SDF Domain

Authors:

J. T. Buck

E. A. Lee

D. G. Messerschmitt

S. Ha

1. Introduction

Synchronous dataflow (SDF) is a data-driven, statically scheduled domain in **Ptolemy**. It is a direct implementation of the techniques given in [1] and [2]. "Data-driven" means that the availability of **`Particle`**s at the inputs of a star enables it. Stars with no input ports are always enabled. "Statically scheduled" means that the firing order of the stars is determined once, during the startup phase. The firing order will be periodic.

1.1. Delays

In the SDF domain, a unit delay is simply an initial particle on an arc. This initial particle may enable a star, assuming the star requires one particle in order to fire. Hence, to avoid deadlock, all feedback loops much have delays. The SDF scheduler will flag the error if it finds a loop with no delays. There is currently no mechanism for setting the value of the initial particle. For most particle types, the initial value will be zero.

1.2. Number of iterations

When running an SDF universe or wormhole, the number of iterations may not be the same as the number of times a star in the system is fired. The number of invocations of each star depends on all the stars in the universe or wormhole. In order to be able to repeat the iterations, each iteration should leave all arcs (geodesics) in the same state after the iteration ends that they were in before the iteration began. In other words, the total number of particles stored on an arc should be the same after the iteration as before. Furthermore, every star should fire at least once in an iteration. These two requirements together determine the number of firings of a star in an iteration.

Consider for example a universe A-->FFT-->B, where FFT is the **`FFTCx`** star. This star consumes some number of particles (given by its *size* parameter) and produces some number of tokens (2^n, where n is its *order* parameter). Hence, one iteration consists of *size* firings of A, one firing of FFT, and 2^n firings of B. No fewer invocations would satisfy the above requirements. Note therefore that if you wish to compute only one FFT, then this system should be run through just one iteration.

1.3. Inconsistencies

It is not always possible to find a consistent number of firings for all stars in the system that returns the geodesics to their original state. For instance, suppose that the input and output of the FFTCx are added together using a AddCx star. Also suppose that $2^n \neq size$. Then no periodic schedule is possible. One interpretation is that we have tried to add two signals with different sample rates. The SDF scheduler detects this error, and refuses to run the system. It is sometimes possible to run the system under the DDF domain.

2. Writing SDF stars

All stars in the SDF domain must follow the basic SDF principle: the number of particles consumed or produced on any **`Porthole`** does not change while the simulation runs. These numbers are given for each porthole as part of the star definition. Most stars consume just one particle on each input and produce just one particle on each output. In these cases, no special action is required, since the porthole SDF properties will be set to unity by default. However, if the numbers differ from unity, the star definition must reflect this. For example, the **`FFTCx`** star has a *size* state (parameter) that specifies how many input samples to read. The value of that state specifies the number of samples required at the input in order for the star to be enabled. The following line in the start() function of the star is used to make this information available to the scheduler:

```
input.setSDFParams (int(size), int(size)-1);
```

The name of the input porthole is "input". The first argument to *setSDFParams* specifies how many samples are consumed by the star when it fires, which is the same as the number of samples required in order to enable the star. The second argument to *setSDFParams* specifies how many past samples (before the most recent one) will be accessed by the star when it fires.

If the number of particles produced or consumed is a constant independent of any states, then it may be declared right along with the declaration of the input. For example,

```
input {
      name { signalIn }
      type { complex }
      numTokens { 2 }
      desc { A complex input that consumes 2 input particles. }
}
```

declares an input that consumes two particles.

3. Accessing Inputs and Outputs

The mechanism for accessing inputs and outputs is explained in the section of the Almagest on the **Ptolemy** preprocessor language.

4. The Scheduler

The SDF scheduler determines the order of execution of stars in a system at start time. It performs most of its computation during its setup() phase. The default scheduler exactly implements the method described in [1] for sequential schedules. If there are sample rate changes in a program graph, some parts of the graph are executed multiple times. The default scheduler does not attempt to generate loops; it simply generates a linear list of blocks to be executed. For example, if star A is executed 100 times, the generated schedule includes 100 instances of A. A loop scheduler will include in its schedule (where possible) only one instance of A and indicate the repetition count of A somehow, say 100(A). For simulation, a long unstructured list might be tolerable, but not in code generation (The SDF schedulers are also used in the code generation for a single processor target).

By setting the **`loopScheduler`** target parameter, we can select a scheduler developed by J. Buck. Before applying the default scheduling algorithm, this algorithm collects actors into a hierarchy of clusters. This clustering algorithm consists of alternating a "merging" step and a "looping" step until no further changes can be made. In the merging step, blocks connected together are merged into a cluster if there is no sample rate change between them and the merge will not introduce deadlock. In the looping step, a cluster is looped until it is possible to merge it with the neighbor blocks or clusters. Since this looping algorithm is conservative, some complicated looping possibilities are not always discovered.

A more complicated looping algorithm was developed by adding postprocessing steps to the above algorithm to handle cases that it cannot handle. For lack of a better name, we call this technique "SJS scheduling", for the first initials of the designers (Shuvra Bhattacharyya, Joe Buck, and Soonhoi Ha). We apply the previously described scheduling algorithm as the first pass. In the second pass, we decompose the graph (Bhattacharyya’s contribution) so that the graph becomes acyclic [3]. The decomposed graphs are expanded to acyclic precedence graphs in which looping structures are extracted (Ha’s contribution). This scheduling option is selected when the **`loopTarget`** is chosen instead of the default SDF target.

The looping result can be seen by setting the **`logFile`** target parameter. That file will contain all the intermediate procedures of looping and the final scheduling result. The loop scheduling algorithms are usually used in code generation domain not in the simulation SDF domain. Refer to the CG domain documentation for detailed discussion.

References

[1] E. A. Lee and D. G. Messerschmitt, "Static Scheduling of Synchronous Data Flow Programs for Digital Signal Processing", *IEEE Trans. on Computers*, January 1987, **C-36(2)**.

[2] E. A. Lee and D. G. Messerschmitt, "Synchronous Data Flow" *IEEE Proceedings*, September, 1987.

[3] S. Bhattacharyya and E. A. Lee, "Scheduling Synchronous Dataflow Graphs for Efficient Looping," to appear in *J. of VLSI Signal Processing*, 1993.

SDF Stars and Galaxies

The source code for all standard SDF stars is in one of three directories:

- $PTOLEMY/src/domains/sdf/stars: The general-purpose stars. Because there are so many, these stars are divided into many sub-palettes, as described below.
- $PTOLEMY/src/domains/sdf/dsp/stars: Stars implementing digital signal processing and communication functions.
- $PTOLEMY/src/domains/sdf/image/stars: Stars that perform image processing functions.

By "standard" we mean the stars that are owned by **Ptolemy**. Because of their large number, their icons are divided into sub-palettes. When you open the main SDF palette in pigi, you will see one icon representing each subpalette. Look inside one of these icons, and you will see the icons representing the stars in the subpalette. The main palette is identified as $PTOLEMY/src/domains/sdf/icons/main.pal. The sub-palettes are:

- **signal sources**: Stars with no inputs that generate signals.
- **signal sinks**: Stars with no outputs used for graphing or printing signals or "playing" them to the workstation speaker.
- **arithmetic**: Basic arithmetic operations.
- **nonlinear functions**: Mathematical function operators, such as trigonometric functions.
- **control**: Stars such as fork, upsample, downsample, distributor, that manipulate the flow of particles. All the stars in this palatte operate on **ANYTYPE** of particle.
- **conversion**: Type conversion functions, such as real to complex and bits to integer.
- **dsp**: Signal processing functions.
- **image**: Image display and processing.
- **communcations**: Communication functions such as channel models, modulators, and encoders.

All galaxy definitions are stored in $PTOLEMY/src/domains/sdf/demo, but since they will almost always be accessed through a palette, this should not matter to a user. Below all stars and galaxies (blocks) in any of these palettes are listed. The location field (usually) indicates the palette in which the block can be found.

Note that any star library in **Ptolemy** should be viewed as a set of examples, not as an all inclusive set. Adding new stars is easy, so the library tends to contain only stars for which we have encountered a need. Many users will want to add their own.

NAME: `AddCx`

Output the sum of the inputs, as a complex value.

LOCATION: SDF main library

DOMAIN: `SDF` **(DERIVED FROM:** `SDFStar`)

VERSION: 2.6 (11/25/92)

AUTHOR: *J. T. Buck*

INPUTS: *input* (multiple), (complex)

OUTPUTS: *output* (complex)

NAME: `AddMotionVecs`

Over each block in the input image, superimpose an arrow indicating the size and direction of the corresponding motion vector.

LOCATION: SDF image library

DOMAIN: `SDF` **(DERIVED FROM:** `SDFStar`)

VERSION: 1.4 (11/25/92)

AUTHOR: *Paul Haskell*

INPUTS: *inimage* (message)
inmvec (message)

OUTPUTS: *outimage* (message)

NAME: `Add`

Output the sum of the inputs, as a floating value.

LOCATION: SDF main library

DOMAIN: `SDF` **(DERIVED FROM:** `SDFStar`)

VERSION: 2.7 (11/25/92)

AUTHOR: *J. T. Buck*

INPUTS: *input* (multiple), (float)

OUTPUTS: *output* (float)

NAME: **`ADPCMCoder`**

This galaxy uses the LMS star to implement Adaptive Differential PCM. Both the quantized and unquantized prediction-error signals are available as outputs.

LOCATION: SDF communications palette (galaxy)

DOMAIN: **`SDF`** **(DERIVED FROM: `SDFGalaxy`)**

VERSION: 1.1 (12/17/92)

AUTHOR: *G.S. Walter*

INPUTS: *input* (float)

OUTPUTS: *d(n)* (float)
u(n) (float)

STATES: *stepSize* (FloatState): Step size used for updating filter taps in LMS algorithm.
Default = 1.0E-12
order (IntState): Specifies order of LMS prediction-error filter.
Default = 16
range (FloatState): Positive range of prediction-error coefficients for scaling since quantization is based on interval [-1.0, 1.0].
Default = 800

DESCRIPTION:

This galaxy performs Adaptive Differential PCM using a feed-back-around quantizer structure. The *u(n)* output may be fed into the **`ADPCMDecoder`** galaxy for decoding. A user may choose to utilize the **`ADPCMToBits`** galaxy which will convert the 16-level output *u(n)* into an appropriate stream of bits mimicking a 32 kbps circuit for an 8 kHz-voice input. The bit stream may be converted back to floats using **`ADPCMFromBits`** which could subsequently be decoded using **`ADPCMDecoder.`**

A *d(n)* output has also been made available in this galaxy. This allows the user to see the unquantized prediction error. It is interesting to note that without loss of information between the **`ADPCMCoder`** and the **`ADPCMDecoder,`** the noise found in the decoded signal is identical to the quantization noise injected by the coder. The quantization noise, in this galaxy, is equal to *d(n) - u(n)*.

SEE ALSO: **`ADPCMDecoder ADPCMToBits ADPCMFromBits`**

NAME: **`ADPCMDecoder`**

This galaxy performs decoding of the quantized prediction error signal produced by **`ADPCMCoder`.**

LOCATION: SDF communications palette (galaxy)

DOMAIN: **`SDF` (DERIVED FROM: `SDFGalaxy`)**

VERSION: 1.1 (12/17/92)

AUTHOR: *G.S. Walter*

INPUTS: *input* (float)

OUTPUTS: *output* (float)

STATES: *stepSize* (FloatState): Step size used for updating filter taps in LMS algorithm.
Default = 1.0E-12
order (IntState): Specifies order of LMS prediction-error filter.
Default = 16

DESCRIPTION:

This galaxy performs the ADPCM decoding function. Its *input* is the quantized prediction error produced by the **`ADPCMCoder`** galaxy. For proper decoding to occur, the states *order* and *stepSize* found in both the coder and decoder must be identical. Otherwise mistracking may occur at the decoder.

A **`Limit`** star has been placed between the output of the **`LMS`** and the summer to prevent overflowing the computer's capability of handling float values. In a case where mistracking occurs between the coder and decoder, it is possible for the decoder to become unstable causing the *output* to approach infinity. The **`Limit`** star allows for viewing the instabilities with an **`XGraph`** star while staying within the computer's computative capabilities.

SEE ALSO: **`ADPCMCoder ADPCMToBits ADPCMFromBits`**

NAME: **`ADPCMFromBits`**

This galaxy may be used to convert a bit stream encoded with **`ADPCMToBits`** back to float values. Bits are read in 4 at a time and changed to 1 of 16 float values scaled by *range*.

