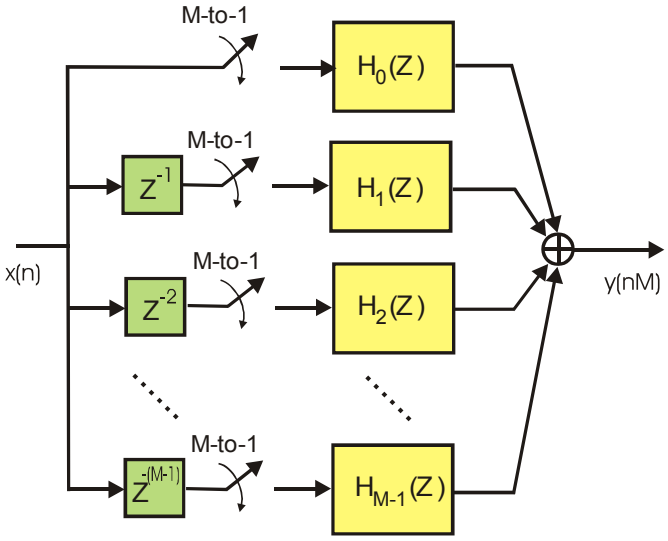

Figure 22. M-Path Partition of Prototype Low-Pass Filter with Input Resamplers

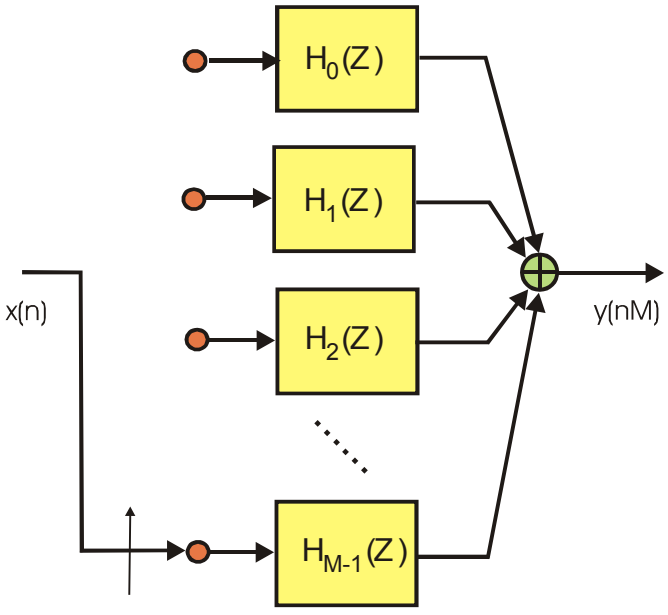

Figure 23. M-Path Partition of Prototype Low-Pass Filter with Input Path Delays and M-to-1 Resamplers Replaced by Input Commutator

$$\text{if}\quad H(Z) = h(0) + h(1)Z^{-1} + h(2)Z^{-2} + \cdots + n(N-1)Z^{-(N-1)} = \sum_{n=0}^{N-1} h(n)Z^{-n}$$

and

$$\begin{aligned} G(Z) &= h(0) + h(1)e^{j\theta}Z^{-1} + h(2)e^{j2\theta}Z^{-2} + \cdots + h(N-1)e^{j(N-1)\theta}Z^{-(N-1)} \\ &= h(0) + h(1)[e^{-j\theta}Z]^{-1} + h(2)[e^{-j\theta}Z]^{-2} + \cdots + h(N-1)[e^{-j\theta}Z]^{-(N-1)} \\ &= \sum_{n=0}^{N-1} h(n)[e^{-j\theta}Z]^{-n} \end{aligned} \qquad (15)$$

then

$$G(Z) = H(Z)\Big|_{Z=e^{-j\theta}Z} = H(e^{-j\theta}Z)$$

We now apply this relationship to eq-10, or equivalently to figure 23 by replacing each Z with $Z\ e^{-j\theta}$, or perhaps more clearly, replacing each Z^{-1} with $Z^{-1}\ e^{j\theta}$, with the phase term θ satisfying the congruency constraint of the previous section, that $\theta = k(2\pi/M)$. Thus Z^{-1} is replaced with $Z^{-1}\ e^{jk(2\pi/M)}$, and Z^{-M} is replaced with $Z^{-M}\ e^{jkM(2\pi/M)}$. By design, the kM-th multiple of $2\pi/M$ is a multiple of 2π for which the complex phase rotator term defaults to unity, or in our interpretation, aliases to base band (DC). The default to unity of the complex phase rotator occurs in each path of the M-path filter shown in figure 24. The non-default complex phase angles are attached to the delay elements on each of the M paths. For these delays, the terms Z^{-r} are replaced by the terms $Z^{-r}\ e^{jkr(2\pi/M)}$. The complex scalar $e^{jkr(2\pi/M)}$ attached to each path of the M-path filter can be placed anywhere along the path, and in anticipation of the next step, we choose to place the complex scalar after the down sampled path filter segments $H_r(Z)$. This is shown in figure 24.

The modification to the original partitioned Z-Transform of eq-14 to reflect the added phase rotators of figure 24 is shown in eq-16.

$$H(Ze^{-j\frac{2\pi}{M}k}) = \sum_{r=0}^{M-1} Z^{-r} e^{j\frac{2\pi}{M}rk} H_r(Z) \qquad (16)$$

The computation of the time series obtained from the output summation in figure 24 is shown in eq-17. Here the argument nM reflects the down sampling operation which increments through the time index in stride of length M, delivering every M-th sample of the original output series. The variable $y_r(nM)$ is the nM-th sample from the filter segment in the r-th path, and y(nM,k) is the nM-th time sample of the time series from the k-th center frequency. Remember that the down converted center frequencies located at integer multiples of the output

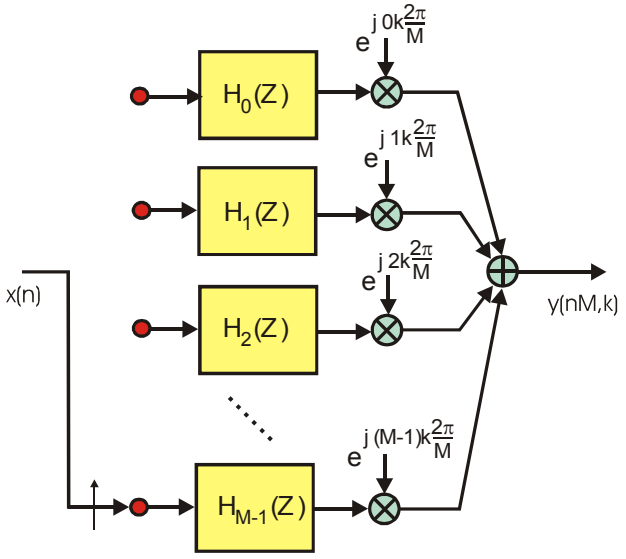

Figure 24. Resampling M-Path Down Converter

sample frequency are the frequencies that alias to zero frequency under the resampling operation. Note the output y(nM,k) is computed as a phase coherent summation of the M output series $y_r(nM)$. This phase coherent sum is in fact, a DFT of the M-path outputs, which can be likened to beam-forming the output of the path filters.

$$y(nM,k) = \sum_{r=0}^{M-1} y_r(nM) e^{j\frac{2\pi}{M}rk} \qquad (17)$$

The beam-forming perspective offers interesting insight to the operation of the resampled down-converter system we have just examined. The reasoning proceeds as follows: the commutator delivering consecutive samples to the M input ports of the M-path filter performs a down sampling operation. Each port of the M-path filter receives data at one-Mth of the input rate. The down sampling causes the M-to-1 spectral folding, effectively translating the M-multiples of the output sample rate to base band. The alias terms in each path of the M-path filter exhibit unique phase profiles due to their distinct center frequencies and the time offsets of the different down sampled time series delivered to each port. These time offset are in fact the input delays shown if figure 22 and in eq-18. Each of the aliased center frequency experiences a phase shift shown in eq-18, equal to the product of its center frequency and the path time delay.

$$\begin{aligned} \phi(r,k) &= \omega_k \, \Delta T_r \\ &= 2\pi \frac{f_s}{M} k \; rT_s \\ &= 2\pi \frac{f_s}{M} k \; r \frac{1}{f_s} = \frac{2\pi}{M} kr \end{aligned} \qquad (18)$$

The phase shifters of the DFT perform phase coherent summation, very much like that performed in narrow band beam forming, extracting from the myriad of aliased time series, the alias with the particular matching phase profile. This phase sensitive summation aligns contributions from the desired alias to realize the processing gain of the coherent sum while the remaining alias terms, which exhibit rotation rates corresponding to the M roots of unity, are destructively canceled in the summation.

The inputs to the M-path filter are not narrow band, and phase shift alone is insufficient to effect the destructive cancellation over the full bandwidth of the undesired spectral contributions. Continuing with our beam-forming perspective, to successfully separate wideband signals with unique phase profiles due to the input commutator delays, we must perform the equivalent of time-delay beam forming. The M-path filters, obtained by M-to-1 down sampling of the prototype low-pass filter supply the required time delays. The M-path filters are approximations to all-pass filters, exhibiting, over the channel bandwidth, equal ripple approximation to unity gain and the set of linear phase shifts that provide the time delays required for the time delay beam forming task.

The filter achieves this property by virtue of the way we partitioned the low-pass prototype. Each of the M-path filters, filter $h_r(n)$ for instance, with weights h(r+nM) is formed by starting with an initial offset of "r" samples and then incrementing by stride of M samples. The initial offsets, unique to each path, are the source of the different linear phase shift profiles. It is for this reason, the different linear phase profiles, that the filter partition is known as *a polyphase* filter. The phase shift and group delay profiles for a 10-path filter are shown in figures 25 and 26. These figures are part of the output suite of figures formed by the MATLAB m-file ***filter_ten*** contained in appendix-1 of the electronic version of this paper on the CDROM accompanying this issue. This file synthesizes a 10-stage polyphase channelizer and presents input and output time series and spectra.

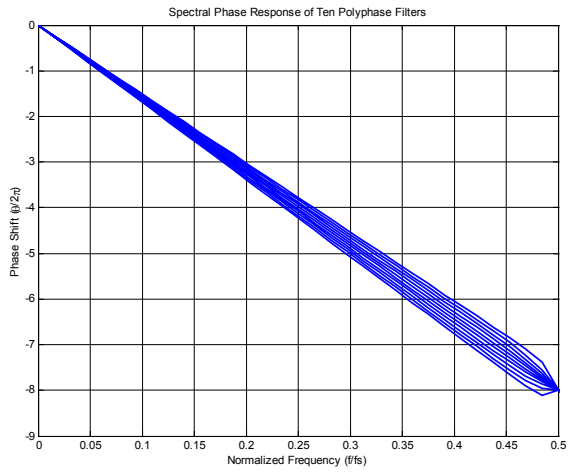

Figure 25. Phase Profiles for Ten-Stages of Ten Path Polyphase Partition

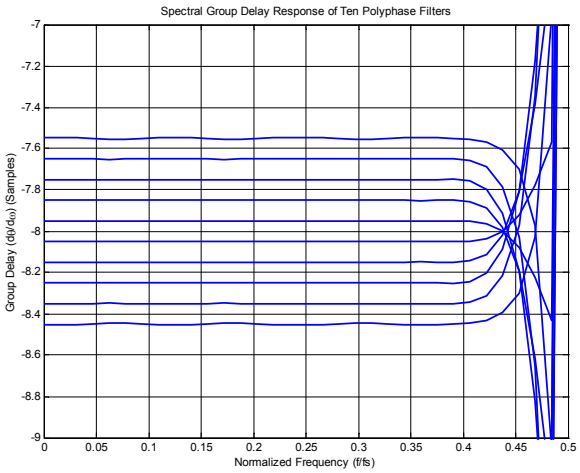

Figure 26. Group Delay Profiles for Ten-Stages of Ten Path Polyphase Partition

A useful perspective is that the phase rotators following the filters perform phase alignment of the band center for each aliased spectral band while the polyphase filters perform the required differential phase shift across these same channel bandwidths. When the polyphase filter is used to down convert and down sample a single channel the phase rotators are implemented as external complex products following each path filter. When a small number of channels are being down converted and down sampled, appropriate sets of phase rotators can be applied to the filter stage outputs and summed to form each channel output. We take a different approach when the number of channels becomes sufficiently large. Here sufficiently large means on the order of $\log_2(N)$. Since the phase rotators following the polyphase filter stages are the same as the phase rotators of a DFT, we can use the DFT to simultaneously apply the phase shifters for all of the channels we wish to extract from the aliased signal set. This is reminiscent of phased array beam forming. For computational efficiency, the FFT algorithm implements the DFT.

It is useful to once again examine figure 12 in which the polyphase filter and FFT was first presented as a channelized receiver. Think of the many input arms of the FFT as being coupled through a set of distributed phase rotators to each output port of the FFT with each output port accessed through a different vector of phase slopes. Readers may recognize that the phase shifts between input and output ports of the FFT are the same as those forming the Butler Matrix used in phased array beam forming [19, 20].

At this point it is instructive to make a comparison of the conventional mixer down converter and the resampled polyphase down converter. The input to either process can be real or complex. In the mixer down converter model, a separate mixer pair and filter pair must be assigned to each channel of the channelizer and that these mixers all operate at the high input data rate, prior to down sampling. By way of contrast, in the resampled polyphase there is only one low pass filter required to service all the channels of the channelizer, and this single filter accommodates all the channels as co-occupying alias contributors of the base band bandwidth. This means that all the processing performed in the resampled polyphase channelizer occurs at the low output sample rate. When the input signal is real, there is another significant difference between the two processes. In the mixer down converter model the signal is made complex by the input mixers as we enter the process, which means that the low pass-filtering task requires two filters, one for each of the quadrature components, while in the resampling channelizer the signal is made complex by the phase rotators as we leave the process, consequently we require only one partitioned low pass filter to process the real input signal.

Before moving on to the next topic, let us summarize what we have accomplished to this point. The commutator performs an input sample rate reduction by commutating successive input samples to selected paths of the M-path filter. Sample rate reduction occurring prior to any signal processing causes spectral regions residing at multiples of the output sample rate to alias to baseband. This desired result allows us to replace the many down-converters of a standard channelizer, implemented with dual mixers, quadrature oscillators, and bandwidth reducing filters, with a collection of trivial aliasing operations performed in a single portioned and resampled filter.

The partitioned M-path filter performs the task of aligning the time origins of the offset sampled data sequences delivered by the input commutator to a single common output time origin. This is accomplished by the

all-pass characteristics of the M-path filter sections that apply the required differential time delay to the individual input time series. The DFT performs the equivalent of a beam forming operation; the coherent summation of the time aligned signals at each output port with selected phase profiles. The phase coherent summation of the outputs of the M-path filters separate the various aliases residing in each path by constructively summing the selected aliased frequency components located in each path, while simultaneously destructively canceling the remaining aliased spectral components.

This section of the presentation emphasized the structure of an N-channel polyphase receiver. A similar exposition can be mounted for the N-channel polyphase transmitter. Rather than repeat the many steps that took us to the polyphase structure from the more conventional structure, we will merely comment that the transmitter is the dual process of the receiver. The dual process simply reverses all signal flow of the original process. In the dual structure, we enter the N-channel process at the FFT and leave the process by the polyphase commutator. Reversing the signal flow results in a process that up-samples and up-converts rather than one that down converts and down samples. The two processes are shown in figure 27.

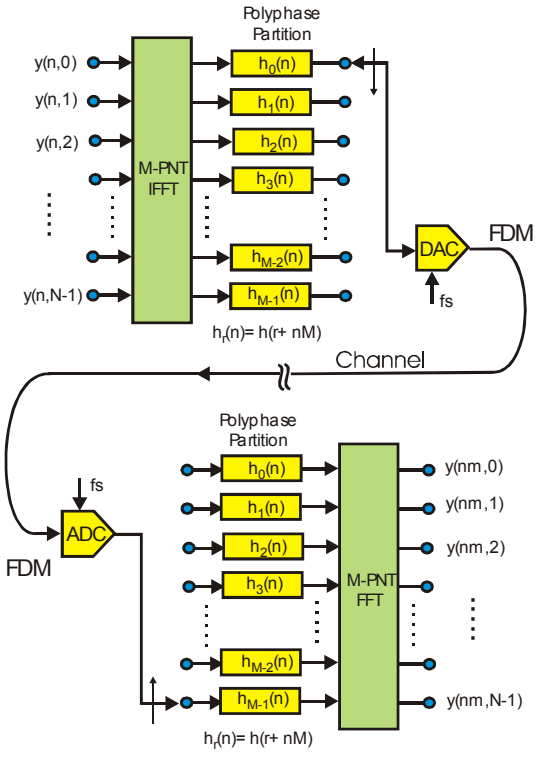

Figure 27. N-Channel Transmitter and N-Channel Receiver: Dual Circuits Formed With Polyphase Filters, FFT, and Commutator

Arbitrary Bandwidth, Spectral Spacing, and Output Sample Rates: We now address the interaction and coupling, or lack of coupling, between the parameters that define the polyphase filter bank [21]. We observe that the DFT performs the task of separating the channels after the polyphase filter so it is natural to conclude that the transform size is locked to the number of channels and this is a correct assessment. We then note that the filter bandwidth is determined by the weights of the low pass prototype and that this bandwidth and spectral shape is common to all the channels. We comment on filter length later when we address total computation complexity of the polyphase channelizer.

In standard channelizer designs the bandwidth of the prototype is specified in accord with the end use of the channelizer outputs. For instance, when the channelizer is used as a spectral analyzer, the channels may be designed to have a specified pass band attenuation such as –3 dB, or –1 dB or –0.1 dB at their crossover frequency and have a specified stop band attenuation at their adjacent center frequency. Overlap of adjacent channel responses permits a narrow band input signal to straddle one or more output channels, which is a common occurrence in the spectral analysis of signals with arbitrary bandwidth and center frequencies. On the other hand, when a channelizer is used to separate adjacent communication channels which are characterized by known center frequencies and known controlled, non-overlapping bandwidths, the channelizer must preserve separation of the channel outputs. Inadequate adjacent channel separation results in adjacent channel interference. Typical spectral responses for channel bandwidths corresponding to the two scenarios just described are shown in figure 28.

The polyphase filter channelizer uses the input M-to-1 resampling to alias the spectral terms residing at multiples of the output sample rate to base band. This means that for the standard polyphase channelizer, the output sample rate is the same as the channel spacing. For the case of the spectral analyzer application operating at this sample rate permits aliasing of the band edges into the down sampled pass band. When operated in this mode, the system is called a maximally decimated filter bank. For the case of the communication channelizer, operating at this rate satisfies the Nyquist criterion, permitting the separation of the channels with an output rate that avoids band edge aliasing. An example of a spectrum that would require this mode of operation is the Quadrature Amplitude Modulation (QAM) channels of a digital cable system. Here the channels are separated by 6-MHz centers and operate with 20%

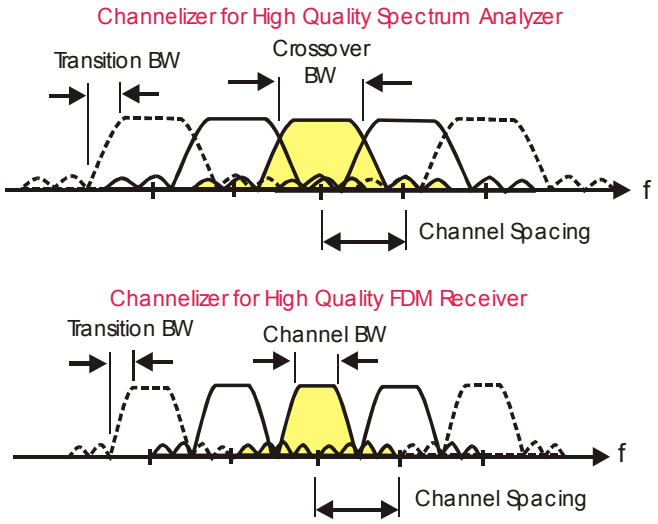

Figure 28. Spectral Characteristics of Two Channelizers with Same Channel Spacing One for Spectral Analysis and one for FDM Channel Separation

excess bandwidth, square-root Nyquist shaped spectra, at symbol rates of 5.0 MHz. The sampled data rate from a cable channelizer would be 6.0 Mega samples/sec and would satisfy the Nyquist sample criterion.

Another example used on the upstream cable link (subscriber to head end) is a set of channels operating at 128 kHz symbol rate, with square root Nyquist spectra having 50% excess bandwidth. The channels are separated by 192 kHz, which matches the two-sided bandwidth of the shaped channel. Here too, when operated at 192-kHz the sample rate, matched to the channel spacing, satisfies the Nyquist sample rate criterion. Systems that channelize and supply samples of the Nyquist shaped spectra most often present the sampled data to an interpolator to resample the time series from the collected Nyquist rate to two times the symbol rate. For the two example cited earlier, the 6 Ms/s, 5-Msymbol TV signal would have to be resampled by 5/3 to obtain 10 Ms/s, and the 192 ks/s, 128 K-symbol reverse channel signal would have to be resampled by 4/3 to obtain 256 ks/s. These are not difficult tasks and it is done quite regularly in single channel receivers. This represents significant computational burden if we are to perform this interpolative resampling for every channel.

The conventional way we use the M-path polyphase filter bank is to deliver M-input samples to the M-paths and then compute outputs from each channel at the rate fs/M. The thought may occur to us, “Is it possible to operate the polyphase filter bank in a manner that the output rate is higher than one M-th of the input rate?” For instance, can we operate the bank so that we deliver M/2 inputs prior to computing an output sample rather than delivering M input samples before computing an output sample? Increasing the output sample rate of the polyphase channel bank by a factor of two makes subsequent interpolation tasks less expensive since the spectra of the output signals would already be oversampled by a factor two with increased spectral separation. Operation in this mode would also permit channelization of overlapped channels without aliasing of the spectral transition bands. The alias free partition is handy in applications requiring perfect reconstruction of the original time series from spectrally partitioned sub channels, a requirement sometimes imposed in receivers used for Electronic Warfare (EW) applications. For the record, a polyphase filter bank can be operated with an output sample rate any rational ratio times the input sample rate. With minor modifications the filter can be operated with totally arbitrary ratios between input and output sample rates. This is true for the sample rate reduction imbedded in a polyphase receiver as well as for the sample rate increase embedded in a polyphase transmitter.

We first examine the task of increasing the output sample rate from the polyphase filter bank from fs/M to 2 fs/M. We accomplish this by controlling the commutator delivering input data samples to the polyphase stages. We normally deliver M inputs to the M-stage filter by delivering successive input samples starting at port M-1 progressing up the stack to port 0 and by doing so deliver M inputs per output for an M-to-1 down sampling. To obtain the desired (M/2)-to-1 down sampling, we deliver M/2 successive input samples starting at port (M/2)-1 progressing up the stack to port 0. The M/2 addresses to which the new M/2 input samples are delivered are first vacated by their former contents, the M/2 previous input samples. All the samples in the two-dimensional filter undergo a serpentine shift of M/2 samples with the M/2 samples in the bottom half of the first column sliding into the M/2 top addresses of the second column while the M/2 samples in the top half of the second column slide into the M/2 addresses in the bottom half of the second column and so on. This is equivalent to performing a linear shift through the prototype one-dimensional filter prior to the polyphase partition. In reality, we do not perform the serpentine shift but rather perform a swap of two memory banks as shown in figure 29 for successive sequences of length 32 being delivered to a filter bank with 64 stages.