LOCATION: SDF communications palette (galaxy)

DOMAIN: **`SDF` (DERIVED FROM: `SDFGalaxy`)**

VERSION: 1.1 (12/17/92)

AUTHOR: *G.S. Walter*

INPUTS: *input* (int)

OUTPUTS: *output* (float)

STATES: *range* (FloatState): Positive range for *output* since all quantization is based on interval [-1.0, 1.0].
Default = 800

DESCRIPTION:

This galaxy accepts a bit stream. Four bits are read at a time and converted to 1 of 16 float values prior to being sent to the *output*. The primary application of this galaxy is for use with the **`ADPCMToBits`** galaxy which converts a quantized prediction error signal into a stream of bits. This galaxy, then, would convert that stream of bits back to the quantized prediction error. The output of this galaxy may be fed to the **`ADPCMDecoder`** galaxy which will decode the quantized prediction error.

SEE ALSO: **`ADPCMCoder ADPCMToBits ADPCMDecoder`**

NAME: **`ADPCMToBits`**

This galaxy may be used for converting the quantized prediction error of the **`ADPCMCoder`** galaxy into a bit stream. The quantized prediction error has 16 possible levels so this galaxy produces 4 bits/sample.

LOCATION: SDF communications palette (galaxy)

DOMAIN: **`SDF`** **(DERIVED FROM: `SDFGalaxy`)**

VERSION: 1.1 (12/17/92)

AUTHOR: *G.S. Walter*

INPUTS: *input* (float)

OUTPUTS: *output* (int)

STATES: *range* (FloatState): Positive range of *input* since all quantization is based on interval [-1.0, 1.0].
Default = 800

DESCRIPTION:

This galaxy accepts a signal which has 16 levels. A bit stream will be produced at the output with 4 bits/sample. Its main application is for use with the **`ADPCMCoder`** galaxy in cases where a user would like the quantized prediction error turned into a bit stream. If the input to the **`ADPCMCoder`** is 8 kHz voice, using this galaxy would aid in mimicking a 32 kbps circuit. The bits may be transformed to float values using the **`ADPCMFromBits`** and decoding may be performed using **`ADPCMDecoder.`**

SEE ALSO: **`ADPCMCoder ADPCMFromBits ADPCMDecoder`**

NAME: **ask**

Generate an amplitude-shift keyed (ASK) signal.

LOCATION: SDF communications palette (galaxy)

DOMAIN: **SDF (DERIVED FROM: SDFGalaxy)**

VERSION: 1.1 (12/17/92)

AUTHOR: *E. A. Lee*

OUTPUTS: *out* (float)

STATES: *level* (FloatState): The level of the binary antipodal symbols.
Default = 1.0
excessBW (FloatState): The excess bandwidth of the raised cosine pulse used (1.0=100%).
Default = 1.0

DESCRIPTION:

This galaxy generates a random sequence of bits, encodes them in symbols with value ± *level*, and transmits these symbols using a raised cosine pulse with the given excess bandwidth.

SEE ALSO: qam

NAME: **autocorrelation**

Estimate a power spectrum using the autocorrelation method.

LOCATION: SDF dsp library (galaxy)

DOMAIN: **SDF (DERIVED FROM: SDFGalaxy)**

VERSION: 1.112/17/92 ()

AUTHOR: *E. A. Lee*

INPUTS: *input* (float)

OUTPUTS: *output* (float)

STATES: *numInputs* (IntState): The number of inputs to use for the estimate.
Default = 512
order (IntState): The linear predictor order to use.
Default = 16
log2resolution (IntState): The log base 2 of the number of points in the frequency domain.
Default = 7

DESCRIPTION:

This galaxy estimates the power spectrum of its input using the autocorrelation method (which uses the Levinson Durbin algorithm). If the input process is a Gaussian random process, this method produces an approximate *maximum entropy* spectral estimate. If the autocorrelation of the input process were known exactly, then this methods would produce exact maximum entropy spectra. However, the autocorrelation is estimated from observations of the input using the **`Autocor`** star. Per the classical technique, a biased estimate is used (see references). Based on this autocorrelation estimate, the parameters of an all-pole filter that could have produced the observations given a white noise input are estimated using the Levinson-Durbin algorithm. The transfer function of the all-pole filter is:

$$H(z) = \frac{1}{1 + \sum_{n=1}^{N} d_n z^{-n}}$$

The **`LevDur`** star outputs the d_n parameter estimates, also called the AR parameters. The autocorrelation method power spectrum estimate is simply

$$|H(e^{j\omega})|^2$$

so the rest of the autocorrelation galaxy is devoted to computing this quantity at various values of ω. The number of values of ω to use (128, by default) is specified by the *resolution* parameter of the galaxy. The final output is scaled by an estimate of the power of the input process, extracted from the autocorrelation estimate using the **`Cut`** star. The **`Repeat`** star is needed to maintain consistent sample rates. The **`FloatPad`** star is used to prepend the first AR coefficient, which is always unity, and hence need not be computed.

Prior to computing the spectrum, the signal is modulated with a signal consisting of alternating + and - one, generated by the **`WaveForm`** star. This signal shifts the d.c. component of the signal to the Nyquist frequency, resulting in a spectrum that is centered. Without this modulation, the d.c. component would be the first output of the galaxy, and the last would be the component just below the sampling frequency.

REFERENCES

[1] J. Makhoul, "Linear Prediction: A Tutorial Review", *Proc. IEEE*, Vol. 63, pp. 561-580, Apr. 1975.

[2] S. M. Kay, *Modern Spectral Estimation: Theory & Application*, Prentice-Hall, Englewood Cliffs, NJ, 1988.

[3] S. Haykin, *Modern Filters,* MacMillan Publishing Company, New York, 1989.

SEE ALSO: Burg, LevDur, levinsonDurbin, linearPrediction.

NAME: **`Autocor`**

Estimates an autocorrelation by averaging input samples.

LOCATION: SDF dsp library

DOMAIN: **`SDF`** **(DERIVED FROM: `SDFStar`)**

VERSION: 2.8 (12/8/92)

AUTHOR: *E. A. Lee*

INPUTS: *input* (float)

OUTPUTS: *output* (float)

STATES: *noInputsToAvg* (IntState): Number of input samples to average.
Default = 256

noLags (IntState): Number of autocorrelation lags to output.
Default = 64
unbiased (IntState): If YES, the estimate will be unbiased.
Default = YES

DESCRIPTION:

Estimates a certain number of samples of the autocorrelation of the input by averaging a certain number of input samples. The number of outputs is twice the number of lags requested. This makes the output almost symmetric (discard the last sample to get a perfectly symmetric output).

If the parameter *unbiased* is NO, then the autocorrelation estimate is

$$\hat{r}(k) = \frac{1}{N} \sum_{n=0}^{N-1-k} x(n)x(n+k)$$

for $k = 0, \cdots, p$, where N is the number of inputs to average (*noInputsToAvg*) and p is the number of lags to estimate (*noLags*). This estimate is biased because the outermost lags have fewer than N terms in the summation, and yet the summation is still normalized by N.

If the parameter *unbiased* is YES (the default), then the estimate is

$$\hat{r}(k) = \frac{1}{N-k} \sum_{n=0}^{N-1-k} x(n)x(n+k) .$$

In this case, the estimate is unbiased. However, note that the unbiased estimate does not guarantee a positive definite sequence, so a power spectral estimate based on this autocorrelation estimate may have negative components.

NAME: **`AverageCx`**

Averages some number of input samples or blocks of input samples.

LOCATION: SDF main library

DOMAIN: **`SDF`** **(DERIVED FROM: `SDFStar`)**

VERSION: 1.5 (12/8/92)

AUTHOR: *E. A. Lee*

INPUTS: *input* (complex)

OUTPUTS: *output* (complex)

STATES: *numInputsToAverage* (IntState): The number of input samples or blocks to average.
Default = 8
blockSize (IntState): Input blocks of this size will be averaged to produce an output block.
Default = 1

NAME: **`Average`**

Averages some number of input samples or blocks of input samples.

LOCATION: SDF main library

DOMAIN: **`SDF` (DERIVED FROM: `SDFStar`)**

VERSION: 1.4 (12/8/92)

AUTHOR: *E. A. Lee*

INPUTS: *input* (float)

OUTPUTS: *output* (float)

STATES: *numInputsToAverage* (IntState): The number of input samples or blocks to average.
Default = 8
blockSize (IntState): Input blocks of this size will be averaged to produce an output block.
Default = 1

NAME: **`AWGNchannel`**

Model an additive white Gaussian noise channel with optional linear distortion.

LOCATION: SDF comm library (galaxy)

DOMAIN: **`SDF` (DERIVED FROM: `SDFStar`)**

VERSION: 1.1 (12/17/92)

AUTHOR: *J. Buck*

INPUTS: *in* (float)

OUTPUTS: *out* (float)

STATES: *fwdTaps* (floatArray): Initial forward tap values.
Default = 1
fdbkTaps (floatArray): Initial feedback tap values.
Default = 0
noisePwr (float): The noise power.
Default = 0

DESCRIPTION:

This galaxy filters a signal and then adds Gaussian white noise. The filter is broken into two parts so that the poles and zeros can be conveniently specified separately. This is especially useful to testing equalizers (use an all-pole model) and testing echo cancellers (use an all-zero model). The zeros are modeled first using an FIR filter with taps given by the "fwdTaps" parameter. The poles are then modeled using an FIR filter with taps given by "fdbkTaps" in a feedback loop. Suppose the "fwdTaps" filter has transfer function $H(z)$, and the "fdbkTaps" FIR filter has transfer function $G(z)$. Then the overall transfer function of the channel is

$$C(z) = \frac{H(z)}{1 + z^{-1} G(z)} .$$

SEE ALSO: IIDGaussian, QAM4withDFE

NAME: `Biquad`

A two-pole, two-zero IIR filter (a biquad). The default is a Butterworth filter with a cutoff 0.1 times sample freq. The transfer function is (n0+n1*z1+n2*z2)/(1+d1*z1+d2*z2) where z1 = {z sup -1}, z2 = {z sup -2}.

LOCATION: SDF dsp library

DOMAIN: `SDF` **(DERIVED FROM:** `SDFStar`)

VERSION: 2.8 (11/25/92)

AUTHOR: *J. T. Buck*

INPUTS: *input* (float)

OUTPUTS: *output* (float)

STATES: *d1* (FloatState)
Default = -1.1430
d2 (FloatState)
Default = 0.41280
n0 (FloatState)
Default = 0.067455
n1 (FloatState)
Default = 0.135
n2 (FloatState)
Default = 0.067455
state1 (FloatState): internal state.
Default = 0.0
state2 (FloatState): internal state.
Default = 0.0

DESCRIPTION:

This two-pole, two-zero IIR filter has a transfer function of

$$\frac{n_0 + n_1 z^{-1} + n_2 z^{-2}}{1 + d_1 z^{-1} + d_2 z^{-2}} .$$

The default is a Butterworth filter with a cutoff 0.1 times sample freq.

This star will eventually be replaced by a general IIR star.

NAME: **`bits`**

Produce "0" with probability "probOfZero", else produce "1".

LOCATION: SDF main library (galaxy)

DOMAIN: **`SDF`** **(DERIVED FROM: `Galaxy`)**

VERSION: 1.1 (12/17/92)

AUTHOR: *J. Buck*

OUTPUTS: *out* (int)

STATES: *probOfZero* (FloatState): probability of producing zero
Default = 0.5

DESCRIPTION:

This galaxy produces an independent and identically distributed stream of random values. The output value, an integer, is 0 with probability "probOfZero" and is otherwise 1.

NAME: **`BitsToInt`**

Reads "nBits" bits from the input, packs them into an integer, and outputs it. The first received bit becomes the most significant bit of the output.

LOCATION: SDF main library

DOMAIN: **`SDF`** **(DERIVED FROM: `SDFStar`)**

VERSION: 1.3 (12/8/92)

AUTHOR: *J. T Buck*

INPUTS: *input* (int)

OUTPUTS: *output* (int)

STATES: *nBits* (IntState): number of bits read per execution
Default = 4

NAME: **`BlackHole`**

Discards all inputs.

LOCATION: SDF main library

DOMAIN: **`SDF`** **(DERIVED FROM: `SDFStar`)**

VERSION: 1.5 (11/25/92)

AUTHOR: *J. T. Buck*

INPUTS: *input* (multiple), (ANYTYPE)

DESCRIPTION:

A BlackHole accepts input Particles, but doesn't do anything with them. It is typically used to discard unwanted outputs from other blocks.