We continue this discussion with comments on the 64-stage example. After each 32-point data sequence is delivered to the partitioned 64-stage polyphase filter the outputs of the 64-stages are computed and conditioned for delivery to the 64-point FFT. The data shifting into the polyphase filter stages causes a frequency dependent phase shift of the form shown in eq-19. The time delay due to shifting is nT where n is the number of samples, and T is the interval between samples. The frequencies of interest are integer multiple “k” of 1/M-th of the sample

rate 2π/T. Substituting these terms in eq-19 and canceling terms, we obtain the frequency dependent phase shift shown in eq-20. Here we see that for time shifts "n" equal to multiples of M, the phase shift is a multiple of 2π and contributes no offset to the spectra observed at the output of the FFT. The M-sample time shift is the time

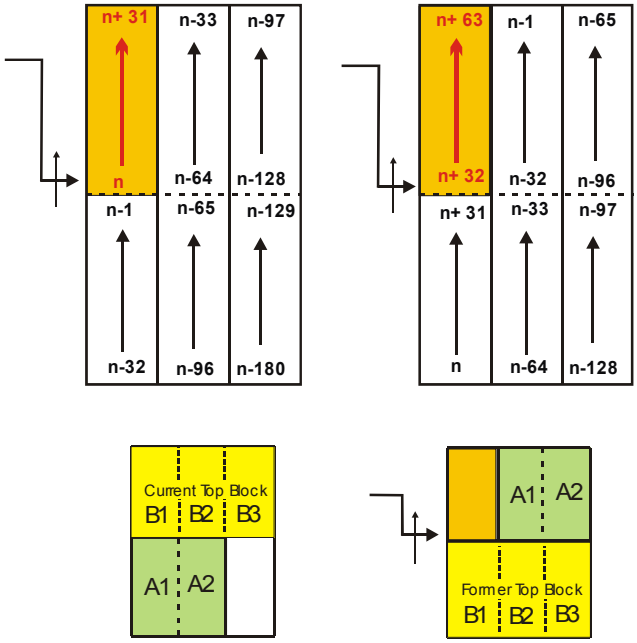

Figure 29. Data Memory Loading for Successive 32-Point Sequences in a 64-Stage Polyphase Filter

shift applied to the data in the normal use of the polyphase filter. Now suppose that the time shift is M/2 time samples. When substituted in eq-20 we find a frequency dependent phase shift of kπ from which we conclude that odd indexed frequency terms experience a phase shift of π radians for each successive N/2 shift of input data.

$$\theta(\omega) = \Delta t \cdot \omega \qquad (19)$$

$$\theta(\omega) = nT \cdot k \frac{1}{M} \frac{2\pi}{T} = \frac{nk}{M} 2\pi \qquad (20)$$

This π radian phase shift is due to the fact that the odd indexed frequencies alias to the half sample rate when the input signal is down sampled by M/2. We can compensate for the alternating signs in successive output samples by applying the appropriate phase correction to the spectral data as we extract successive time samples from the odd-indexed frequency bins of the FFT. The phase correction here is trivial, but for other down sampling ratios, the residual phase correction would require a complex multiply at each transform output port. Alternatively, we can cancel the frequency dependent phase shift by applying a circular time shift on N/2 samples to the vector of samples prior to their presentation to the FFT. As in the case of the serpentine shift of the input data, the circular shift of the polyphase filter output data is implemented as a data swap. This data swap occurs on alternate input cycles and a simple two-state machine determines for which input cycle the output data swap is applied. This option is shown in figure 30.

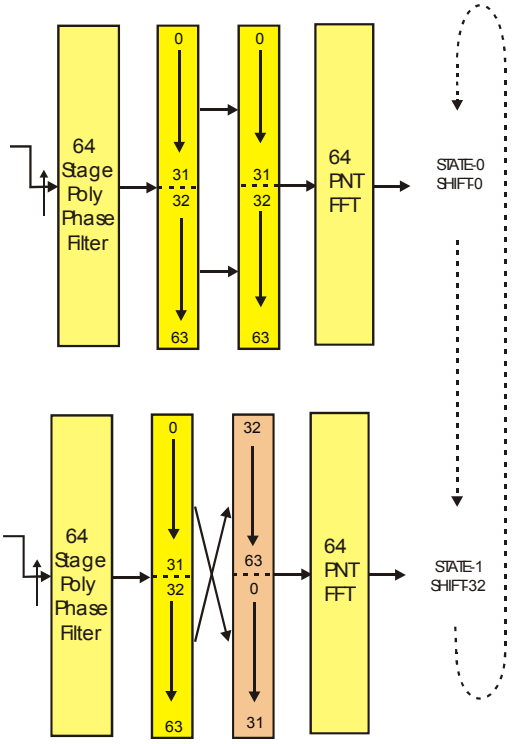

Figure 30. Cyclic Shift of Input Data to FFT to Absorb Phase Shift Due to Linear Time Shift of Data Through Polyphase Filter

We are now prepared to examine the process that permits resampling of the polyphase filter bank by any rational ratio. We first demonstrated the modification to the standard polyphase structure to support N/2 down sampling. The modifications involved a serpentine shift of input memory and a circular shift of output memory that are both implemented by data swaps.

There are relatively few papers in the open literature that describe arbitrary resampling embedded in the polyphase filter bank. Our group at SDSU had written an application note in 1989 [3] that demonstrated how to obtain arbitrary sample rates from the polyphase filter bank by use of the commutator mechanism with serpentine data shifts just described. An earlier contribution [22] presented the technique for absorbing the phase shifts

at the output of the FFT as circular shifts of the data vector at the input to the FFT. Numerous papers have been written describing arbitrary resamplers for baseband interpolators. We believe this paper is the first open literature description of the two merged techniques for polyphase channelizers. The technique is based on the observation that the commutator is the component in the polyphase filter bank that effects and controls the resampling, not the spacing between the adjacent channels. This is true even though it was the resampling process that first guided us to select the channel spacing so we could access the aliasing to baseband. As we just demonstrated, there are two modifications to the polyphase-resampling filter required to obtain arbitrary resampling. These modifications would normally lead to an exercise in time varying reside index mapping of the two-dimensional input data array. If we limit the presentation to the index mapping process we would develop little insight into the process and further would be bored to tears. Instead we derive and illustrate the modifications by examining a specific example and observe the process develop.

Second Example Processing Task: Here we describe a more general resampling channelizer and present the process that guides us to the solution. The problem is this: we have a signal containing 50 FDM channels separated by 192 kHz centers containing symbols modulated at 128 kHz by square-root Nyquist filters with 50% excess bandwidth. Our task is to base-band channelize all 50 channels and output data samples from each channel at 256 ks/s, which is two samples per symbol. We start by selecting a sample rate and transform size matched to the channel spacing. We select a 64-point FFT to span the 50 channels with the excess bandwidth allocated to the analog anti-alias filter. Thus the sample rate for the collected spectra is 64 times the 192 kHz channel spacing or 12.288 MHz. These are complex samples formed from either a base band block conversion or a digital down conversion and resampling from a digital IF, often centered at the quarter sample rate. The desired output sample rate is 2 times 128 or 256 kHz. The ratio between the input and output sample rates is the resampling ratio, which is 12288/256 or 48-to-1. Thus our task is to use the 64 point DFT to separate and deliver 50 of the possible 64 channels spanned by the sample rate, but to deliver one output sample for every 48 input samples. Figure 31 is a block diagram of the original maximally decimated version of the 64-stage polyphase channelizer and the modified form of the same channelizer. The difference in the two systems resides in the block inserted between the 64-stage polyphase filter and the 64-point FFT. Remarkably, the inserted block performs no computation but rather only performs a set of scheduled circular buffers shifts. We are about to develop and describe the operation of the circular buffer stage and state machine scheduler.

Our first task is to modify the input commutator to support the 48-to-1 down sample rather than the standard 64-to-1 down sample. This is an almost trivial task. We arrange for the modified resampling by keeping the 64-path filter but stripping 16-ports from the commutator. The commutator for the standard 64-point polyphase filter starts at port 63 and delivers 64 successive inputs to ports 64, 63, 62, and so on through 0, the modified commutator starts at port 47 and delivers 48 successive inputs to ports 47, 46, 45, and so on through 0. Input memory for the 64-path filter must be modified to support this shortened commutator input schedule. The mapping structure of the reindexing scheme is best seen in the original, one-dimensional, prototype filter shown in figure 32 and then transferred to the two-dimensional polyphase partition.

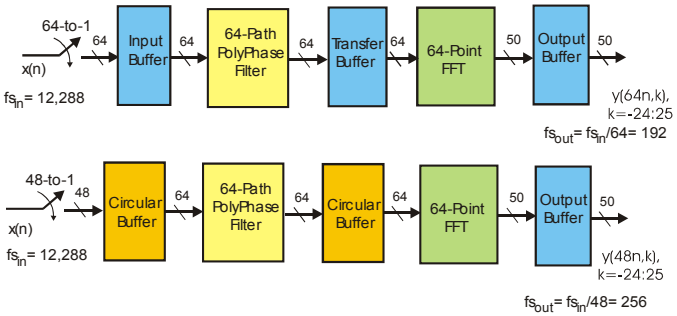

Figure 31. Maximally Decimated Filter Bank Structure and Modified Two-Sample-per-Symbol Filter Bank Structure

Figure 32 presents the memory content for a sequence of successive 48-point input data blocks presented to the 64-point partitioned prototype filter. In this figure we have indicated the interval of 64-tap boundaries that become the columns of the two-dimensional array as well as the boundaries of successive 48-point input blocks that are presented to the input array. Successive input blocks start loading at address 47 and work up to address 0. The beginning and end of this interval are denoted by the tail and arrow respectively, of the left most input interval in the filter array. As each new 48-point input array is delivered, the earlier arrays must shift to the right. These shifting array blocks cross the 64-point column boundaries hence move to adjacent columns in the equivalent two-dimensional partition. This crossing can be visualized as a serpentine shift of data in the two dimensional array, or equivalently as a circular row buffer down shift of 48 rows in the poly phase memory with a simultaneous column buffer right shift of the input data column. The operation of this circular buffer is illustrated in figure 33, which indicates the indices of in-

put data for two input cycles. Here we see that between two successive input cycles the rows in the top one-fourth of memory translates to the bottom fourth while the bottom three-fourths of rows translates up one-fourth of memory. We also see that the columns in the bottom three-fourths shift to the right on column during the circular row translations. The next input array is loaded in the left most column of this group of addresses.

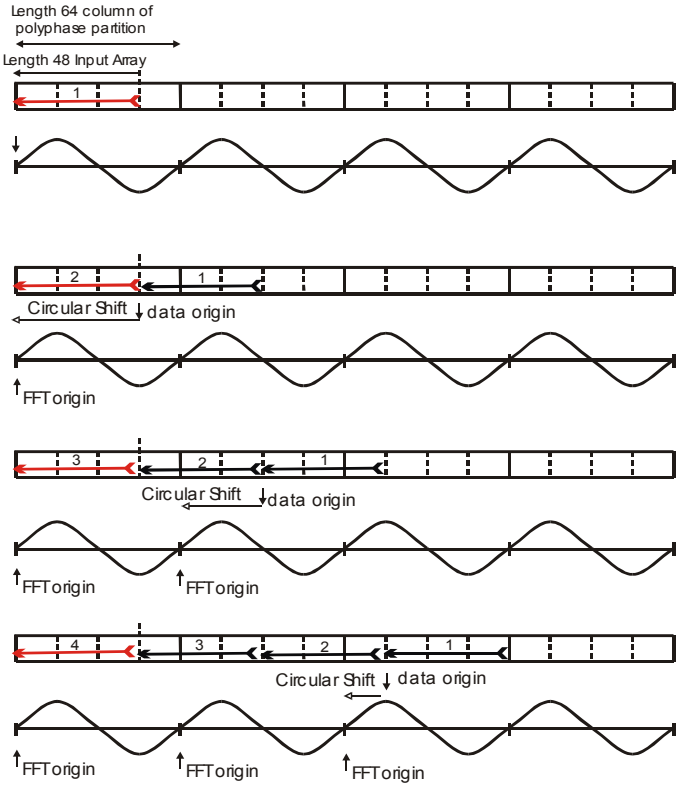

Figure 32. Memory Contents for Successive 48-Point Input Data Blocks Into a 64-Point Prototype Pre-Polyphase Partitioned Filter and FFT

Returning to figure 32, the one-dimensional prototype, we note that every new data block shifts the input data origin to the right by 48 samples. The vector $\hat{y}(r,48n)$ formed as the polyphase filter output from all 64 path filters is processed by the FFT to form the vector $\hat{Y}(k,48n)$ of channelized (index k) output time series (index 48n). On each successive call to the FFT, the origin of the sinusoids in the FFT is reset to the beginning of the input array. Since the origin of the input array shifts to the right on successive inputs while the origin of the FFT simultaneously resets to the beginning of the input array, a precessing offset exists between the origins of the polyphase filter and of the FFT. We align the origins, removing the offsets, by performing a circular shift of the vector $\hat{y}(r,48n)$ prior to passing it to the FFT. Since the offset is periodic and is a known function of the input array index the circular offset of the vector can be scheduled and controlled by a simple state machine. Figure 33 shows the location of the two origins for four successive 48-point input arrays and the amount of circular offset required to align the two prior to the FFT. Note that the offset schedule repeats in four cycles, four being the number of input intervals of length 48 that is a multiple of 64.

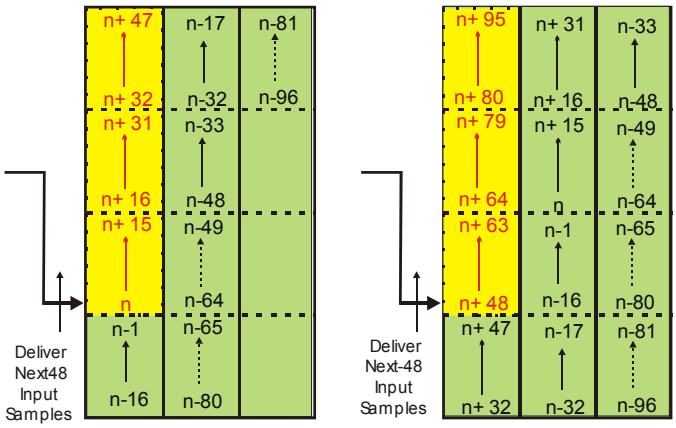

Figure 33. Memory Contents for Successive 48-Point Input Data Blocks Into a 64-Point Polyphase Filter

The cyclic shift for schedule for the array $\hat{y}(r,48n)$ prior to the FFT is shown in figure 34,

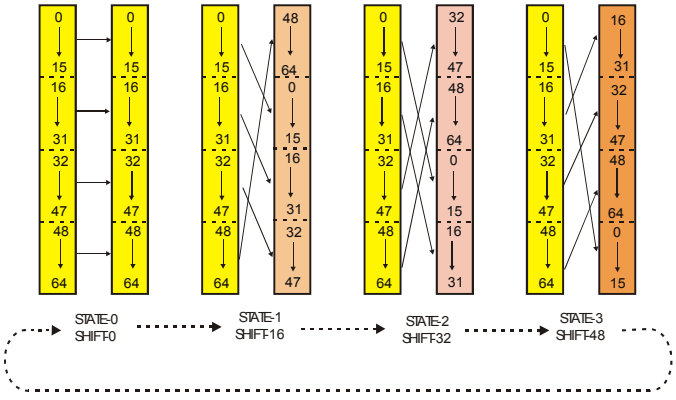

Figure 34 Cyclic Shift Schedule for Input Array to FFT

Appendix-II in the electronic version of this article contain a MATLAB m-file that implements the 50-stage polyphase channelizer described in this section. The program plots the impulse response of the prototype low pass filter and the frequency response at the input sample rate and the output sample rate. It then plots the spectrum and the input signal formed as a sum of sinusoids distributed over the span of frequencies matching the bandwidth of the channelizer. Finally a series of plots are generated showing the time series from 60 of the

channelizer outputs. Here we observe the transient of the filtering process in the occupied and unoccupied channels as well as the effect of filtering and aliasing spectral translation of the occupied channels. The code is well annotated and the reader is invited to modify the input time series and probe the performance of the channelizer with a variety of test signals.

Polyphase Computational Complexity: This section compares the computational workload required to implement a channelizer as a bank of conventional down converters with that required to implement the polyphase resampling approach. Here we call on the example of the 50-channel channelizer to supply actual numbers. We first determine the length of the finite impulse response (FIR) prototype filter required to satisfy the filter specifications. We note that the filter designed to operate at its input rate (12288 MHz) has its specifications controlled by its output rate (256 kHz). This is because the filter must satisfy the Nyquist sampling criterion after spectral folding as a result of the down sample operation. The length of any FIR filter is controlled by the ratio of input sample rate to filter transition bandwidth and the required out-of band attenuation as well as level of in-band ripple. The specifications of the filter used in this paper are listed in figure 35.

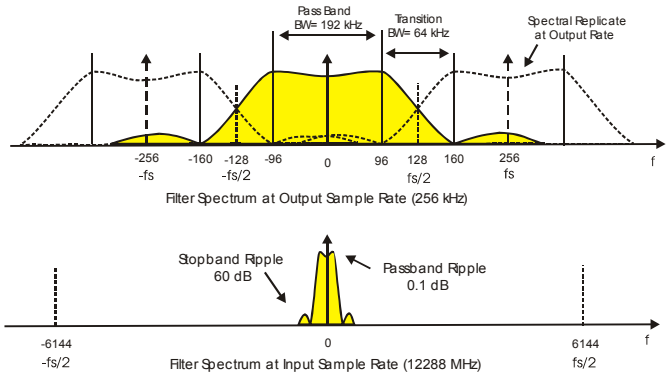

Figure 35. Spectral Characteristics of Prototype Filter at Output and Input Sample Rates

Standard design rules determine the filter length from the filter specification and for those indicated in figure 35, the filter length was found to be 512 samples and the Remez algorithm [23, 24] was used to supply the filter impulse response. A side comment is called for here. Filters designed by the standard Remez algorithm exhibit constant level out-of-band side-lobe levels. The spectra corresponding to these side-lobes folds back into the filter pass band under the resampling operation resulting in integrated side lobe levels considerably above the designed attenuation level. Modifying the penalty function of the Remez algorithm so that the side lobes fall off at approximately 6-dB per octave reduces the level of integrated side lobes. See, for instance, the filter design package QED-2000 available from Momentum Data Systems. An alternate scheme to control spectral side lobe decay rate involves modifying the end points of the filter impulse response to suppress the outlier samples responsible for the constant level spectral side lobes. See, for instance, the MATLAB examples in the appendices of the electronic version of this paper.

An important consideration and perspective for filters that have different input and output sample rates is the ratio of filter length (with units of operations/output) to resample ratio (with units of inputs/output) to obtain the filter workload (with units of operations/input). A useful comparison of two processes is the number of multiplies and adds per input point. We count a multiply and add with their requisite data and coefficient fetch cycles as a single processor operation and use the shorthand notation of “ops” per input.

A single channel of a standard down converter channelizer requires one complex multiply per input point for the input heterodyne and computes one complex output from the pair of 512 tap filters after collecting 48 inputs from the heterodyne. The 4 real ops per input for the mixer and the 2 (512/48) = 22 ops per input for the filter result in a per channel workload of 26 ops per input which occur at the input sample rate.

The polyphase version of the down converter collects 48 input samples from the input commutator, performs 1024 ops in the pair of 512 tap filters and then performs a 64-point FFT with its upper bound workload of $2N \log_2(N)$ real ops. The total workload of 1024 ops for the filter and 768 ops for the FFT results in 1792 ops performed once per 48 inputs for an input workload of 38 real ops/input. The higher workload per input is the consequence of forming 64 output channels in the FFT but preserving only 50 of them.

Now we must be careful: the workload per input sample for the standard channelizer was found to be 26 ops, and for the polyphase channelizer was found to be 38 ops: where is the promised advantage? The advantage is that the polyphase 38 ops per input built all 50 channels, and the standard down converter’s 26 ops per input built only one channel and has to be repeated 50 times. Isn’t that impressive? Comparing numbers we see that we should use the polyphase form even if we are forming just a few output channels, because the polyphase down converter requires less computations than even two standard down converters.

On a final note, when we compare hardware resources, we observe that the standard channelizer must build and apply 50 complex sinusoids as input heterodynes to the input data at the high input sample rate and further must store the 50 sets of down converted data for the filtering operations. On the other hand, the polyphase filter bank only stores one set of input data because the complex phase rotators are applied after the

filter rather than before and the phase rotators are applied at the filter output rate as opposed to the filter input rate.

Applications: The polyphase filter structure we have just reviewed and demonstrated can absorb a number of different system specifications. We now discuss some of the options. When the FDM signal is a collection of independent signals, as might occur in a multiple access application, the many signals do not share a common time reference or a common carrier frequency reference. The channels likely differ in small carrier and timing offsets that must be resolved in subsequent modem processing. For this situation it is appropriate for the channelizer to only perform the standard down conversion, bandwidth reduction, and sample rate reduction. For this application the filter passes the selected channel bandwidths without further spectral modification. This form of multiple channelization is also appropriate for analog modulation such as the 30 kHz narrow-band FM channels found in analog wireless Advanced Mobile Phone Service (AMPS), or the old standby SSB super-group.

In another application, the multiple channels are tightly coupled with negligible frequency offset and only differ in local time base for the modulation. For this case we can require the polyphase filter bank to perform the functions of the matched filter. The filter passes the selected channel bandwidths while performing the spectral shaping required of the matched filter. Since the data is formed at 2-samples per symbol, subsequent processing only has to align the phase of each channel and interpolate to the correct timing phase of the complex envelope. We note that the timing interpolation process requires significantly fewer resources than does the simultaneous interpolation matched filter and timing task.

In yet a third application, the multiple channels share a common clock and carrier reference hence are fully synchronous and exhibit negligible frequency offset. For this case we can ask the polyphase filter bank to perform the functions of the matched filter and to participate in timing recovery [25, 26]. Since in the example cited, we are down sampling by a factor of 48-to-1 there are 48 different contenders for the arbitrary origin of the process task. We can have the timing recovery loop advance or retard the start of the vector load of the polyphase filter. Using the processing origin to shift the sample time relative to the underlying modulation epochs offers fine grain time offsets equal to 1% of symbol interval without the need for additional interpolation.