NAME: `BlockAllPole`

An all pole filter with coefficients externally supplied.

LOCATION: SDF dsp library

DOMAIN: `SDF` **(DERIVED FROM:** `SDFStar`)

VERSION: 2.10 (12/8/92)

AUTHOR: *E. A. Lee*

INPUTS: *signalIn* (float)
coefs (float): Coefficients of the denominator polynomial

OUTPUTS: *signalOut* (float)

STATES: *blockSize* (IntState): Number of inputs that use each each coefficient set.
Default = 128
order (IntState): Number of new coefficients to read each time.
Default = 16

DESCRIPTION:

This star implements an all pole filter with coefficients that are periodically updated from the outside. The *blockSize* parameter tells how often the updates occur. It is an integer specifying how may input samples should be processed using each set of coefficients. The *order* parameter tells how many coefficients there are. The transfer function of the filter is

$$H(z) = \frac{1}{1 - z^{-1}D(z)}$$

where $D(z)$ is the specified by the externally supplied coefficients. Let

$$D(z) = d_1 + d_2 z^{-1} + \cdots + d_M z^{M-1} .$$

Then d_i is the i^{th} coefficient supplied on the *coefs* input.

No decimation or interpolation is supported.

SEE ALSO: FIR and BlockFIR.

NAME: **`BlockFIR`**

An FIR filter with coefficients externally supplied.

LOCATION: SDF dsp library

DOMAIN: **`SDF`** **(DERIVED FROM: `SDFStar`)**

VERSION: 1.9 (12/8/92)

AUTHOR: *E. A. Lee*

INPUTS: *signalIn* (float)
coefs (float)

OUTPUTS: *signalOut* (float)

STATES: *blockSize* (IntState): Number of inputs that use each each coefficient set.
Default = 128
order (IntState): Number of new coefficients to read each time.
Default = 16
decimation (IntState): Decimation ratio.
Default = 1
decimationPhase (IntState): Downsampler phase.
Default = 0
interpolation (IntState): Interpolation ratio.
Default = 1

DESCRIPTION:

This star implements an FIR filter with coefficients that are periodically updated from the outside. The *blockSize* parameter tells how often the updates occur. It is an integer specifying how may input samples should be processed using each set of coefficients. The *order* parameter tells how many coefficients there are. The same interpolation and decimation of the FIR star is supported.

Unfortunately, it proves not too convenient to derive this star from FIR because of the changes in the way the inputs and outputs are handled. Hence, much of the code here is quite similar to that in the FIR star.

SEE ALSO: FIR.

NAME: **`BlockLattice`**

A Block Forward Lattice filter. It is identical to the Lattice star except that the reflection coefficients are updated each time the star fires by reading the "coefs" input. The "order" parameter indicates how many coefficient should be read. The "blockSize" parameter specifies how many signalIn samples should be processed for each set of coefficients.

LOCATION: SDF dsp library

DOMAIN: `SDF` **(DERIVED FROM:** `SDFStar`)

VERSION: 1.7 (12/8/92)

AUTHOR: *Alan Kamas and Edward Lee*

INPUTS: *signalIn* (float)
coefs (float)

OUTPUTS: *signalOut* (float)

STATES: *blockSize* (IntState): Number of inputs that use each each coefficient set.
Default = 128
order (IntState): Number of new coefficients to read each time.
Default = 16

SEE ALSO: Lattice, RLattice, BlockRLattice, FIR, FIRCx and Biquad.

NAME: `BlockRLattice`

A block Recursive (Backward) (IIR) Lattice filter. It is identical to the RLattice star, except that the reflection coefficients are updated each time the star fires by reading the "coefs" input. The "order" parameter indicates how many coefficient should be read. The "blockSize" parameter specifies how many signalIn samples should be processed for each set of coefficients.

LOCATION: SDF dsp library

DOMAIN: `SDF` **(DERIVED FROM:** `SDFStar`)

VERSION: 1.4 (12/8/92)

AUTHOR: *Alan Kamas and Edward Lee*

INPUTS: *signalIn* (float)
coefs (float)

OUTPUTS: *signalOut* (float)

STATES: *blockSize* (IntState): Number of inputs that use each each coefficient set.
Default = 128
order (IntState): Number of new coefficients to read each time.
Default = 16

SEE ALSO: IIR, Lattice, RLattice and BlockLattice.

NAME: **`Burg`**

Burg's algorithm. The lp output receives the linear prediction coefficients and the refl output receives reflection coefficients. The errPower output gets the power of the prediction error at each stage.

LOCATION: SDF dsp library

DOMAIN: **`SDF`** **(DERIVED FROM: `SDFStar`)**

VERSION: 1.12 (12/8/92)

AUTHOR: *E. A. Lee and J. T. Buck*

INPUTS: *input* (float): Input random process.

OUTPUTS: *lp* (float): AR coefficients output.
refl (float): Lattice predictor coefficients output.
errPower (float): Prediction error power.

STATES: *order* (IntState): The number of reflection coefficients to generate.
Default = 8
numInputs (IntState): The number of inputs used to estimate the model.
Default = 64

DESCRIPTION:

This star uses Burg's algorithm to estimate the reflection coefficients and AR parameters of an input random process. The number of inputs looked at is given by the *numInputs* parameter and the order of the AR model is given by the *order* parameter. The order specifies how many outputs appear on the *lp* and *refl* output portholes. These outputs are, respectively, the autoregressive (AR) parameters (also called the linear predictor parameters), and the reflection coefficients.

Note that the definition of reflection coefficients is not quite universal in the literature. The reflection coefficients in references [2] and [3] are the negative of the ones generated by this star, which correspond to the definition in most other texts, and to the definition of partial-correlation (PARCOR) coefficients in the statistics literature.

The *errPower* output is the power of the prediction error as a function of the model order. There are *order*+1 output samples, where the first corresponds to the prediction error of a zero-th order predictor. This is simply an estimate of the input signal power.

REFERENCES

[1] J. Makhoul, "Linear Prediction: A Tutorial Review", *Proc. IEEE*, Vol. 63, pp. 561-580, Apr. 1975.

[2] S. M. Kay, *Modern Spectral Estimation: Theory & Application*, Prentice-Hall, Englewood Cliffs, NJ, 1988.

[3] S. Haykin, *Modern Filters*, MacMillan Publishing Company, New York, 1989.

SEE ALSO: LevDur, linearPrediction and powerSpectrum.

NAME: **`burg`**

Estimate a power spectrum using Burg's method.

LOCATION: SDF dsp library (galaxy)

VERSION: 1.112/17/92 ()

AUTHOR: *E. A. Lee*

INPUTS: *input* (float)

OUTPUTS: *output* (float)

STATES: *numInputs* (IntState): The number of inputs to use for the estimate.
Default = 512
order (IntState): The linear predictor order to use.
Default = 16
log2resolution (IntState): The log base 2 of the number of points in the f requency domain.
Default = 7

DESCRIPTION:

This galaxy estimates the power spectrum of its input using Burg's method. If the input process is a Gaussian random process, this method produces an approximate *maximum entropy* spectral estimate. The parameters of an all-pole filter that could have produced the observations given a white noise input are estimated. The transfer function of the all-pole filter is:

$$H(z) = \frac{1}{1 + \sum_{n=1}^{N} d_n z^{-n}}$$

The **`Burg`** star outputs the d_n parameter estimates, also called the AR parameters. The spectrum estimate is simply

$$|H(e^{j\omega})|^2$$

so the rest of the burg galaxy is devoted to computing this quantity at various values of ω. The number of values of ω to use (128, by default) is specified by the *resolution* parameter of the galaxy. The final output is scaled by an estimate of the power of the input process, extracted from the burg estimate using the **`Cut`** star. The **`Repeat`** star is needed to maintain consistent sample rates. The **`FloatPad`** star is used to prepend the first AR coefficient, which is always unity, and hence need not be computed.

Prior to computing the spectrum, the signal is modulated with a signal consisting of alternating + and - one, generated by the **`WaveForm`** star. This signal shifts the d.c. component of the signal to the Nyquist frequency, resulting in a spectrum that is centered. Without this modulation, the d.c. component would be the first output of the galaxy, and the last would be the component just below the sampling frequency.

REFERENCES

[1] J. Makhoul, "Linear Prediction: A Tutorial Review", *Proc. IEEE*, Vol. 63, pp. 561-580, Apr. 1975.

[2] S. M. Kay, *Modern Spectral Estimation: Theory & Application*, Prentice-Hall, Englewood Cliffs, NJ, 1988.

[3] S. Haykin, *Modern Filters*, MacMillan Publishing Company, New York, 1989.

SEE ALSO: LevDur, levinsonDurbin, linearPrediction, powerSpectrum.

NAME: `BusToNum`

Takes in a bus (with 'bits' no of lines) and outputs a signed integer corresponding to the value of this binary number.

DOMAIN: `SDF` **(DERIVED FROM:** `SDFStar`)

VERSION: 1.10 (11/25/92)

AUTHOR: *Asawaree Kalavade*

INPUTS: *input* (multiple), (int)

OUTPUTS: *output* (int)

STATES: *bits* (IntState): number of bits in input
Default = 7
previous (IntState): previous value of output
Default = 0

NAME: `Commutator`

Takes N input streams (where N is the number of inputs) and synchronously combines them into one output stream. It consumes B input particles from each input (where B is the blockSize), and produces N*B particles on the output. The first B particles on the output come from the first input, the next B particles from the next input, etc.

LOCATION: SDF main library

DOMAIN: `SDF` **(DERIVED FROM:** `SDFStar`)

VERSION: 2.8 (12/8/92)

AUTHOR: *J. T. Buck and E. A. Lee*

INPUTS: *input* (multiple), (ANYTYPE)

OUTPUTS: *output* (ANYTYPE)

STATES: *blockSize* (IntState): Number of particles in a block.
Default = 1

NAME: **`conj`**

Compute the complex conjugate of the input.

LOCATION: SDF nonlinear palette (galaxy)

VERSION: 1.112/17/92 ()

AUTHOR: *J. Buck*

INPUTS: *inc* (complex)

OUTPUTS: *out* (complex)

DESCRIPTION:

This galaxy converts a complex input to real and imaginary parts, negates the imaginary part, and then converts back.

NAME: **`ConstCx`**

Outputs a constant complex signal with real part "real" (default 0.0) and imaginary part "imag" (default 0.0).

LOCATION: SDF main library

DOMAIN: **`SDF`** **(DERIVED FROM: `SDFStar`)**

VERSION: 1.5 (11/25/92)

AUTHOR: *J. T. Buck*

OUTPUTS: *output* (complex)

STATES: *real* (FloatState): Real part of DC value.
Default = 0.0
imag (FloatState): Imaginary part of DC value.
Default = 0.0

DESCRIPTION:

This star produces a complex DC output (default zero) Even though we have ComplexState, the real and imaginary parts are separate states. (Should this change?)

NAME: **`Const`**

Output a constant signal with value level (default 0.0).

LOCATION: SDF main library

DOMAIN: `SDF` **(DERIVED FROM:** `SDFStar`)

VERSION: 1.5 (11/25/92)

AUTHOR: *J. T. Buck*

OUTPUTS: *output* (float)

STATES: *level* (FloatState): The constant value.
Default = 0.0

NAME: `Convolve`

Convolve two causal finite sequences. Set truncationDepth larger than the number of output samples of interest.

LOCATION: SDF dsp library

DOMAIN: `SDF` **(DERIVED FROM:** `SDFStar`)

VERSION: 1.5 (12/8/92)

AUTHOR: *E. A. Lee and K. White*

INPUTS: *inA* (float)
inB (float)

OUTPUTS: *out* (float)

STATES: *truncationDepth* (IntState): Maximum number of terms in convolution sum.
Default = 256
iterationCount (IntState): Count current iteration.
Default = 0

DESCRIPTION:

This star convolves two causal finite input sequences. In the current implementation, you should set the truncation depth larger than the number of output samples of interest. If it is smaller, you will get unexpected results after truncationDepth samples.

SEE ALSO: FIR, FIRCx, blockFIR and firDemo.

NAME: `Cos`

This star computes the cosine of its input, in radians.