The example we used to demonstrate the polyphase resampling channelizer implemented a 1/48 resampling in a 64-stage channelizer. Any ratio of small integers can be implemented using variations of the technique we have presented. Also the number of polyphase filter stages in a multi-channel receiver does not have to be large to warrant the application of the process described here. For instance, we have designed a number of polyphase resampling structures for 3rd generation (3G) [27] wireless applications that employ from three to eleven channels, using 5-point and 15-point transforms. These applications have required sample rate changes from 3.84 MHz to 6.144 MHz and to 15.36 MHz requiring ratios of 3-to-5 and of 2-to-5.

In a fashion similar to the process we have presented here to design down-sampling polyphase receiver channelizers we can also design polyphase up-sampling channelized transmitters. In fact when we design polyphase receivers, we are often obliged to design polyphase transmitters to test the receivers. An unexpected result discovered when we designed the transmitters for a given receiver is that they are not each other's duals because they are, in fact, not performing the inverse operators. For instance, in the receiver example we developed here, we formed a 48-to-1 down sampler in a 64-point DFT, which is a ratio of 3/4. The modulator on the other hand forms channels at 128 kHz symbol rate at a common sample rate of 64 times 192 kHz, an up sample of 96 in the 64-point DFT, which is a ratio of 2/3.

Finally, we note that there are filter structures [28] that permit the substitution of recursive polyphase filters for the non-recursive filter we have examined in this paper. The recursive filter options offer a reduction in workload by a factor of 3 to 6 and are available with both non-uniform phases and an equal-ripple approximation to linear phase. A minor drawback here is that the recursion in the filter prohibits computation pipeline delay, which limits the maximum output sample rate to the range of 200 to 400 MHz.

Conclusions: We have presented a description of the process by which a multichannel polyphase filter bank can simultaneously perform the uncoupled tasks of down conversion, bandwidth limiting, and sample rate change. We included a tutorial derivation of the polyphase filter bank as a sequence of transformations that rearrange the operations of mixing, filtering and resampling to obtain remarkably efficient processing structures. The sequence of transformations included application of the equivalency theorem, alias based spectral translation, sometimes referred to as IF sampling and of the noble identity. We also demonstrated that the ratio of input sample rate to output sample rate could differ from the conventional resampling ratio, the number of stages in the polyphase partition. The modification to the conventional polyphase channelizer required the insertion of circular buffers between the input commutator and the polyphase filter and the insertion of a second circular buffer between the output of the polyphase filter and the FFT phase rotator. A number of excellent tutorials [29, 30, 31, 32] are available in the literature that present aspects of some of the material presented here from a

number of different perspectives. Readers may find value in examining other author's perspectives after being exposed to ours.

We examined a specific example of a polyphase resampling channelizer to better illustrate the processes required to obtain arbitrary resampling in the filter bank. Comments on variations to the modified polyphase structure were included to give the reader a sense of the wide range of applicability of this process. Finally we compared the workload of a standard mixer based down converter filter bank with that of the polyphase resampling form. We invite readers to e-mail requests to the author for the MATLAB code that implements the 10-channel and the 50-channel channelizer described in this paper. The electronic version of this paper has the MATLAB files attached as appendices. A MATLAB script for an animated version of the 10-channel channelizer is also included in the appendices as is a 40-channel modulator that performs a 1-to-56 up sampling with a companion 40-channel demodulator that performs a 28-to-1 down sampling embedded in the channelization processes.

References:

1. N.J. Fliege, "Polyphase FFT Filter Bank for QAM Data Transmission", Proc. IEEE ISCAS'90, pp.654-657, 1990.
2. R.E. Chrochier and L.R. Rabiner, "Multirate Digital Signal Processing", Prentice Hall, 1981.
3. f.j. harris, "On the Relationship Between Multirate Polyphase FIR Filters and Windowed, Overlapped FFT Processing", Twenty-third Annual Asilomar Conference on Signals and Computers, 1989.
4. Gordon Moore, "Cramming more Components onto Integrated Circuits", Electronics, Vol. 38, No. 8, April 18 1965, also available at http://www.intel.com/research/silicon/mooreslaw.htm
5. C. Dick, "The Platform FPGA: Enabling the Software Radio", SDR'02, 2002 Software Defined Radio Technical Conference, 11, 12 November 2002, San Diego, CA.
6. A.M. Badda and M. Donati, "The Software Defined Radio Technique Applied to the RF Front-End for Cellular Mobile Systems", in Software Radio Technologies and Services", Editor Enrico Del Re, Springer-Verlog 2001.
7. f.j. harris, "On Measuring the Gain and Phase Imbalance and DC Offsets of Quadrature A-to-D Converters with an Adaptive Canceling Filter", Twenty –first Annual Asilomar Conference on Signals, Systems, and Computers, Pacific Grove, CA, 2-4 November 1987.
8. B. Sklar, "Digital Communications: Fundamentals and Applications", Second Edition, Prentice- Hall, 2002, Section 12.2.
9. J.G Proakis and D.G. Manolokis, "Digital Signal Processing: Principles, Algorithms, and Applications", Third Edition, 1996, Section 1.4.
10. D. Steinbrecher, "Broadband High Dynamic Range A/D Conversion Limitations", International Conference on Analogue to Digital and Digital to Analogue Conversion, 17-19 September 1991, Swansea, UK.
11. MAXIM Application Notes, "Defining and Testing Dynamic Parameters in High Speed ADC. Part –I and Part-II", Feb. 2001 and Jan 2001. http://www.maxim-ic.com/appnotes.cfm/appnote_number/728 http://www.maxim-ic.com/appnotes.cfm/appnote_number/729
12. M. Ribeyre, "Exploration of Transmultiplexers in Telecommunication Networks", IEEE Trans. Commun., COM-30 July 1982, pp.1493-1497.
13. M. Bellanger and J. Daguet, "TDM-FDM Transmultiplxer: Digital Polyphase and FFT", IEEE Trans. Commun., Vol. COM-22, Sept 1974, pp.1199-1204.
14. f.j. harris, "A Fresh View of Digital Signal Processing for Software Defined Radios, Part I", International Telemetry Conference (ITC), San Diego, CA, 21-24 October 2002.
15. f.j. harris, "A Fresh View of Digital Signal Processing for Software Defined Radios, Part II", International Telemetry Conference (ITC), San Diego, CA, 21-24 October 2002.
16. J. Wozencraft and I.M. Jacobs, "Principles of Communication Engineering", John-Wiley, 1967, Section 7.2.
17. P.P. Vaidyanathan, "Multirate Systems and Filter Banks", Prentice-Hall, 1993.
18. S.K. Mitra, "Digital Signal Processing: A Computer Based Approach", Second Edition, McGraw-Hill, 2001
19. J.L. Butler, in R.M. Foster (Ed), "Microwave Scanning Antennas" Volume III, Chapter 3, Academic Press, 1964.
20. Bruno Pattan, "Robust Modulation Methods & Smart Antennas in Wireless Communications", Prentice Hall, 2000, Chapter 9, "The Butler Matrix"
21. f.j harris and C. Dick, "Performing Simultaneous Arbitrary Spectral Translation and Sample Rate Change in Polyphase Interpolating and Decimating Filters in Transmitters and Receivers", 2002-Software Defined Radio Technical Conference, San Diego, CA, 11-12 November 2002.
22. D. Elliot, Editor, "Handbook of Digital Signal Processing: Engineering Applications", Academic Press, 1987, Chapter 8, "Time Domain Signal Processing with the DFT", pp 639-666.
23. J.H McClellan, T.W. Parks, and L.B. Rabiner, "A Computer Program for Designing Optimum FIR

Linear Phase Digital Filters", IEEE Trans. On Audio and Electroacoustics, Vol. AU-21, No. 6, pp. 506-526, December 1973.
24. MATLAB help menu for *remez.m*
25. f.j. harris and M. Rice, "Multirate Digital Filters for Symbol Timing Synchronization in Software Defined Radios", IEEE Journal on Selected Areas in Communications, Vol. 19, pp. 2346-2357, Dec. 2001.
26. M, Rice and f.j harris, "Polyphase Filter Banks for Symbol Synchronization in Sampled Data Receivers", MILCOM-2002, Anaheim, CA, 7-10 October 2002
27. D. Metri, "Reconfigurable Receivers for 3-G Applications", Master's Thesis, San Diego State University, December 2002
28. M. Renfors and T. Saramaki, "Recursive N-th Band Digital Filters, Parts I an II", IEEE Trans. CAS, Vol. 34, pp. 24-51, January 1987
29. P.P Vaidyanathan, "Multirate Digital Filters, Filter Banks, Polyphase Networks and Applications: A Tutorial", Proc. IEEE, Vol. 78, pp 56-93, January 1990
30. R. D. Koilpillai, T.Q. Nguyen, and P.P. Vaidyanathan, "Some Results in the Theory of Crosstalk Free Transmultiplexer", IEEE Trans. SP. Vol.39, pp. 2174-2183, October 1991.
31. H. Scheuermann and H. Göckler, "A Comprehensive Survey of Digital Transmultiplexing Methods", Proc. IEEE, Vol. 69, No. 11, November 1981, pp-1419-1450.
32. K.C. Zangi and R.D Koilpillai, "Software Radio Issues in Cellular Base Stations", IEEE Journal on Selected Areas in Communications, Vol. 17, No. 4, April 1999, pp. 561-573