LOCATION: SDF main library

DOMAIN: `SDF` **(DERIVED FROM:** `SDFStar`)

VERSION: 1.6 (11/25/92)

AUTHOR: *J. T. Buck*

INPUTS: *input* (float)

OUTPUTS: *output* (float)

NAME: `Cut`

On each execution, this star reads a block of "nread" samples (default 128) and writes "nwrite" of these samples (default 64), skipping the first "offset" samples (default 0). It is an error if nwrite + offset > nread. If nwrite > nread, then the output consists of overlapping windows, and hence "offset" must be negative.

LOCATION: SDF main library

DOMAIN: `SDF` **(DERIVED FROM:** `SDFStar`)

VERSION: 2.6 (12/8/92)

AUTHOR: *J. T. Buck*

INPUTS: *input* (ANYTYPE)

OUTPUTS: *output* (ANYTYPE)

STATES: *nread* (IntState): Number of particles read.
Default = 128
nwrite (IntState): Number of particles written.
Default = 64
offset (IntState): Position of output block relative to input block.
Default = 0

DESCRIPTION:

This star reads a block of particles of any type, and writes a block of particles that somehow overlaps the input block. The number of input particles consumed is given by *nread*, and the number of output particles produced is given by *nwrite*. The *offset* parameter (default 0) specifies where the output block should start, relative to the beginning of the input block. To avoid trying to read samples that have not yet been consumed, it is required that $nwrite + offset \le nread$. Hence, if $nwrite > nread$, *offset* must be negative, and the output consists of overlapping blocks input particles.

NAME: `CutVarOffset`

This star has the same functionality as the Cut star except now the offset parameter is determined at run time through a control input.

LOCATION: SDF control palette

DOMAIN: `SDF` **(DERIVED FROM:** `SDFCut`)

VERSION: 1.4 (12/17/92)

AUTHOR: *GSWalter, E. A. Lee*

INPUTS: *offsetCntrl* (int)

NAME: `CxToRect`

Convert complex data to real and imaginary parts.

LOCATION: SDF main library

DOMAIN: **`SDF` (DERIVED FROM: `SDFStar`)**

VERSION: 1.6 (11/25/92)

AUTHOR: *E. A. Lee*

INPUTS: *input* (complex)

OUTPUTS: *real* (float)
imag (float)

NAME: `DB`

Converts input to dB. Zero and negative values are converted to "min" (default -100).

LOCATION: SDF main library

DOMAIN: **`SDF` (DERIVED FROM: `SDFStar`)**

VERSION: 1.10 (12/8/92)

AUTHOR: *J. T. Buck*

INPUTS: *input* (float)

OUTPUTS: *output* (float)

STATES: *min* (FloatState): Minimum output value.
Default = -100
inputIsPower (IntState): TRUE if input is a power measurement, FALSE if it's an amplitude measurement.
Default = FALSE

DESCRIPTION:

For inputs that are greater than zero, the output either $N \log_{10}(input)$ or *min*, whichever is larger, where $N = 10$ if *inputIsPower* is TRUE, and $N = 20$ otherwise. The default is $N = 20$. For inputs that are zero or negative, the output is *min*.

NAME: `DCTImageCodeInv`

This star reads two coded DCTImages (one high priority and one low-priority), inverts the run-length encoding, and outputs the resulting DCTImage. Protection is built in to avoid crashing even if some of the coded input data is affected by loss.

NOTE!! This star is different from the SDFRunLenInv star. This one works on DCTImages, not GrayImages.

LOCATION: SDF image palette

DOMAIN: `SDF` **(DERIVED FROM:** `SDFStar`)

VERSION: 1.8 (11/25/92)

AUTHOR: *Paul Haskell*

INPUTS: *hiport* (message)
loport (message)

OUTPUTS: *output* (message)

STATES: *HiPri* (IntState): Number of DCT coeffs per block to 'highport' input.
Default = 1

SEE ALSO: DCTImageCode.

NAME: `DCTImageCode`

This star takes a DCTImage input, inserts "StartOfBlock" markers, run-length encodes it using 'StartOfRun' markers, and outputs the modified DCTImage.

For the run-length encoding, all values with absolute value less than "Thresh" are set to 0.0, to help improve compression.

Runlengths are coded with a start symbol of "StartOfRun" (defined below) and then an (integer) run-length.

"HiPri" DCT coefficients per block are sent to "hiport", the high-priority output. The remainder of the coefficients are sent to "loport", the low-priority output.

NOTE!! This differs from the SDFRunLen star in that this block processes DCTImages, not GrayImages.

LOCATION: SDF image palette

DOMAIN: `SDF` **(DERIVED FROM:** `SDFStar`)

VERSION: 1.11 (12/8/92)

AUTHOR: *Paul Haskell*

INPUTS: *inport* (message)

OUTPUTS: *hiport* (message)
loport (message)
compr (float)

STATES: *Thresh* (FloatState): Threshold for run-length coding.
Default = 0
HiPri (IntState): Number of DCT coefficients per block sent to 'hiport'.
Default = 1

NAME: **`DCTImageInv`**

This star takes a DCTImage input and does inverse discrete cosine transform (DCT) coding and outputs a GrayImage.

LOCATION: SDF image library

DOMAIN: **`SDF`** **(DERIVED FROM: `SDFStar`)**

VERSION: 1.15 (12/8/92)

AUTHOR: *Sun-Inn Shih, Paul Haskell*

INPUTS: *input* (message)

OUTPUTS: *output* (message)

DESCRIPTION:

NAME: **`DCTImage`**

This star takes GrayImage input and does discrete cosine transform (DCT) and outputs a DCTImage.

LOCATION: SDF image library

DOMAIN: **`SDF`** **(DERIVED FROM: `SDFStar`)**

VERSION: 1.14 (12/8/92)

AUTHOR: *Sun-Inn Shih, Paul Haskell*

INPUTS: *input* (message)

OUTPUTS: *output* (message)

STATES: *BlockSize* (IntState): Block size of the DCT transform.
Default = 8

DESCRIPTION:

NAME: **`DeMux`**

Demultiplexes one input onto any number of output streams. The star consumes B particles from the input, where B is the blockSize. These B particles are copied to exactly one output, determined by the "control" input. The other outputs get a zero of the appropriate type.

Integers from 0 through N-1 are accepted at the "control" input, where N is the number of outputs. If the control input is outside this range, all outputs get zero.

LOCATION: SDF main library

DOMAIN: `SDF` **(DERIVED FROM:** `SDFStar`)

VERSION: 1.9 (12/8/92)

AUTHOR: *E. A. Lee*

INPUTS: *input* (anytype)
control (int)

OUTPUTS: *output* (multiple), (anytype)

STATES: *blockSize* (IntState): Number of particles in a block.
Default = 1

DESCRIPTION:

NAME: `DiffImage`

Accept two black-and-white images from input GrayImages, take their absolute difference, scale the absolute difference, and pass the result to the output.

LOCATION: SDF image library

DOMAIN: `SDF` **(DERIVED FROM:** `SDFStar`)

VERSION: 1.8 (11/25/92)

AUTHOR: *Paul Haskell*

INPUTS: *input1* (message)
input2 (message)

OUTPUTS: *outData* (message)

STATES: *Scale* (FloatState): Amount by which to scale the image difference.
Default = 1.0

DESCRIPTION:

NAME: `DisplayImage`

Accept a black-and-white input GrayImage and generate output in PGM format. Send the output to a user-specified command (by default, "xv" is used). Of course, the specified program must be in your PATH.

The user can set the root filename of the displayed image (which will probably be printed in the image display window titlebar) and can choose whether or not the image file is saved or deleted. The image frame number is appended to the root filename to form the complete filename of the displayed image.

LOCATION: SDF image library

DOMAIN: **`SDF` (DERIVED FROM: `SDFStar`)**

VERSION: 1.13 (12/8/92)

AUTHOR: *J. Buck, Paul Haskell*

INPUTS: *inData* (message)

STATES: *command* (StringState): Program to run on PGM data
Default = xv
imageName (StringState): If non-null, name for PGM file
Default =
saveImage (StringState): If 'y' or 'Y', then save the file
Default = n

DESCRIPTION:

NAME: **`DisplayRGB`**

Accept three ColorImages (Red, Green, and Blue) from three input GrayImages and generate a PPM format color image file. Send the filename to a user-specified command (by default, "xv" is used).

The user can set the root filename of the displayed image (which will probably be printed in the image display window titlebar) and can choose whether or not the image file is saved or deleted. The frameId of the received image is appended to the root filename to produce the full filename of the displayed image.

LOCATION: SDF image library

DOMAIN: **`SDF` (DERIVED FROM: `SDFStar`)**

VERSION: 1.14 (11/25/92)

AUTHOR: *Sun-Inn Shih*

INPUTS: *rinput* (message)
ginput (message)
binput (message)

STATES: *command* (StringState): Program to run on PGM data
Default = xv
imageName (StringState): If non-null, name for PGM file
Default =
saveColor (StringState): If 'y' or 'Y', then save the file
Default = n

DESCRIPTION:

NAME: **`DisplayVideo`**

Accept a stream of black-and-white images from input GrayImages, save the images to files, and display the resulting files as a moving video sequence. This star requires that programs from the "Utah Raster Toolkit" be in your "path" variable. Although this toolkit is not included with Ptolemy it is available for free. See this star’s long description (with the "look-inside" or "manual" commands in the Ptolemy menu) for info on how to get the toolkit.

The user can set the root filename of the displayed images (which probably will be printed in the display window titlebar) with the ’ImageName’ state. If no filename is set, a default will be chosen.

The ’Save’ state can be set to "YES" or "NO" to choose whether the created image files should be saved or deleted. Each image’s frame number is appended to the root filename to form the image’s complete filename.

LOCATION: SDF image palette

DOMAIN: **`SDF`** **(DERIVED FROM: `SDFStar`)**

VERSION: 1.11 (12/8/92)

AUTHOR: *Paul Haskell*

INPUTS: *inData* (message)

STATES: *ImageName* (StringState): If non-null, name for RLE file.
Default =
Save (IntState): If true (YES), then save the file
Default = NO

DESCRIPTION:

At the end of a simulation this star pops up an X window and loads in a sequence of video frames for display. Pressing the left or right mouse buttons inside the window plays the video sequence backwards or forwards. The middle mouse button allows single-stepping through frames. If you hold down the shift key while pressing the left mouse button, you loop through the sequence. The shift key and middle mouse button lets you alter the frame rate of the displayed video. The shift key and right mouse button loops through the video sequence alternately forwards and backwards. To end a loop playback, press any mouse button in the video window. To close the window type "q" inside.

This star uses programs from the *Utah Raster Toolkit* to display moving video in an X window. The Utah Raster Toolkit is a collection of software tools from the University of Utah. These programs are available free via anonymous ftp. To get the software:

unix> ftp cs.utah.edu

ftp name> anonymous
ftp passwd> YOUR EMAIL ADDRESS
ftp> binary
ftp> cd pub

ftp> get urt-3.0.tar.Z
ftp> quit

unix> uncompress urt-3.0.tar.Z
unix> tar xvf urt-3.0.tar

Then, change directories to the new urt_3.0 directory and build the software. To use the software, put the name of the directory with the URT executable files into the definition of the *path* variable inside the .cshrc file in your home directory.

These instructions are appropriate as of December 1992 but may change in the future.

NAME: **`Distributor`**

Takes one input stream and synchronously splits it into N output streams, where N is the number of outputs. It consumes N*B input particles, where B = blockSize, and sends the first B particles to the first output, the next B particles to the next output, etc.

LOCATION: SDF main library

DOMAIN: **`SDF`** **(DERIVED FROM: `SDFStar`)**

VERSION: 2.7 (12/8/92)

AUTHOR: *J. T. Buck and E. A. Lee*

INPUTS: *input* (ANYTYPE)

OUTPUTS: *output* (multiple), (ANYTYPE)

STATES: *blockSize* (IntState): Number of particles in a block.
Default = 1

NAME: **`DownSample`**

A decimator by "factor" (default 2). The "phase" tells which sample to output. If phase = 0, the most recent sample is the output, while if phase = factor-1 the oldest sample is the output. Phase = 0 is the default. Note that "phase" has the opposite sense of the phase parameter in the UpSample star, but the same sense as the phase parameter in the FIR star.

LOCATION: SDF main library

DOMAIN: **`SDF`** **(DERIVED FROM: `SDFStar`)**

VERSION: 2.8 (12/8/92)

AUTHOR: *J. T. Buck*

INPUTS: *input* (ANYTYPE)

OUTPUTS: *output* (ANYTYPE)

STATES: *factor* (IntState): Downsample factor.
Default = 2
phase (IntState): Downsample phase.
Default = 0

DESCRIPTION:

NAME: **`DPCMImageInv`**

If the "past" input is not a GrayImage or has size 0, pass the "diff" directly to the "output". Otherwise, add the "past" to the "diff" (with leak factor "alpha") and send the result to "output".