Appendix-I: MATLAB Simulation of 10-Stage Polyphase Channelizers

```
function filter_ten(flag)
% filter_ten(flag) flag=0 for flat sidelobes, flag=1 for falling sidelobes

hh1=remez(169,[0 40 60 500]/500,[1 1 0 0],[1 100]);
frq=[0 40 60 99 100 149 150 199 200 249 250 299 300 349 350 399 400 449 450 500]/500;
gn= [1  1  0  0  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0];
pn= [  1   100    140     180     220     260     300     340     380     420];

hh2=remez(169,frq,gn,pn);
hh=hh1;
if flag==1
   hh=hh2;
end

figure(1)
subplot(2,1,1)
plot(hh)
grid
title('Impulse Response: Prototype Filter')
xlabel('Normalized time nT/T')
ylabel('Amplitude')
subplot(2,1,2)
plot((-0.5:1/1024:.5-1/1024)*1000,fftshift(20*log10(0.000001+abs(fft(hh,1024)))))
grid
axis([-500 500 -90 10])
title('Frequency Response: Prototype Filter')
xlabel('Frequency (kHz)')
ylabel('Log-Magnitude (dB)')
pause

x1=1*cos(2*pi*(0:2299)*10/1000);
x2=2+cos(2*pi*(0:2299)*15/1000);
x2=x2.*cos(2*pi*(0:2299)*100/1000);
x3=(3*cos(2*pi*(0:2299)*22/1000)+5*sin(2*pi*(0:2299)*6/1000));
x3=x3.*sin(2*pi*(0:2299)*300/1000);

xx=x1+x2+x3;
xx=[xx zeros(1,200)];

figure(2)
subplot(2,1,1)
plot(xx(1:200));
grid
title('Real Input Time Series')
xlabel('Normalized time nT/T')
ylabel('Amplitude')

subplot(2,1,2)
ww=kaiser(1024,8)';
ww=ww/sum(ww);
plot((-0.5:1/1024:.5-1/1024)*1000,fftshift(20*log10(abs(fft(xx(1:1024).*ww,1024)))))
grid
axis([-500 500 -90 10])
title('Spectrum of Real Input Series')
```

```
xlabel('Frequency (kHz)')
ylabel('Log-Magnitude (dB)')

pause
hold
plot((-0.5:1/1024:.5-1/1024)*1000,fftshift(20*log10(0.00001+abs(fft(hh,1024)))),'r')
pause
gg1=hh.*exp(j*2*pi*(-84.5:84.5)*100/1000);
plot((-0.5:1/1024:.5-1/1024)*1000,fftshift(20*log10(abs(fft(gg1,1024)))),'r--')
gg2=hh.*exp(j*2*pi*(-84.5:84.5)*200/1000);
plot((-0.5:1/1024:.5-1/1024)*1000,fftshift(20*log10(abs(fft(gg2,1024)))),'r--')
gg3=hh.*exp(j*2*pi*(-84.5:84.5)*300/1000);
plot((-0.5:1/1024:.5-1/1024)*1000,fftshift(20*log10(abs(fft(gg3,1024)))),'r--')
gg4=hh.*exp(j*2*pi*(-84.5:84.5)*400/1000);
plot((-0.5:1/1024:.5-1/1024)*1000,fftshift(20*log10(abs(fft(gg4,1024)))),'r--')
hold
pause

hh2=reshape(hh,10,17);

reg=zeros(10,17);
n2=1;
for nn=1:10:2500

   reg(:,2:17)=reg(:,1:16);
   reg(:,1)=flipud(xx(nn:nn+9)');
   for mm=1:10
      vv(mm)=reg(mm,:)*hh2(mm,:)';
   end
   yy(:,n2)=fft(vv)';
   n2=n2+1;
end


figure(3)
subplot(5,2,1)
plot(real(yy(1,:)))
grid
title('Time Series, Channels 1-5')
rr=axis;
rr(2)=250;
axis(rr);
subplot(5,2,2)
ww=kaiser(200,8)';
ww=ww/sum(ww);
plot((-0.5:1/256:.5-1/256)*100,fftshift(20*log10(abs(fft(yy(1,20:219).*ww,256)))))
axis([-50 50 -80 10])
grid
title('Spectra, Channels 1-5')

subplot(5,2,3)
plot(real(yy(2,:)))
grid
rr=axis;
rr(2)=250;
axis(rr);
```

```
subplot(5,2,4)
plot((-0.5:1/256:.5-1/256)*100,fftshift(20*log10(abs(fft(yy(2,20:219).*ww,256)))))
axis([-50 50 -80 10])
grid

subplot(5,2,5)
plot(real(yy(3,:)))
grid
rr=axis;
rr(2)=250;
axis(rr);

subplot(5,2,6)
plot((-0.5:1/256:.5-1/256)*100,fftshift(20*log10(abs(fft(yy(3,20:219).*ww,256)))))
axis([-50 50 -80 10])
grid

subplot(5,2,7)
plot(real(yy(4,:)))
grid
rr=axis;
rr(2)=250;
axis(rr);

subplot(5,2,8)
plot((-0.5:1/256:.5-1/256)*100,fftshift(20*log10(abs(fft(yy(4,20:219).*ww,256)))))
axis([-50 50 -80 10])
grid

subplot(5,2,9)
plot(real(yy(5,:)))
grid
rr=axis;
rr(2)=250;
axis(rr);

subplot(5,2,10)
plot((-0.5:1/256:.5-1/256)*100,fftshift(20*log10(abs(fft(yy(5,20:219).*ww,256)))))
axis([-50 50 -80 10])
grid

pause
figure(4)
subplot(2,2,1)
plot(0,0)
hold
for mm=1:10
   plot(0:1/64:1-1/64,(10*abs(fft(hh2(mm,:),64))))
end
hold
grid
title('Spectral Magnitude Response of Ten Polyphase Filters')
xlabel('Normalized Frequency')
ylabel('Amplitude')
axis([0 0.5 0.0 1.2])

subplot(2,2,3)
```

```
plot(0,0)
hold
for mm=1:10
    plot(0:1:16,0.5*mm+10*hh2(mm,:))
end
plot([7.55 8.45],[5.628 1.128],'r')
hold
grid
title('Impulse Response of Ten Polyphase Filters')
xlabel('Normalized Time')
axis([-1 17 -0.2 6])

subplot(2,2,2)
plot(0,0)
hold
for mm=1:10
   plot(0:1/64:1-1/64,unwrap((angle((fft(hh2(mm,:),64)))))/pi)
end
hold
grid
title('Spectral Phase Response of Ten Polyphase Filters')
axis([0 0.5 -9 0])
xlabel('Normalized Frequency (f/fs)')
ylabel('Phase Shift (\theta/2\pi)')

subplot(2,2,4)
plot(0,0)
hold
for mm=1:10
   vv=unwrap(angle(fftshift(fft(hh2(mm,:),64))))/(2*pi);
   vv2=64*filter([1 -1],1, vv);
   plot(0:1/64:1-1/64,fftshift(vv2))
end
hold
grid
title('Spectral Group Delay Response of Ten Polyphase Filters')
xlabel('Normalized Frequency (f/fs)')
ylabel('Group Delay (d\theta/d\omega) (Samples)')
%axis([0 0.5 -9 0])
axis([0 0.5 -9 -7])


% pause
% figure(5)
% plot(0,0)
% hold
% for mm=1:10
%    plot(0:1/64:1-1/64,unwrap((angle((fft(hh2(mm,:),64)))))/pi)
% end
% hold
% grid
% title('Spectral Phase Response of Ten Polyphase Filters')
% axis([0 0.5 -9 0])
% xlabel('Normalized Frequency (f/fs)')
% ylabel('Phase Shift (\theta/2\pi)')
% figure(6)
% plot(0,0)
```

```
% hold
% for mm=1:10
%    vv=unwrap(angle(fftshift(fft(hh2(mm,:),64))))/(2*pi);
%    vv2=64*filter([1 -1],1, vv);
%    plot(0:1/64:1-1/64,fftshift(vv2))
% end
% hold
% grid
% title('Spectral Group Delay Response of Ten Polyphase Filters')
% axis([0 0.5 -9 -7])
% xlabel('Normalized Frequency (f/fs)')
% ylabel('Group Delay (d\theta/d\omega) (Samples)')
%
% gg=get(gca);
% set(gca,'gridlinestyle','-')
```

Appendix-II: MATLAB SIMULATION OF 50-CHANNEL CHANNELIZER

```
function polyphase_50a
% demonstration of resampling polyphase filter bank

% building filter
hh=remez(511,[0 96 160 192*32]/(192*32),[1 1 0 0],[1 13]);
hh(1)=hh(2)/4;
hh(2)=hh(2)/2;
hh(512)=hh(1);
hh(511)=hh(2);

% examine prototype filter
figure(1)
subplot(2,1,1)
plot(hh)
axis([-10 520 -0.005 0.022])
title(' Impulse Response, Prototype Filter’)
grid
subplot(2,1,2)
plot((-0.5:1/2048:.5-1/2048)*64,fftshift(20*log10(abs(fft(hh,2048)))),'r')
grid
axis([-32 32 -80 10])
xlabel('Normalized Frequency f/f_C_h_a_n_n_e_l')
ylabel('log magnitude (dB)')
title('Frequency Response, Prototype Filter')
pause

% zoom to passband and compare with spectral copies at output rate
hold
het1=exp(j*2*pi*(0:511)/48);
het2=exp(-j*2*pi*(0:511)/48);
plot((-0.5:1/2048:.5-1/2048)*64,fftshift(20*log10(abs(fft(hh.*het1,2048)))))
plot((-0.5:1/2048:.5-1/2048)*64,fftshift(20*log10(abs(fft(hh.*het2,2048)))))
hold
axis([-2 2 -80 10])
title('Frequency Response, Prototype and Replicates at Output Rate')
pause
```

```
% building sample input signal: can replace with any other signal set
% channel 0 is empty.
ff=[1:25]+0.02;
xx=zeros(1,4848);
for rr=1:25
   xx=xx+exp(j*2*pi*(0:4847)*ff(rr)/64);
   xx=xx+exp(-j*2*pi*(0:4847)*ff(rr)/64);
end

% mapping one-dimensional prototype filter to two-dimensional filter
hh2=reshape(hh,64,8);

% defining two-dimensional array of commutated input samples
reg2=zeros(64,8);

% defining initial condition of 4-state state machine
state=1;

% filtering data

n_dat=1;
for nn=1:48:4800

% circularly roll and shift data array
temp=reg2(1:16,:);
reg2(1:48,2:8)=reg2(17:64,1:7);
reg2(49:64,:)=temp;

%load input data array
reg2(1:48,1)=xx(nn+47:-1:nn)';

% form inner products
for mm=1:64
  dd(mm)=reg2(mm,:)*hh2(mm,:)';
end

% circular shift output array dd
if state==1;
   state=2;
   dd_shift=dd;
elseif state==2;
   state=3;
   dd_shift=[dd(49:64) dd(1:48)];
elseif state==3;
   state=4;
   dd_shift=[dd(33:64) dd(1:32)];
elseif state==4;
   state=1;
   dd_shift=[dd(17:64) dd(1:16)];
end

% phase shift via fft

dd2(n_dat,:)=fftshift(fft(dd_shift));
n_dat=n_dat+1;
end
```

```
% output 60 channelized time series
figure(2)
for kk=1:12
   subplot(3,4,kk)
   plot(real(dd2(:,3+kk)))
   ss=sprintf('time series:, channel %2i ', -28+kk);
   title(ss)
   grid
   axis([0 100 -1.1 1.1])
end
figure(3)
for kk=1:12
   subplot(3,4,kk)
   plot(real(dd2(:,15+kk)))
   ss=sprintf('time series:, channel %2i ', -17+kk);
   title(ss)
   grid
   axis([0 100 -1.1 1.1])
end
figure(4)
for kk=1:12
   subplot(3,4,kk)
   if kk==6
      plot(real(dd2(:,27+kk)),'r')
   else
     plot(real(dd2(:,27+kk)))
   end
   ss=sprintf('time series:, channel %2i ', -6+kk);
   title(ss)
   grid
   axis([0 100 -1.1 1.1])
end
figure(5)
for kk=1:12
   subplot(3,4,kk)
   plot(real(dd2(:,39+kk)))
   ss=sprintf('time series:, channel %2i ', 5+kk);
   title(ss)
   grid
   axis([0 100 -1.1 1.1])
end
figure(6)
for kk=1:12
   subplot(3,4,kk)
   plot(real(dd2(:,51+kk)))
   ss=sprintf('time series:, channel %2i ', 16+kk);
   title(ss)

   grid
   axis([0 100 -1.1 1.1])
end
```

Appendix-III: MATLAB SIMULATION OF 40-CHANNEL CHANNELIZER

```
function receiver_40z;
% receiver_40z is a demo of a 40 channel receiver, demodulating 30 channels,
% of nominal symbol rate 20 MHz, separated by 28 MHz centers (1.4 times symbol rate)
% input sample rate is 40*28 = 1120 MHz.
% receiver performs a 40 point transform on the output of a 40-stage polyphase filter
% the polyphase filter operates at input rate but outputs at 2 samples/symbol or
% 40 MHz. The resampling rate is 1120/40 = 28-to-1, thus output from 40-channels are
% computed once for every 28 input samples. channelizer is not matched filter,
% prototype filter is 10% wider than two sided bandwidth of input signal to accommodate
% frequency uncertainty of separate channel centers.

%igo=0;
%while igo==0
%   igo=1;
%chan=input('enter channel number (-15 to +14) -> ');
%  if chan>14
%     igo=0;
%  end
%  if chan<-15
%     igo=0;
%  end
%end

%if chan<15
%   chan=chan+1;
%end
%if chan<0
%chan=chan+40;
%end


% signal generator section

hh_a=rcosine(1,112,'sqrt',0.4,6);
hh_b=hh_a(2:2:1345);
hh_b2=reshape(56*hh_b,56,12);
rr2=zeros(56*5,12);

xx1=2*floor(2*rand(28,100))-1;
xx1=xx1+j*(2*floor(2*rand(28,100))-1);
rr_a=zeros(1,40);


flag=0;
for nn=1:100
   %rr_a(10:29)=xx1(:,nn)';
   rr_a(1:14)=xx1(1:14,nn)';
   rr_a(27:40)=xx1(15:28,nn);
   rr_a(38)=0;
   rr_a(39)=0;

   %rr_a=fftshift(rr_a);
   rr_b=fft(rr_a);
   rr_b_ext=[rr_b rr_b rr_b rr_b rr_b rr_b rr_b];
```

```
   rr2(:,2:12)=rr2(:,1:11);
   rr2(:,1)=rr_b_ext';


   for mm=1:56
      xx_d((nn-1)*56+mm)=rr2(mm+56*flag,:)*hh_b2(mm,:)';
   end

   flag=flag+1;
   if flag==5;
      flag=0;
   end

end
figure(1)

subplot(2,1,1)
plot(real(xx_d(1:800)));
grid
title('real part of composite time signal')
subplot(2,1,2)
ww=kaiser(4096,8)';
ww=ww/sum(ww);
fxx=fftshift(20*log10(abs(fft(xx_d(1:4096).*ww))));
plot((-0.5:1/4096:.5-1/4096)*40,fxx)
hold
plot((-51/4096:1/4096:51/4096)*40,fxx(2049-51:2049+51),'r')
hold
grid
axis([-20 20 -60 10])
title('Spectrum: composite time signal')
pause

%xx=exp(j*2*pi*(0:5600)*chan/40);
%xx=xx+exp(j*2*pi*(0:5600)*5.001/40);
%xx(1000:1500)=zeros(1,501);

xx=xx_d;

%receiver section

%hh=remez(319, [0 14 27 560]/560,[1 1 0 0]);
ff=[0 12 17 56 57 84 85 112 113 140 141 168 169 196 197 224 225 252 253 280 281 308 309
336 337 364 365 392 393 420 421 448 449 476 477 504 505 532 533 560]/560;
gg=[1  1  0  0  0  0  0  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 ];
dd=[  1     6     7     9      11      13      15      17      19      21     23
25       27      29      31      33      35      37      39      41];

hh=remez(599,ff,gg,dd);

hh2=reshape(hh,40,15);
rr=zeros(40,15);

tt=1;
flg=1;
for nn=1:28:5600-28
```

```
   temp=rr(1:12,:);
   rr(1:28,2:15)=rr(13:40,1:14);
   rr(1:28,1)=flipud(xx((nn-1)+1:(nn-1)+28)');
   rr(29:40,:)=temp;

   for mm=1:40
      y(mm)=rr(mm,:)*hh2(mm,:)';
   end
   if flg==1
      r1=fftshift(fft(y));
      flg=2;
   elseif flg==2
      r1=fftshift(fft([y(29:40) y(1:28)]));
      flg=3;
    elseif flg==3
      r1=fftshift(fft([y(17:40) y(1:16)]));
      flg=4;
    elseif flg==4
      r1=fftshift(fft([y(5:40) y(1:4)]));
      flg=5;
    elseif flg==5
      r1=fftshift(fft([y(33:40) y(1:32)]));
      flg=6;
    elseif flg==6
      r1=fftshift(fft([y(21:40) y(1:20)]));
      flg=7;
    elseif flg==7
      r1=fftshift(fft([y(9:40) y(1:8)]));
      flg=8;
    elseif flg==8
      r1=fftshift(fft([y(37:40) y(1:36)]));
      flg=9;
    elseif flg==9
      r1=fftshift(fft([y(25:40) y(1:24)]));
      flg=10;
    elseif flg==10
      r1=fftshift(fft([y(13:40) y(1:12)]));
      flg=1;
end

      yy(:,tt)=r1';

      tt=tt+1;
end

figure(2)
for kk=1:30
   subplot(5,6,kk)
   if kk==16
      plot(real(yy(kk+5,:)),'r')
   else
      plot(real(yy(kk+5,:)))
   end

grid
axis([0 200 -10 10])
end
```

```
figure(3)

ww=kaiser(199,7)';
ww=ww/sum(ww);
for kk=1:30
   subplot(5,6,kk)
   if kk==16
plot((-0.5:1/256:.5-1/256)*2,fftshift(20*log10(abs(fft(yy(kk+5,:).*ww,256)))),'r')
else
plot((-0.5:1/256:.5-1/256)*2,fftshift(20*log10(abs(fft(yy(kk+5,:).*ww,256)))))
end

grid
axis([-1 1 -60 10])
end
subplot(5,6,16)
hold
plot([-0.95 0.95 0.95 -0.95 -0.95],[-59 -59 9 9 -59],'r')
hold

%pause
%figure(4)
%subplot(2,2,1)
%plot(hh)
%grid

%title('prototype receiver filter')

%subplot(2,2,3)
%plot((-0.5:1/2048:.5-1/2048)*40,fftshift(20*log10(abs(fft(hh,2048)))));
%grid
%axis([-20 20 -80 10])
%title('spectrum: receiver filter')

%subplot(2,2,2)
%plot(hh_b)
%grid

%title('prototype transmitter filter')

%subplot(2,2,4)
%plot((-0.5:1/2048:.5-1/2048)*40,fftshift(20*log10(abs(fft(hh_b/sum(hh_b),2048)))));
%grid
%axis([-20 20 -80 10])
%title('spectrum: transmitter filter')
```

Appendix-IV: MATLAB ANIMATED SIMULATION OF 10-CHANNEL CHANNELIZER (CALLS filter_ten_a_call IN-APPENDIX-V.)

```
%filter_ten_a

% Animated spectra and time response of 10-stage polyphase filter bank.
% Can move single tone through filter bank via a slider control or by a scheduled sweep

clear all

freq=0.1;
 xx= exp(j*2*pi*(0:2499)*freq);

figure(1)
subplot(6,2,1)
d_in=zeros(1,100);
plot(0:99,real(d_in));
grid
axis([0 100 -1.1 1.1]);

subplot(6,2,3)
d_1=zeros(1,100);
plot(0:0.1:9.9,d_1);
grid
axis([0 10 -1.1 1.1]);

subplot(6,2,4)
d_2=zeros(1,100);
plot(0:0.1:9.9,d_2);
grid
axis([0 10 -1.1 1.1]);

subplot(6,2,5)
d_3=zeros(1,100);
plot(0:0.1:9.9,d_3);
grid
axis([0 10 -1.1 1.1]);

subplot(6,2,6)
d_4=zeros(1,100);
plot(0:0.1:9.9,d_4);
grid
axis([0 10 -1.1 1.1]);

subplot(6,2,7)
d_5=zeros(1,100);
plot(0:0.1:9.9,d_5);
grid
axis([0 10 -1.1 1.1]);

subplot(6,2,8)
d_6=zeros(1,100);
plot(0:0.1:9.9,d_6);
grid
axis([0 10 -1.1 1.1]);

subplot(6,2,9)
d_7=zeros(1,100);
plot(0:0.1:9.9,d_7);
grid
```

```
axis([0 10 -1.1 1.1]);

subplot(6,2,10)
d_8=zeros(1,100);
plot(0:0.1:9.9,d_8);
grid
axis([0 10 -1.1 1.1]);

subplot(6,2,11)
d_9=zeros(1,100);
plot(0:0.1:9.9,d_9);
grid
axis([0 10 -1.1 1.1]);

subplot(6,2,12)
d_10=zeros(1,100);
plot(0:0.1:9.9,d_10);
grid
axis([0 10 -1.1 1.1]);

freq=0.1;
flag1=1;
% flag1=1 slider control, flag1=0 scheduled sweep
flag2=1;
% flag2=1 time display, flag2=0 frequency display

slider_1=uicontrol('style','slider','units','normalized','pos',[0.65 0.90 0.19
0.028],...
         'min',-0.4,'max',+0.4,'value',freq,...
         'callback',['freq=0.01*round(100*get(slider_1,''value''));',...
             'set(slider_1_cur,''string'',num2str(freq)),',...
             'set(gca,''view'',[0 0.5]),',...
             'filter_ten_a_call(freq,flag1,flag2)']);

slider_1_min=uicontrol('style','text','units','normalized','pos',[0.62 0.90 0.025
0.025],...
    'string', num2str(get(slider_1,'min')));

slider_1_max=uicontrol('style','text','units','normalized','pos',[0.84 0.90 0.025
0.025],...
    'string',num2str(get(slider_1,'max')));

slider_1_cur=uicontrol('style','text','units','normalized','pos',[0.78 0.87 0.055
0.025],...
    'string',num2str(0.01*round(100*get(slider_1,'value'))));

slider1_title=uicontrol('style','text','units','normalized','pos',[0.690 0.87 0.10
0.025],...
    'string','Center Frequency');


h30=uicontrol('style','pushbutton','string','SLIDER','units','normalized',...
    'position',[0.6 0.830 0.055
0.030],'callback',['flag1=1;','filter_ten_a_call(freq,flag1,flag2)']);
h40=uicontrol('style','pushbutton','string','SWEEP','units','normalized',...
    'position',[0.67 0.830 0.055
0.030],'callback',['flag1=0;','filter_ten_a_call(freq,flag1,flag2)']);
```

```
h50=uicontrol('style','pushbutton','string','TIME','units','normalized',...
    'position',[0.74 0.830 0.055
0.030],'callback',['flag2=1;','filter_ten_a_call(freq,flag1,flag2)']);
h60=uicontrol('style','pushbutton','string','FREQ','units','normalized',...
    'position',[0.81 0.830 0.055
0.030],'callback',['flag2=0;','filter_ten_a_call(freq,flag1,flag2)']);
```