LOCATION: SDF image library

DOMAIN: **`SDF`** **(DERIVED FROM: `SDFStar`)**

VERSION: 1.11 (12/8/92)

AUTHOR: *Paul Haskell*

INPUTS: *diff* (message)
past (message)

OUTPUTS: *output* (message)

STATES: *alpha* (FloatState): Leak value to aid error recovery.
Default = 1.0

DESCRIPTION:

SEE ALSO: DPCMImage.

NAME: **`DPCMImage`**

If the "past" input is not a GrayImage or has size 0, pass the "input" directly to the "output". Otherwise, subtract the "past" from the "input" (with leak factor "alpha") and send the result to "output".

LOCATION: SDF image library

DOMAIN: **`SDF`** **(DERIVED FROM: `SDFStar`)**

VERSION: 1.11 (12/8/92)

AUTHOR: *Paul Haskell*

INPUTS: *input* (message)
past (message)

OUTPUTS: *output* (message)

STATES: *alpha* (FloatState): Leak value for error-recovery.
Default = 1.0

DESCRIPTION:

NAME: **DTFT**

Discrete-time Fourier transform.

LOCATION: SDF dsp library

DOMAIN: **SDF (DERIVED FROM: SDFStar)**

VERSION: 1.8 (12/8/92)

AUTHOR: *E. A. Lee*

INPUTS: *signal* (complex): Signal to be transformed.
omega (float): Frequency values at which to sample the transform.

OUTPUTS: *dtft* (complex): The samples of the transform.

STATES: *length* (IntState): The length of the input signal.
Default = 8
numberOfSamples (IntState): The number of samples of the transform to output.
Default = 128
timeBetweenSamples (FloatState): The time between input samples (T).
Default = 1.0

DESCRIPTION:

This star computes arbitrary samples of the discrete-time Fourier transform of a finite length sequence. The *signal* input is the signal to be transformed. The number of input samples consumed is given by *length*. Let these be written $a(0), \cdots, a(L-1)$, where L is the *length*. Then the output is

$$A(j\omega) = \sum_{k=0}^{L-1} a(k) e^{-j\omega kT}$$

where T is the time between samples (*timeBetweenSamples*). The number of samples produced at the output is determined by the *numberOfSamples* parameter. The values of ω at which samples are taken are provided by the *omega* input. Hence, any frequency range or ranges can be examined at any desired resolution, and samples need not be taken at uniform intervals. Note that $\omega = 2\pi/T$ is the sampling frequency.

SEE ALSO: FFTCx.

NAME: **expgen**

Generate a complex exponential with the given frequency (relative to the sample_rate).

LOCATION: SDF sources palette (galaxy)

DOMAIN: `SDF` **(DERIVED FROM:** `SDFGalaxy`)

VERSION: 1.1 (12/17/92)

AUTHOR: *E. A. Lee*

OUTPUTS: *out* (complex)

STATES: *sample_rate* (FloatState): The sample rate, for reference to the frequency.
Default = 2*PI
frequency (FloatState): The frequency, relative to the sample rate.
Default = PI/50

DESCRIPTION:

The sample_rate parameter is used only locally to determine how to construct the complex output waveform. The frequency is given relative to the sample rate.

SEE ALSO: expjx

NAME: `expjx`

Compute the complex exponential function of the input.

LOCATION: SDF nonlinear palette (galaxy)

VERSION: 1.112/17/92 ()

AUTHOR: *J. Buck*

INPUTS: *in* (float)

OUTPUTS: *out* (float)

DESCRIPTION:

This galaxy computes the sine and the cosine of the input and combines them into a single complex number.

SEE ALSO: Exp

NAME: **Exp**

Outputs the exponential function of the input.

LOCATION: SDF main library

DOMAIN: `SDF` **(DERIVED FROM:** `SDFStar`)

VERSION: 1.6 (11/25/92)

AUTHOR: *J. T. Buck*

INPUTS: *input* (float)

OUTPUTS: *output* (float)

DESCRIPTION:

Bugs: Overflow is not handled well.

NAME: `FFTCx`

Complex Fast Fourier transform. Parameter "order" (default 8) is the log, base 2, of the transform size. Parameter "size" (default 256) is the number of samples read (<= 2^order). Parameter "direction" (default 1) is 1 for forward, -1 for inverse FFT.

LOCATION: SDF dsp library

DOMAIN: `SDF` **(DERIVED FROM:** `SDFStar`)

VERSION: 1.14 (12/8/92)

AUTHOR: *J. T. Buck*

INPUTS: *input* (complex)

OUTPUTS: *output* (complex)

STATES: *order* (IntState): Log base 2 of the transform size.
Default = 8
size (IntState): Number of input samples to read.
Default = 256
direction (IntState): = 1 for forward, = -1 for inverse.
Default = 1

DESCRIPTION:

A number of input samples given by the parameter *size* will be consumed at the input, zero-padded if necessary to make 2^{order} samples, and transformed using a fast Fourier transform algorithm. If *direction* is 1, then the forward Fourier transform is computed. If *direction* is -1, then the inverse Fourier transform is computed.

Note a single firing of this star consumes *size* inputs and produces 2^{order} outputs. This must be taken into account when determining for how many iterations to run a universe. For example, to compute just one FFT, only one iteration should be run.

Bugs: the routine currently used (from Gabriel) recomputes trig functions for each term, instead of using a table. Instead, FFTCx::setup() should compute a table of appropriate size to save time. This has no effect, obviously, if only one transform is performed.

NAME: `FIRCx`

A complex Finite Impulse Response (FIR) filter. Coefficients are in the "taps" state variable. Default coefficients give an 8th order, linear phase, lowpass filter. To read coefficients from a file, use the syntax: "<fileName". Real and imaginary parts should be paired with parentheses, e.g. (1.0, 0.0).

LOCATION: SDF dsp library

DOMAIN: **SDF (DERIVED FROM: `SDFStar`)**

VERSION: 1.9 (12/8/92)

AUTHOR: *E. A. Lee*

INPUTS: *signalIn* (complex)

OUTPUTS: *signalOut* (complex)

STATES: *taps* (ComplexArrayState): Filter tap values.
Default = (-.040609,0.0) (-.001628,0.0) (.17853,0.0) (.37665,0.0) (.37665,0.0) (.17853,0.0) (-.001628,0.0) (-.040609,0.0)
decimation (IntState): Decimation ratio.
Default = 1
decimationPhase (IntState): Downsampler phase.
Default = 0
interpolation (IntState): Interpolation ratio.
Default = 1

DESCRIPTION:

ComplexFIR implements a finite-impulse response filter with multirate capability. The default coefficients correspond to a real eighth order, equiripple, linear-phase, lowpass filter. The 3dB cutoff frequency at about 1/3 of the Nyquist frequency. To load filter coefficients from a file, simply replace the default coefficients with the string "<filename". The real and imaginary parts should be enclosed in parenthesis, as follows: (0.1,0.3).

It is advisable to use an absolute path name as part of the file name, especially if you are using the graphical interface. This will allow the filter to work as expected regardless of the directory in which the ptolemy process actually runs. It is best to use tildes in the filename.

When the *decimation* (*interpolation*) state is different from unity, the filter behaves exactly as it were followed (preceded) by a DownSample (UpSample) star. However, the implementation is much more efficient than it would be using UpSample and DownSample stars; a polyphase structure is used internally, avoiding unnecessary use of memory and unnecessary multiplication by zero. Arbitrary sample-rate conversions by rational factors can be accomplished this way.

To design a filter for a multirate system, simply assume the sample rate is the product of the interpolation parameter and the input sample rate, or equivalently, the product of the decimation parameter and the output sample rate. In particular, considerable care must be taken to avoid aliasing. Specifically, if the input sample rate is f, then the filter stopband should begin before f/2. If the interpolation ratio is i, then f/2 is a fraction 1/2i of the sample rate at which you must design your filter.

The *decimationPhase* parameter is somewhat subtle. It is exactly equivalent the phase parameter of the DownSample star. Its interpretation is as follows; when decimating, samples are conceptually discarded (although a polyphase structure does not actually compute the discarded samples). If you are decimating by a factor of three, then you will select one of every three outputs, with three possible phases. When decimationPhase is zero (the default), the latest (most recent) samples are the ones selected. The decimationPhase must be strictly less than the decimation ratio.

For more information about polyphase filters, see F. J. Harris, "Multirate FIR Filters for Interpolating and Desampling", in *Handbook of Digital Signal Processing*, Academic Press, 1987.

SEE ALSO: FIR, Biquad, UpSample, DownSample and analytic.

NAME: **`FIR`**

A Finite Impulse Response (FIR) filter. Coefficients are in the "taps" state variable. Default coefficients give an 8th order, linear phase lowpass filter. To read coefficients from a file, replace the default coefficients with "<fileName".

LOCATION: SDF dsp library

DOMAIN: **`SDF`** **(DERIVED FROM: `SDFStar`)**

VERSION: 1.9 (12/8/92)

AUTHOR: *E. A. Lee*

INPUTS: *signalIn* (float)

OUTPUTS: *signalOut* (float)

STATES: *taps* (FloatArrayState): Filter tap values.
Default = -.040609 -.001628 .17853 .37665 .37665 .17853 -.001628 -.040609
decimation (IntState): Decimation ratio.
Default = 1
decimationPhase (IntState): Downsampler phase.
Default = 0
interpolation (IntState): Interpolation ratio.
Default = 1

DESCRIPTION:

This star implements a finite-impulse response filter with multirate capability. The default coefficients correspond to an eighth order, equiripple, linear-phase, lowpass filter. The 3dB cutoff frequency at about 1/3 of the Nyquist frequency. To load filter coefficients from a file, simply replace the default coefficients with the string "<filename".

It is advisable to use an absolute path name as part of the file name, especially if you are using the graphical interface. This will allow the FIR filter to work as expected regardless of the directory in which the ptolemy process actually runs. It is best to use tilde's in the filename to reference them to user's home directory. This way, future filesystem reorganizations will have minimal effect.

When the *decimation* (*interpolation*) state is different from unity, the filter behaves exactly as it were followed (preceded) by a DownSample (UpSample) star. However, the implementation is much more efficient than it would be using UpSample and DownSample stars; a polyphase structure is used internally, avoiding unnecessary use of memory and unnecessary multiplication by zero. Arbitrary sample-rate conversions by rational factors can be accomplished this way.

To design a filter for a multirate system, simply assume the sample rate is the product of the interpolation parameter and the input sample rate, or equivalently, the product of the decimation parameter and the output sample rate. In particular, considerable care must be taken to avoid aliasing. Specifically, if the input sample rate is f, then the filter stopband should begin before f/2. If the interpolation ratio is i, then f/2 is a fraction 1/2i of the sample rate at which you must design your filter.

The *decimationPhase* parameter is somewhat subtle. It is exactly equivalent the phase parameter of the DownSample star. Its interpretation is as follows; when decimating, samples are conceptually discarded (although a polyphase structure does not actually compute the discarded samples). If you are decimating by a factor of three, then you will select one of every three outputs, with three possible phases. When decimationPhase is zero (the default), the latest (most recent) samples are the ones selected. The decimationPhase must be strictly less than the decimation ratio.

For more information about polyphase filters, see F. J. Harris, "Multirate FIR Filters for Interpolating and Desampling", in *Handbook of Digital Signal Processing*, Academic Press, 1987.

SEE ALSO: FIRCx, Biquad, UpSample, DownSample, firDemo, interp and multirate.

NAME: `Floor`

Outputs the greatest integer <= input.

LOCATION: SDF main library

DOMAIN: `SDF` **(DERIVED FROM:** `SDFStar`)

VERSION: 1.4 (11/25/92)

AUTHOR: *J. T. Buck*

INPUTS: *input* (float)

OUTPUTS: *output* (int)

NAME: `Fork`

Copies input particles to each output.