Appendix-V: MATLAB ANIMATED SIMULATION OF 10-CHANNEL CHANNELIZER (CALLED BY filter_ten_a IN APPEDIX-IV.)

```
function filter_ten_a_call(freq,flag1,flag2)
% called by filter_ten_a

if flag1==1
    n_dat=1200;
xx= exp(j*2*pi*(0:n_dat-1)*freq);
ww1=kaiser(100,6)';
ww1=ww1/sum(ww1);
nn_dat=100;


else
    n_dat=10000;
    frq=[0:1/2500:1 1-1/2500:-1/2500:-1 -1+1/2500:1/2500:-1/2500];
    phs=filter([1 0],[1 -1],frq);
    xx=exp(j*2*pi*phs*0.45);

ww1=kaiser(100,6)';
ww1=ww1/sum(ww1);
nn_dat=12;
end

if flag2==1

    subplot(6,2,1)
    plt0=plot(0:99,real(xx(1:100)),'linewidth',1.5,'erasemode','xor');
    grid
    axis([0 100 -1.1 1.1])
    title('Input Time Series')

    mm=0;
    txt=text(105,0.0,['Sample # ',num2str(mm,3)]);


    subplot(6,2,3)
    plt1=plot(0:99,zeros(1,100),'linewidth',1.5,'erasemode','xor');
    grid
    axis([0 10 -1.1 1.1])
    text(10.6, 0.3, 'bin(-1)')
     text(10.2, -0.3, '-0.15 -0.05')

    subplot(6,2,4)
    plt2=plot(0:99,zeros(1,100),'linewidth',1.5,'erasemode','xor');
    grid
```

```
axis([0 10 -1.1 1.1])
text(10.6, 0.3, 'bin(0)')
 text(10.2, -0.3, '-0.05 +0.05')

subplot(6,2,5)
plt3=plot(0:99,zeros(1,100),'linewidth',1.5,'erasemode','xor');
grid
axis([0 10 -1.1 1.1])
 text(10.6, 0.3, 'bin(-2)')
  text(10.2, -0.3, '-0.25 -0.15')

subplot(6,2,6)
plt4=plot(0:99,zeros(1,100),'linewidth',1.5,'erasemode','xor');
grid
axis([0 10 -1.1 1.1])
 text(10.6, 0.3, 'bin(+1)')
  text(10.2, -0.3, '+0.05 +0.15')

subplot(6,2,7)
plt5=plot(0:99,zeros(1,100),'linewidth',1.5,'erasemode','xor');
grid
axis([0 10 -1.1 1.1])
 text(10.6, 0.3, 'bin(-3)')
  text(10.2, -0.3, '-0.35 -0.25')

subplot(6,2,8)
plt6=plot(0:99,zeros(1,100),'linewidth',1.5,'erasemode','xor');
grid
axis([0 10 -1.1 1.1])
 text(10.6, 0.3, 'bin(+2)')
  text(10.2, -0.3, '+0.15 +0.25')

subplot(6,2,9)
plt7=plot(0:99,zeros(1,100),'linewidth',1.5,'erasemode','xor');
grid
axis([0 10 -1.1 1.1])
 text(10.6, 0.3, 'bin(-4)')
 text(10.2, -0.3, '-0.45 -0.35')

subplot(6,2,10)
plt8=plot(0:99,zeros(1,100),'linewidth',1.5,'erasemode','xor');
grid
axis([0 10 -1.1 1.1])
 text(10.6, 0.3, 'bin(+3)')
 text(10.2, -0.3, '+0.25 +0.35')

 subplot(6,2,11)
plt9=plot(0:99,zeros(1,100),'linewidth',1.5,'erasemode','xor');
grid
axis([0 10 -1.1 1.1])
 text(10.6, 0.3, 'bin(-5)')
  text(10.2, -0.3, '-0.45 +0.45')

subplot(6,2,12)
plt10=plot(0:99,zeros(1,100),'linewidth',1.5,'erasemode','xor');
grid
axis([0 10 -1.1 1.1])
```

```
     text(10.6, 0.3, 'bin(+4)')
      text(10.2, -0.3, '+0.35 +0.45')
else
    ww=kaiser(100,6)';
    ww=ww/sum(ww);
    hp=remez(99,[0 40 60 500]/500,[1 1 0 0],[1 1]);

    fhp=fftshift(abs(fft(hp,100)));
    subplot(6,2,1)
    plt0=plot(-0.5:1/100:0.5-
1/100,fftshift(abs(fft(xx(1:100).*ww))),'linewidth',3,'erasemode','xor');

    hold on
    plot(-0.5:1/100:.5-1/100,fhp,'r--')
    plot(-0.5:1/100:.5-1/100,[fhp(11:100) fhp(1:10)],'r--')
    plot(-0.5:1/100:.5-1/100,[fhp(21:100) fhp(1:20)],'r--')
    plot(-0.5:1/100:.5-1/100,[fhp(31:100) fhp(1:30)],'r--')
    plot(-0.5:1/100:.5-1/100,[fhp(41:100) fhp(1:40)],'r--')
    plot(-0.5:1/100:.5-1/100,[fhp(61:100) fhp(1:60)],'r--')
    plot(-0.5:1/100:.5-1/100,[fhp(71:100) fhp(1:70)],'r--')
    plot(-0.5:1/100:.5-1/100,[fhp(81:100) fhp(1:80)],'r--')
    plot(-0.5:1/100:.5-1/100,[fhp(91:100) fhp(1:90)],'r--')
         cc=get(gca,'children');
         set(cc,'linewidth',1.5)
    hold off
    grid
    axis([-0.5 0.5 0 1.1])
    title('Input Spectrum and Channel Bandwidths')
    mm=0;
    txt=text(0.55,0.5,['Sample # ',num2str(mm,3)]);

    subplot(6,2,3)
    plt1=plot((-0.5:1/100:0.5-
1/100)/10,zeros(1,100),'linewidth',1.5,'erasemode','xor');
    grid
    axis([-0.1501 -0.0499 0  1.1])
    text(-0.043, 0.65, 'bin(-1)')
    text(-0.048, 0.35, '-0.15 -0.05')

    subplot(6,2,4)
    plt2=plot((-0.5:1/100:0.5-
1/100)/10,zeros(1,100),'linewidth',1.5,'erasemode','xor');
    grid
    axis([-0.05 0.05 0  1.1])
    text(0.057, 0.65, 'bin(0)')
     text(0.052, 0.35, '-0.05 +0.05')

    subplot(6,2,5)
    plt3=plot((-0.5:1/100:0.5-
1/100)/10,zeros(1,100),'linewidth',1.5,'erasemode','xor');
    grid
    axis([-0.25 -0.15 0  1.1])
     text(-0.143, 0.65, 'bin(-2)')
     text(-0.148, 0.35, '-0.25 -0.15')

     subplot(6,2,6)
```

```
    plt4=plot((-0.5:1/100:0.5-
1/100)/10,zeros(1,100),'linewidth',1.5,'erasemode','xor');
    grid
    axis([0.05 0.15 0  1.1])
    text(0.157, 0.65, 'bin(+1)')
    text(0.152, 0.35, '+0.05 +0.15')

    subplot(6,2,7)
    plt5=plot((-0.5:1/100:0.5-
1/100)/10,zeros(1,100),'linewidth',1.5,'erasemode','xor');
    grid
    axis([-0.3501 -0.2499 0  1.1])
    text(-0.243, 0.65, 'bin(-3)')
    text(-0.248, 0.35, '-0.35 -0.25')

    subplot(6,2,8)
    plt6=plot((-0.5:1/100:0.5-
1/100)/10,zeros(1,100),'linewidth',1.5,'erasemode','xor');
    grid
    axis([0.15 0.25 0  1.1])
    text(0.257, 0.65, 'bin(+2)')
    text(0.252, 0.35, '+0.15 +0.25')

    subplot(6,2,9)
    plt7=plot((-0.5:1/100:0.5-
1/100)/10,zeros(1,100),'linewidth',1.5,'erasemode','xor');
    grid
    axis([-0.45 -0.35 0  1.1])
    text(-0.343, 0.65, 'bin(-4)')
    text(-0.348, 0.35, '-0.45 -0.35')

    subplot(6,2,10)
    plt8=plot((-0.5:1/100:0.5-
1/100)/10,zeros(1,100),'linewidth',1.5,'erasemode','xor');
    grid
    axis([0.25 0.35 0  1.1])
    text(0.357, 0.65, 'bin(+3)')
    text(0.352, 0.35, '+0.25 +0.35')

    subplot(6,2,11)
    plt9=plot((-0.5:1/100:0.5-
1/100)/10,zeros(1,100),'linewidth',1.5,'erasemode','xor');
    grid
    axis([-0.55 -0.45 0  1.1])
    text(-0.443, 0.65, 'bin(-5)')
    text(-0.448, 0.35, '-0.45 +0.45')

    subplot(6,2,12)
    plt10=plot((-0.5:1/100:0.5-
1/100)/10,zeros(1,100),'linewidth',1.5,'erasemode','xor');
    grid
    axis([0.35 0.4501 0  1.1])
    text(0.457, 0.65, 'bin(+4)')
    text(0.452, 0.35, '+0.35 +0.45')
end
```

```
%hh=remez(169,[0 40 60 500]/500,[1 1 0 0],[1 100]);
frq=[0 40 60 99 100 149 150 199 200 249 250 299 300 349 350 399 400 449 450 500]/500;
gn= [1  1  0  0  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0];
pn= [  1   100    140     180     220     260     300     340     380     420];
hh=remez(169,frq,gn,pn);
hh2=reshape(hh,10,17);

reg=zeros(10,17);
n2=1;
rr=zeros(1,100);
yy=zeros(10,n_dat/10);
yy2=zeros(10,100);

ww2=ones(1,100)/100;
ww1=kaiser(100,6)';
ww1=ww1/sum(ww1);

for nn=1:10:n_dat-10
  rr=[fliplr(xx(nn:nn+9)) rr(1:90)];
   reg(:,2:17)=reg(:,1:16);
   reg(:,1)=flipud(xx(nn:nn+9)');
   for mm=1:10
      vv(mm)=reg(mm,:)*hh2(mm,:)';
   end
   yy(:,n2)=fft(vv).';
   n2=n2+1;
    yy2=[fft(vv).' yy2(:,1:99)];
   if flag2==1

  set(plt0,'xdata',0:99,'ydata',real(rr));
  set(plt1,'xdata',0:0.1:9.9,'ydata',real(yy2(10,:)));
  set(plt2,'xdata',0:0.1:9.9,'ydata',real(yy2(1,:)));
  set(plt3,'xdata',0:0.1:9.9,'ydata',real(yy2(9,:)));
  set(plt4,'xdata',0:0.1:9.9,'ydata',real(yy2(2,:)));
  set(plt5,'xdata',0:0.1:9.9,'ydata',real(yy2(8,:)));
  set(plt6,'xdata',0:0.1:9.9,'ydata',real(yy2(3,:)));
  set(plt7,'xdata',0:0.1:9.9,'ydata',real(yy2(7,:)));
  set(plt8,'xdata',0:0.1:9.9,'ydata',real(yy2(4,:)));
  set(plt9,'xdata',0:0.1:9.9,'ydata',real(yy2(6,:)));
  set(plt10,'xdata',0:0.1:9.9,'ydata',real(yy2(5,:)));

else

  set(plt0,'xdata',(-0.5:1/100:0.5-1/100),'ydata',fftshift(abs(fft((rr.*ww1)'))));
  set(plt1,'xdata',-0.1+(-0.5:1/100:0.5-
1/100)/10,'ydata',fftshift(abs(fft(yy2(10,1:nn_dat)/nn_dat,100))));
  set(plt2,'xdata',(-0.5:1/100:0.5-
1/100)/10,'ydata',fftshift(abs(fft(yy2(1,1:nn_dat)/nn_dat,100))));
  set(plt3,'xdata',-0.2+(-0.5:1/100:0.5-
1/100)/10,'ydata',fftshift(abs(fft(yy2(9,1:nn_dat)/nn_dat,100))));
  set(plt4,'xdata',+0.1+(-0.5:1/100:0.5-
1/100)/10,'ydata',fftshift(abs(fft(yy2(2,1:nn_dat)/nn_dat,100))));
  set(plt5,'xdata',-0.3+(-0.5:1/100:0.5-
1/100)/10,'ydata',fftshift(abs(fft(yy2(8,1:nn_dat)/nn_dat,100))));
  set(plt6,'xdata',+0.2+(-0.5:1/100:0.5-
1/100)/10,'ydata',fftshift(abs(fft(yy2(3,1:nn_dat)/nn_dat,100))));
```

```
  set(plt7,'xdata',-0.4+(-0.5:1/100:0.5-
1/100)/10,'ydata',fftshift(abs(fft(yy2(7,1:nn_dat)/nn_dat,100))));
  set(plt8,'xdata',+0.3+(-0.5:1/100:0.5-
1/100)/10,'ydata',fftshift(abs(fft(yy2(4,1:nn_dat)/nn_dat,100))));
  set(plt9,'xdata',-0.5+(-0.5:1/100:0.5-
1/100)/10,'ydata',fftshift(abs(fft(yy2(6,1:nn_dat)/nn_dat,100))));
  set(plt10,'xdata',+0.4+(-0.5:1/100:0.5-
1/100)/10,'ydata',fftshift(abs(fft(yy2(5,1:nn_dat)/nn_dat,100))));

end
 set(txt,'string',['Sample #  ',num2str(n2-1)])
pause(0.05)
end
```