LOCATION: SDF main library

DOMAIN: `SDF` **(DERIVED FROM:** `SDFStar`)

VERSION: 2.4 (11/25/92)

AUTHOR: *D. G. Messerschmitt*

INPUTS: *input* (ANYTYPE)

OUTPUTS: *output* (multiple), (ANYTYPE)

DESCRIPTION:

This star is generally used to connect a single output port to multiple input ports. It will be automatically inserted when multiple inputs are connected to the same output using the graphical interface, or when the "nodeconnect" command is used in the interpreter. However, there are times when automatically inserted Fork stars are not desirable. For instance, when there is a delay on one of the arcs, then the Fork must be inserted by the user explicitly to avoid ambiguity about the location of the delay. Also, when multi-portHoles are used, auto-forking can cause problems. In this situation, one may get, for example, two outputs and several inputs on the same net. There is currently no way to automatically decipher what the user intends. Hence, the Fork star should be inserted explicitly.

NAME: `freqPhase`

Impose frequency offset and/or phase jitter on a signal

LOCATION: SDF communications palette (galaxy)

DOMAIN: `SDF` **(DERIVED FROM:** `SDFGalaxy`)

VERSION: 1.112/17/92 ()

AUTHOR: *E. A. Lee*

INPUTS: *signalIn* (float)

OUTPUTS: *signalOut* (float)

STATES: *sample_rate* (FloatState): The sample rate, for reference to the frequency.
Default = 2*PI
frequencyOffset (FloatState): The frequency offset, relative to the sample rate.
Default = 0.0
phaseJitterFrequency (FloatState): The phase jitter frequency, relative to the sample rate.
Default = 0.0
phaseJitterAmplitude_Deg (FloatState): The phase jitter amplitude, in degrees.
Default = 0.0

DESCRIPTION:

The sample_rate parameter is used only locally as a reference. If the default "2*PI"is used, then other frequencies should be specified in radians. Otherwise, those frequencies should be specified relative to the "sample_rate".

SEE ALSO: phaseShift freqPhaseOffset

NAME: `GainCx`

Amplifier: output is input times "gain" (default 1.0). Input, output, and gain factor are all complex numbers.

LOCATION: SDF main library

DOMAIN: `SDF` **(DERIVED FROM:** `SDFStar`)

VERSION: 1.3 (11/25/92)

AUTHOR: *J. T. Buck*

INPUTS: *input* (complex)

OUTPUTS: *output* (complex)

STATES: *gain* (ComplexState): Gain of the star.
Default = (1,0)

NAME: **`Gain`**

Amplifier: output is input times "gain" (default 1.0).

LOCATION: SDF main library

DOMAIN: **`SDF`** **(DERIVED FROM: `SDFStar`)**

VERSION: 1.6 (11/25/92)

AUTHOR: *J. T. Buck*

INPUTS: *input* (float)

OUTPUTS: *output* (float)

STATES: *gain* (FloatState): Gain of the star.
Default = 1.0

NAME: **`Hilbert`**

Output the (approximate) Hilbert transform of the input signal.

LOCATION: SDF dsp library

DOMAIN: **`SDF`** **(DERIVED FROM: `SDFFIR`)**

VERSION: 1.7 (12/8/92)

AUTHOR: *J. T. Buck*

STATES: *N* (IntState): Number of taps
Default = 64

DESCRIPTION:

This star approximates the Hilbert transform by using an FIR filter. The exact Hilbert transform cannot be realized; instead, we just chop off the response symmetrically at -N/2 and N/2. This is good enough for demos, but for high accuracy we suggest the use of the Parks-McClellan algorithm to design a Hilbert transformer filter with the desired characteristics. The "optfir" program supplied with Ptolemy can do this.

NAME: **`IIDGaussian`**

Generates a white Gaussian pseudo-random process with mean "mean" (default 0) and variance "variance" (default 1).

LOCATION: SDF main library

DOMAIN: **`SDF` (DERIVED FROM: `SDFStar`)**

VERSION: 1.11 (12/8/92)

AUTHOR: *D. G. Messerschmitt*

OUTPUTS: *output* (float)

STATES: *mean* (FloatState): Mean of distribution.
Default = 0.0
variance (FloatState): Variance of distribution.
Default = 1.0

DESCRIPTION:

This Star uses the GNU library <Normal.h>

NAME: **`IIDUniform`**

Generates an i.i.d. uniformly distributed pseudo-random process. Output is uniformly distributed between "lower" (default 0) and "upper" (default 1).

LOCATION: SDF main library

DOMAIN: **`SDF` (DERIVED FROM: `SDFStar`)**

VERSION: 1.11 (12/8/92)

AUTHOR: *D. G. Messerschmitt*

OUTPUTS: *output* (float)

STATES: *lower* (FloatState): Lower limit.
Default = 0.0
upper (FloatState): Upper limit.
Default = 1.0

DESCRIPTION:

This Star uses the GNU library <Uniform.h>.

NAME: **`IIR`**

A Infinite Impulse Response (IIR) filter. Coefficients are in the "numerator" and "denominator", both start with z^0 terms and decrease in powers of z.

LOCATION: SDF dsp library

DOMAIN: **`SDF` (DERIVED FROM: `SDFStar`)**

VERSION: 1.9 (12/8/92)

AUTHOR: *Kennard White*

INPUTS: *signalIn* (float)

OUTPUTS: *signalOut* (float)

STATES: *gain* (FloatState): Gain.
Default = 1
numerator (FloatArrayState): Numerator coefficients.
Default = .5 .25 .1
denominator (FloatArrayState): Denominator coefficients.
Default = 1 .5 .3
state (FloatArrayState): State.
Default = 0

DESCRIPTION:

This star implements a infinite-impulse response filter of arbirary order. The parameters of the star specify H(z), the Z-transform of an impulse response h(n). The output of the star is the convolution of the input with h(n).

The transfer function implemented is of the form H(z)=G*N(1/z)/D(1/z), where N() and D() are polynomials. The state "gain" specifies G, and the state arrays "numerator" and "denominator" specify N and D, respectively. Both arrays start with z^0 terms and decrease in powers of z (increase in powers of 1/z). Note that the leading term of D is *not* ommitted.

Note that the numerical finite precision noise increases with the filter order. It is often desirable to expand the filter into a parallel or cascade form.

SEE ALSO: FIR and Biquad.

NAME: **`ImageToPix`**

Accept a black-and-white image from an input packet, and generate the individual pixels.

LOCATION: SDF image library

DOMAIN: **`SDF`** **(DERIVED FROM: `SDFStar`)**

VERSION: 1.5 (12/8/92)

AUTHOR: *J. Buck*

INPUTS: *input* (message)

OUTPUTS: *output* (int)

STATES: *width* (IntState): width of image
Default = 100
height (IntState): height of image
Default = 100

NAME: **`Impulse`**

Generates a stream of impulses of size "level" (default 1.0). The period is given by "period" (default 0). If period = 0 then only one impulse is generated.

LOCATION: SDF main library

DOMAIN: **`SDF` (DERIVED FROM: `SDFStar`)**

VERSION: 2.5 (11/25/92)

AUTHOR: *J. T. Buck*

OUTPUTS: *output* (float)

STATES: *level* (FloatState): The height of the impulse.
Default = 1.0
period (IntState): The period of the impulse train, 0 = aperiodic.
Default = 0
count (IntState): An internal state.
Default = 0

NAME: **`Integrator`**

An integrator with leakage, limits, and reset. With the default parameters, input samples are simply accumulated, and the running sum is the output. To prevent any resetting in the middle of a run, connect a d.c. source with value 0.0 to the "reset" input. Otherwise, whenever a non-zero is received on this input, the accumulated sum is reset to the current input (i.e. no feedback).

Limits are controlled by the "top" and "bottom" parameters. If top <= bottom, no limiting is performed (default). Otherwise, the output is kept between "bottom" and "top". If "saturate" = YES, saturation is performed. If "saturate" = NO, wrap-around is performed (default). Limiting is performed before output.

Leakage is controlled by the "feedbackGain" state (default 1.0). The output is the data input plus feedbackGain*state, where state is the previous output.

LOCATION: SDF main library

DOMAIN: **`SDF` (DERIVED FROM: `SDFStar`)**

VERSION: 2.13 (12/8/92)

AUTHOR: *E. A. Lee*

INPUTS: *data* (float)
reset (int)

OUTPUTS: *output* (float)

STATES: *feedbackGain* (FloatState): The gain on the feedback path.
Default = 1.0
top (FloatState): The upper limit.

Default = 0.0
bottom (FloatState): The lower limit.
Default = 0.0
saturate (IntState): Saturate if YES, wrap around otherwise.
Default = YES
state (FloatState): An internal state.
Default = 0.0

DESCRIPTION:

NAME: `IntToBits`

Reads the least significant "nBits" bits from an integer input, and outputs the bits serially on the output, most significant bit first.

LOCATION: SDF main library

DOMAIN: `SDF` **(DERIVED FROM:** `SDFStar`)

VERSION: 1.3 (12/8/92)

AUTHOR: *J. T Buck*

INPUTS: *input* (int)

OUTPUTS: *output* (int)

STATES: *nBits* (IntState): number of bits read per execution
Default = 4

NAME: `Lattice`

A Forward Lattice filter. Default coefficients are the optimal predictor for a particular 4th order AR random process. To read reflection coefficients from a file, replace the default coefficients with "<fileName", preferably specifying a complete path.

LOCATION: SDF dsp library

DOMAIN: `SDF` **(DERIVED FROM:** `SDFStar`)

VERSION: 1.5 (12/8/92)

AUTHOR: *Alan Kamas*

INPUTS: *signalIn* (float)

OUTPUTS: *signalOut* (float)

STATES: *reflectionCoefs* (FloatArrayState): Reflection or PARCOR coefficients.
Default = 0.804534 -0.820577 0.521934 -0.205

DESCRIPTION:

This star implements a lattice filter. To load reflection coefficients from a file, simply replace the default coefficients with the string "<filename". It is advisable not to use an absolute path name as part of the file name, especially if you are using the graphical interface. This will allow the Lattice filter to work as expected regardless of the directory in which the ptolemy process actually runs. It is best to use tilde’s in the filename to reference them to user’s home

directory. This way, future filesystem reorganizations will have minimal effect.

The default reflection coefficients correspond to the optimal linear predictor for an AR process generated by filtering white noise with the following filter:

$$H(z) = \frac{1}{1 - 2z^{-1} + 1.91z^{-2} - 0.91z^{-3} + 0.205z^{-4}} \ .$$

Since this filter is minimum phase, the transfer function of the lattice filter is $H^{-1}(z)$.

Note that the definition of reflection coefficients is not quite universal in the literature. The reflection coefficients in references [2] and [3] are the negative of the ones used by this star, which correspond to the definition in most other texts, and to the definition of partial-correlation (PARCOR) coefficients in the statistics literature. The sign of the coefficients used in this star is appropriate for values given by the LevDur or Burg stars.

REFERENCES

[1] J. Makhoul, "Linear Prediction: A Tutorial Review", *Proc. IEEE*, Vol. 63, pp. 561-580, Apr. 1975.

[2] S. M. Kay, *Modern Spectral Estimation: Theory & Application*, Prentice-Hall, Englewood Cliffs, NJ, 1988.

[3] S. Haykin, *Modern Filters*, MacMillan Publishing Company, New York, 1989.

SEE ALSO: RLattice, BlockLattice, BlockRLattice, FIR, FIRCx and Biquad.

NAME: `LevDur`

Levinson-Durbin algorithm.

LOCATION: SDF dsp library

DOMAIN: `SDF` **(DERIVED FROM:** `SDFStar`)

VERSION: 1.13 (12/8/92)

AUTHOR: *E. A. Lee*

INPUTS: *autocor* (float): Autocorrelation estimate

OUTPUTS: *lp* (float): FIR linear predictor coefficients output.
refl (float): Lattice predictor coefficients output.
errPower (float): Prediction error power.

STATES: *order* (IntState): The order of the recursion.
Default = 8

DESCRIPTION:

This star takes inputs from the `Autocor` star and uses the Levinson-Durbin algorithm to compute both reflection coefficients and FIR linear predictor coefficients. If the `Autocor` star is set so that it's *unbiased* parameter is zero, then the combined effect of that star and this one is called the autocorrelation algorithm. The given order should be the same as the *noLags* parameter of the `Autocor` star. Three outputs are generated. On the *errPower* output, a sequence of $order + 1$ samples gives the prediction error power for each predictor order from zero to *order*. The first sample, the corresponding to the zeroth order predictor, is simply an estimate of the power of the input process. Note that for signals without noise, the *errPower* output can sometimes end up being a small negative number.

The *lp* output gives the coefficients of an FIR filter that performs linear prediction for the input process. This set of coefficients is suitable for directly feeding a **`BlockFIR`** filter star that accepts outside coefficients. The number of coefficients produced is equal to the *order*. The final output *refl* is the reflection coefficients, suitable for feeding directly to a **`BlockLattice`** star, which will then generate the forward and backward prediction error. The number of coefficients produced is equal to the *order*.

Note that the definition of reflection coefficients is not quite universal in the literature. The reflection coefficients in references [2] and [3] are the negative of the ones generated by this star, which correspond to the definition in most other texts, and to the definition of partial-correlation (PARCOR) coefficients in the statistics literature.

REFERENCES

[1] J. Makhoul, "Linear Prediction: A Tutorial Review", *Proc. IEEE*, Vol. 63, pp. 561-580, Apr. 1975.

[2] S. M. Kay, *Modern Spectral Estimation: Theory & Application*, Prentice-Hall, Englewood Cliffs, NJ, 1988.

[3] S. Haykin, *Modern Filters*, MacMillan Publishing Company, New York, 1989.

SEE ALSO: Autocor, BlockFIR and linearPrediction.

NAME: **`Limit`**

Hard limiter.

LOCATION: SDF main library

DOMAIN: **`SDF`** **(DERIVED FROM: `SDFStar`)**

VERSION: 1.5 (11/25/92)

AUTHOR: *E. A. Lee*

INPUTS: *input* (float)

OUTPUTS: *output* (float)

STATES: *bottom* (FloatState): Lower limit of the output.
Default = 0.0
top (FloatState): Upper limit of the output.
Default = 1.0

DESCRIPTION:

This star hard limits input samples to keep the in the range of (*bottom*, *top*).

NAME: `LinQuantIdx`

The input is quantized to "levels" number of levels for a specified signal ranging between "low" and "high". On the "amplitude" port, the quantized output is generated, while the "stepNumber" output is the quantization level. This integer output is particularly needed for the Thor stars that need an integer input.

DOMAIN: `SDF` **(DERIVED FROM:** `SDFStar`)

VERSION: 1.9 (12/8/92)

AUTHOR: *Asawaree Kalavade*

INPUTS: *input* (float)

OUTPUTS: *amplitude* (float)
stepNumber (int)

STATES: *levels* (IntState): number of levels to quantize to
Default = 128
low (FloatState): lower limit of signal excursion
Default = -3.0
high (FloatState): upper limit of signal excursion
Default = 3.0

NAME: `LMSCx`

Complex adaptive filter using LMS adaptation algorithm. Initial coefficients are in the "taps" state variable. Default initial coefficients give an 8th order, linear phase lowpass filter. To read initial coefficients from a file, replace the default coefficients with "<fileName". Supports decimation, but not interpolation.

LOCATION: SDF dsp library

DOMAIN: `SDF` **(DERIVED FROM:** `SDFFIRCx`)

VERSION: 1.6 (11/25/92)

AUTHOR: *J. T Buck, E. A. Lee*

INPUTS: *error* (complex)

STATES: *stepSize* (FloatState): Adaptation step size.
Default = 0.01
errorDelay (IntState): Delay in the update loop.
Default = 1
saveTapsFile (StringState): File to save final tap values.
Default =

DESCRIPTION:

When correctly used, this filter will adapt to try to minimize the mean-squared error of the signal at its *error* input. In order for this to be possible, the output of the filter should be compared (subtracted from) some reference signal to produce an error signal. That error signal should be fed back to the *error* input. The *delay* parameter must equal the total number of delays in the path from the output of the filter back to the error input. This ensures correct alignment of the adaptation algorithm. The number of delays must be greater than zero or the dataflow graph will deadlock. The adaptation algorithm used is the well-known LMS, or stochastic-gradient algorithm, generalized to use complex signals and filter taps.

If the *saveTapsFile* string is non-null, a file will be created by the name given by that string, and the final tap values will be stored there after the run has completed.

SEE ALSO: FIRCx, LMS and adaptFilter.

NAME: `LMSLeak`

An LMS-based adaptive filter in which the step size is input (to the "step" input) every iteration. The "stepSize" state is unused.

Values in the "taps" array change as the filter adapts to minimize the energy at the "error" input.

The "mu" state is used as a leak factor in the formula that updates the filter coefficients.

LOCATION: SDF dsp library

DOMAIN: `SDF` **(DERIVED FROM:** `SDFLMS`)

VERSION: 1.4 (11/25/92)

AUTHOR: *E. A. Lee, Paul Haskell*

INPUTS: *step* (float): Step-size for LMS algorithm.

STATES: *mu* (FloatState): Leak factor for coefficient update.
Default = 0.0

DESCRIPTION:

If two identical "LMSLeak" filters are used as an adaptive predictive coder and decoder, then with "mu" nearly equal to but greater than 0.0, the effects of channel errors between the coder and decoder will decay away rather than accumulate. As "mu" increases, the effects of channel errors decay away more quickly, but the size of the "error" input increases also.

See pg 54 of _Adaptive_Filters_, Honig and Messerschmitt.

SEE ALSO: LMS.

NAME: `LMSPlotCx`

Complex adaptive filter using LMS adaptation algorithm. In addition, the tap coefficients are plotted using the xgraph program, with a separate plot for the magnitude and phase. If "trace" is NO, only the final tap values are plotted; if it is YES, a trace of each tap is plotted as it adapts.

LOCATION: SDF dsp library

DOMAIN: `SDF` **(DERIVED FROM:** `SDFLMSCx`)

VERSION: 1.10 (11/25/92)

AUTHOR: *J. T. Buck*

STATES: *graphOptsMag* (StringState): options for magnitude graph
Default =
graphOptsPhase (StringState): options for phase graph
Default =
graphTitleMag (StringState): title for magnitude graph
Default = LMS filter tap magnitudes
graphTitlePhase (StringState): title for phase graph
Default = LMS filter tap phases
trace (IntState): if YES, plot a trace of tap values as they adapt
Default = NO

DESCRIPTION:

This star is exactly like the LMSCx star, except that, in addition, it makes a plot of the tap coefficients, one plot for the magnitude, one for the phase. It can produce two types of plots: a plot of the final tap values or a plot that traces the time evolution of each tap value: the time evolution is obtained if *trace* is YES.

GraphTitleMag is used for the title of the magnitude plot; *GraphTitlePhase* is used for the title of the phase plot; *GraphOptsMag* is handed to the xgraph program as option values on the command line when the magnitudes are plotted, and *GraphOptsPhase* serves the same function for the phase.

If *trace* is YES, there may not be more than 64 taps in the filter.

SEE ALSO: LMSCx, Xgraph and XMgraph.

NAME: `LMSPlot`

Adaptive filter using LMS adaptation algorithm. In addition, the tap coefficients are plotted using the xgraph program, with a separate plot for the magnitude and phase. If "trace" is NO, only the final tap values are plotted; if it is YES, a trace of each tap is plotted as it adapts.

LOCATION: SDF dsp library

DOMAIN: `SDF` **(DERIVED FROM:** `SDFLMS`)

VERSION: 1.5 (11/25/92)

AUTHOR: *J. T. Buck*

STATES: *graphOptions* (StringState): options for graph
Default = -0 taps
graphTitle (StringState): title for graph
Default = LMS filter tap values
trace (IntState): if YES, plot a trace of tap values as they adapt
Default = NO

DESCRIPTION:

This star is exactly like the LMS star, except that, in addition, it makes a plot of the tap coefficients. It can produce two types of plots: a plot of the final tap values or a plot that traces the time evolution of each tap value: the time evolution is obtained if *trace* is YES.

graphTitle is used for the title of the plot; *graphOptions* is handed to the xgraph program as option values on the command line.

If *trace* is YES, there may not be more than 64 taps in the filter.

SEE ALSO: LMS, Xgraph and XMgraph.

NAME: `LMS`

Adaptive filter using LMS adaptation algorithm. Initial coefficients are in the "taps" state variable. Default initial coefficients give an 8th order, linear phase lowpass filter. To read default coefficients from a file, replace the default coefficients with "<fileName". Supports decimation, but not interpolation.

LOCATION: SDF dsp library

DOMAIN: `SDF` **(DERIVED FROM:** `SDFFIR`)

VERSION: 2.11 (11/25/92)

AUTHOR: *E. A. Lee*

INPUTS: *error* (float)

STATES: *stepSize* (FloatState): Adaptation step size.
Default = 0.01
errorDelay (IntState): Delay in the update loop.
Default = 1
saveTapsFile (StringState): File to save final tap values.
Default =

DESCRIPTION:

When correctly used, this filter will adapt to try to minimize the mean-squared error of the signal at its *error* input. In order for this to be possible, the output of the filter should be compared (subtracted from) some reference signal to produce an error signal. That error signal should be fed back to the *error* input. The *delay* parameter must equal the total number of delays in the path from the output of the filter back to the error input. This ensures correct alignment of the adaptation algorithm. The number of delays must be greater than zero or the dataflow graph will deadlock. The adaptation algorithm used is the well-known LMS, or stochastic-gradient algorithm.

If the *saveTapsFile* string is non-null, a file will be created by the name given by that string, and the final tap values will be stored there after the run has completed.

SEE ALSO: FIR and adaptFilter.

NAME: **`Log`**

Outputs natural log of input.

LOCATION: SDF main library

DOMAIN: **`SDF`** **(DERIVED FROM: `SDFStar`)**

VERSION: 1.6 (11/25/92)

AUTHOR: *J. T. Buck*

INPUTS: *input* (float)

OUTPUTS: *output* (float)

DESCRIPTION:

Outputs natural log of input. If the input is zero or negative, the run is aborted.

NAME: **`MakeSeqATMCell`**

Inputs ’numInfoBits’ bits and loads them into a SeqATMCell object.

LOCATION: SDF conversion palette

DOMAIN: **`SDF`** **(DERIVED FROM: `SDFStar`)**

VERSION: 1.7 (12/8/92)

AUTHOR: *GSWalter*

INPUTS: *input* (int)

OUTPUTS: *output* (message)

STATES: *numInfoBits* (IntState): Number of information bits contained in a SeqATMCell message type.
Default = 384

headerLength (IntState): Number of bits in SeqATMCell to skip before loading in the information bits.
Default = 40

NAME: **`MedianImage`**

Accept an input GrayImage, median-filter the image, and send the result to the output.

Filter widths of 1, 3, 5 work well. Any length longer than five will take a long time to run.

Median filtering is useful for removing impulse-type noise from images. It also smooths out textures, so it is a useful pre-processing step before edge detection. It removes inter-field flicker quite well when displaying single frames from a moving sequence.

LOCATION: SDF image library

DOMAIN: **`SDF`** **(DERIVED FROM: `SDFStar`)**

VERSION: 1.12 (12/8/92)

AUTHOR: *Paul Haskell*

INPUTS: *inData* (message)

OUTPUTS: *outData* (message)

STATES: *FilterWidth* (IntState): Size of median filter window (should be odd number).
Default = 3

DESCRIPTION:

NAME: **`MotionCmpInv`**

For NULL inputs on 'mvIn' (motion-vector-in), copy the 'diffIn' input unchanged to the output and discard the 'pastIn' input. (A NULL input usually indicates the first frame of a sequence.)

For non-NULL 'mvIn' inputs, perform inverse motion compensation and write the result to 'output'.

LOCATION: SDF image palette

DOMAIN: **`SDF`** **(DERIVED FROM: `SDFStar`)**

VERSION: 1.15 (11/30/92)

AUTHOR: *Paul Haskell*

INPUTS: *diffIn* (message)
pastIn (message)
mvIn (message)

OUTPUTS: *output* (message)

DESCRIPTION:

NAME: `MotionCmp`

If the 'past' input is not a GrayImage (e.g. 'dummyMessage'), copy the 'input' image unchanged to the 'diffOut' output and send a null field of motion vectors to the 'mvOut' output. This should usually happen only on the first firing of the star.

For all other inputs, perform motion compensation and write the difference frames and motion vector frames to the corresponding outputs.

This star can be derived from, to implement slightly different motion compensation algorithms. For example, synchronization techniques can be added or reduced-search motion compensation can be performed.

LOCATION: SDF image palette

DOMAIN: `SDF` **(DERIVED FROM:** `SDFStar`)

VERSION: 1.17 (12/8/92)

AUTHOR: *Paul Haskell*

INPUTS: *input* (message)
past (message)

OUTPUTS: *diffOut* (message)
mvOut (message)

STATES: *BlockSize* (IntState): Size of blocks on which to do motion comp.
Default = 8

DESCRIPTION:

NAME: `MpyCx`

Output the product of the inputs, as a complex value.

LOCATION: SDF main library

DOMAIN: `SDF` **(DERIVED FROM:** `SDFStar`)

VERSION: 2.5 (11/25/92)

AUTHOR: *J. T. Buck*

INPUTS: *input* (multiple), (complex)

OUTPUTS: *output* (complex)

NAME: **`Mpy`**

Output the product of the inputs, as a float value.

LOCATION: SDF main library

DOMAIN: **`SDF` (DERIVED FROM: `SDFStar`)**

VERSION: 2.6 (11/25/92)

AUTHOR: *J. T. Buck*

INPUTS: *input* (multiple), (float)

OUTPUTS: *output* (float)

NAME: **`MuLaw`**

This star compresses its input as per the Mu law. using the conversion formula
|v2| = log (1+ mu*|v1|) / log(1+mu)

DOMAIN: **`SDF` (DERIVED FROM: `SDFStar`)**

VERSION: 1.8 (11/25/92)

AUTHOR: *Asawaree Kalavade*

INPUTS: *input* (float)

OUTPUTS: *output* (float)

STATES: *compress* (IntState): 0 to turn off compression, otherwises compresses with Mu law.
Default = 1
mu (IntState): mu parameter
Default = 255

DESCRIPTION:

NAME: **`Mux`**

Multiplexes any number of inputs onto one output stream. B particles are consumed on each input, where B is the blockSize. But only one of these blocks of particles is copied to the output. The one copied is determined by the "control" input. Integers from 0 through N-1 are accepted at the "control" input, where N is the number of inputs. If the control input is outside this range, an error is signaled.

LOCATION: SDF main library

DOMAIN: `SDF` **(DERIVED FROM:** `SDFStar`)

VERSION: 1.8 (12/8/92)

AUTHOR: *E. A. Lee*

INPUTS: *input* (multiple), (ANYTYPE)
control (int)

OUTPUTS: *output* (ANYTYPE)

STATES: *blockSize* (IntState): Number of particles in a block.
Default = 1

DESCRIPTION:

NAME: `nonLinearDistortion`

Generate second and third harmonic distortion by squaring and cubing the signal, and adding in controlled proportion to the original signal.

LOCATION: SDF communications palette (galaxy)

DOMAIN: `SDF` **(DERIVED FROM:** `SDFGalaxy`)

VERSION: 1.112/17/92 ()

AUTHOR: *E. A. Lee*

INPUTS: *in* (float)

OUTPUTS: *out* (float)

STATES: *secondHarmonic* (FloatState): The gain applied to second harmonic distortion (gotten by squaring the signal).
Default = 0.0
thirdHarmonic (FloatState): The gain applied to third harmonic distortion (gotten by cubing signal).
Default = 0.0

DESCRIPTION:

This galaxy squares and cubes the signal. The square is multiplied by *secondHarmonic* and the cube by *thirdHarmonic*, and the results are added to the original signal.

NAME: `NumToBus`

Receive as its input an integer corresponding to the bus contents and it converts it to a bus with the required number of lines (which is read in as a parameter).

DOMAIN: `SDF` **(DERIVED FROM:** `SDFStar`)

VERSION: 1.9 (11/25/92)

AUTHOR: *Asawaree Kalavade*

INPUTS: *input* (int)

OUTPUTS: *output* (multiple), (int)

STATES: *bits* (IntState): number of bits to output
Default = 7

NAME: `Pad`

On each execution, reads a block of "nread" samples (default 128) and writes a block of "nwrite" samples (default 256), where "nwrite" >= "nread". The samples may be of any type.

The first "offset" samples (default 0) have a zero value of the appropriate type (messages of type DUMMY for message connections), the next "nread" output samples have values taken from the inputs, and the remaining "nwrite" - "nread" - "offset" samples have value 0 again.

LOCATION: SDF main library

DOMAIN: `SDF` **(DERIVED FROM:** `SDFStar`)

VERSION: 1.7 (12/8/92)

AUTHOR: *E. A. Lee, J. T. Buck*

INPUTS: *input* (anytype)

OUTPUTS: *output* (ANYTYPE)

STATES: *nread* (IntState): Number of particles read.
Default = 128
nwrite (IntState): Number of particles written.
Default = 256
offset (IntState): Number of leading fill particles in each output block.
Default = 0

NAME: `PattMatch`

This star accepts a template and a search window. The template is slid over the window one sample at a time and cross correlations are calculated at each step. The cross-correlations are output on the 'values' output. The 'index' output is the value of the time-shift which gives the largest cross correlation. This index refers to a position on the search window beginning with 0 corresponding to the earliest arrived sample of the search window which is part of the "best match" with the template. for viewing the values of the cross correlations.

LOCATION: SDF dsp palette

DOMAIN: `SDF` **(DERIVED FROM:** `SDFStar`)

VERSION: 1.7 (12/8/92)

AUTHOR: *GSWalter, E. A. Lee*

INPUTS: *templ* (float)
window (float)

OUTPUTS: *index* (int)
values (float)

STATES: *tempSize* (IntState): number of samples in template
Default = 32
winSize (IntState): number of samples in search window
Default = 176

NAME: **`PCMBitCoder`**

Converts voice samples to 64 kbps bit stream using CCITT Recommendation G.711.

LOCATION: SDF conversion palette (galaxy)

DOMAIN: **`SDF`** **(DERIVED FROM: `SDFGalaxy`)**

VERSION: 1.1 (12/17/92)

AUTHOR: *G.S. Walter*

INPUTS: *input* (float)

OUTPUTS: *output* (int)

DESCRIPTION:

This galaxy is designed to encode a sample of speech using CCITT Recommendation G.711. The input is a sample of 8 kHz voice and the output consists of the quantized samples's eight-bit codeword.

It is interesting to note with this design the functionality of the coder around 0. Any input value between [-0.5, 0.5] should be given the encoded value of 0. But 0 may be represented in bits as either 10000000 (decimal 128) or 00000000 (decimal 0). Due to the nature of the **`Sgn`** star the levels chosen for the **`Quantizer`** star, the values in [-0.5, 0) will be encoded as 00000000 and those between [0, 0.5] will be transformed to 10000000. The decoder, **`PCMBitDecoder,`** is able to recognize both implementations as the decoded magnitude 0.

Otherwise, the encoding is straight forward. The magnitude of an input sample is put through a quantizer which will return the decimal equivalent of the proper 8-bit codeword. The upper quantizer is used to modify the sign bit of the codeword. The **`IntToBits`** star will then output the proper eight bits.

SEE ALSO: **`PCMBitDecoder`**

NAME: **`PCMBitDecoder`**

Converts 64 kbps stream of PCM-encoded voice to proper quantized levels. Decoding follows CCITT Recommendation G.711.

LOCATION: SDF conversion palette (galaxy)

DOMAIN: **`SDF`** **(DERIVED FROM: `SDFGalaxy`)**

VERSION: 1.1 (12/17/92)

AUTHOR: *G.S. Walter*

INPUTS: *input* (int)

OUTPUTS: *output* (float)

DESCRIPTION:

The decoder performs the decoding function for CCITT Recommendation G.711 on bit streams encoded with **`PCMBitCoder.`** This function is performed by converting the 8-bit codeword into its decimal equivalent and using a look-up table to determine the proper decoded magnitude.

SEE ALSO: **`PCMBitCoder`**

NAME: **`periodogram`**

Estimate a power spectrum using the periodogram method.

LOCATION: SDF dsp library (galaxy)

DOMAIN: **`SDF`** **(DERIVED FROM: `SDFGalaxy`)**

VERSION: 1.1 (12/17/92)

AUTHOR: *E. A. Lee*

INPUTS: *input* (float)

OUTPUTS: *output* (float)

STATES: *numInputs* (int): The number of inputs to use to construct the estimate.
Default = 512
log2order (int): The log base 2 of the FFT order.
Default = 9

DESCRIPTION:

This galaxy estimates the power spectrum of its input using the periodogram method. This consists simply of computing the magnitude squared of the DFT of a set of observations of the signal. The *numInputs* parameter specifies the number of observation samples to collect. The *log2order* parameter gives the log base 2 of the FFT order to use. If $2^{log2order}$ is larger than *numInputs*, then zero padding is done.

Prior to computing the spectrum, the signal is modulated with a signal consisting of alternating + and - one, generated by the **`WaveForm`** star. This signal shifts the d.c. component of the signal to the Nyquist frequency, resulting in a spectrum that is centered. Without this modulation, the d.c. component would be the first output of the galaxy, and the last would be the component just below the sampling frequency.

REFERENCES

[1] S. M. Kay, *Modern Spectral Estimation: Theory & Application*, Prentice-Hall, Englewood Cliffs, NJ, 1988.

SEE ALSO: powerSpectrum

NAME: **`phaseShift`**

Phase shift galaxy.

LOCATION: SDF dsp library (galaxy)

DOMAIN: **`SDF`** **(DERIVED FROM: `SDFStar`)**

VERSION: 1.412/5/92 ()

AUTHOR: *E. A. Lee*

INPUTS: *in* (float): Input signal
shift (float): Amount of phase shift

OUTPUTS: *out* (float): Phase shifted signal

DESCRIPTION:

This galaxy applies a phase shift to a signal according to the *shift* input. If the *shift* input value is time varying, then its slope determines the instantaneous frequency shift.

The phase shift is accomplished by first forming an approximately analytic signal by filtering the input with a complex FIR filter. The filter taps are given in the file ˜ptolemy/src/domains/sdf/demo/analytic2.cx. This filter has approximately unity gain from d.c. up to the Nyquist frequency, and imposes at least 48 dB of attenuation from d.c. down to the negative of the Nyquist frequency. Hence, only positive frequencies are allowed through. (The pigi command *DFT of Cx signal* (‘‘_’’) may be used to examine the details of this frequency response.) The analytic signal is modulated with a complex exponential to shift the phase, and the real part of the result is output.

SEE ALSO: doppler.

NAME: **`PixToImage`**

Accept integer values, and assemble them into an image. Warning: this star produces large sample rate changes, use the loop scheduler!

LOCATION: SDF image library

DOMAIN: **`SDF`** **(DERIVED FROM: `SDFStar`)**

VERSION: 1.4 (12/8/92)

AUTHOR: *J. Buck*

INPUTS: *input* (int)

OUTPUTS: *output* (message)

STATES: *width* (IntState): width of image
Default = 100

height (IntState): height of image
Default = 100

NAME: `Play`

Play an input stream on the SparcStation speaker. The "gain" state (default 1.0) multiplies the input stream before it is mu-law compressed and written. The inputs should be in the range of -32000.0 to 32000.0. The file is played at about 8000 samples/second (sorry, fixed). When the wrapup method is called, a file of 8-bit mu-law samples is handed to a program named "play" which plays the file. The "play" program must be in your path.

LOCATION: SDF main library

DOMAIN: `SDF` **(DERIVED FROM:** `SDFStar`)

VERSION: 1.18 (12/7/92)

AUTHOR: *J. T. Buck*

INPUTS: *input* (float)

STATES: *gain* (FloatState): Output gain.
Default = 1.0
saveFile (StringState): File to save the output mu-law samples.
Default =

DESCRIPTION:

Generate a file for the SparcStation speaker and play it. The "play" program must be on the user's path. This star may be used on a different device provided that a "play" program is written with the following specifications:

When invoked as "play filename", where filename is a sequence of bytes representing mu-law PCM samples, the program should play the file at 8000 samples per second. Perhaps later this rate can be a parameter.

NAME: `PolarToRect`

Convert magnitude and phase to rectangular form.

LOCATION: SDF main library

DOMAIN: `SDF` **(DERIVED FROM:** `SDFStar`)

VERSION: 1.7 (11/25/92)

AUTHOR: *E. A. Lee*

INPUTS: *magnitude* (float)
phase (float)

OUTPUTS: *x* (float)
y (float)

DESCRIPTION:

NAME: **`powerEst`**

Estimate the power in dB by filtering the square of the input using a first order filter with the time constant given as a number of sample periods.

LOCATION: SDF nonlinear palette (galaxy)

DOMAIN: **`SDF`** **(DERIVED FROM: `SDFGalaxy`)**

VERSION: 1.112/17/92 ()

AUTHOR: *Joe Buck*

INPUTS: *in* (float)

OUTPUTS: *out* (float)

STATES: *timeConstant_in_samples* (FloatState): The time constant (in number of sample periods)
Default = 100

DESCRIPTION:

The input signal is squared, and passed through a first order filter with transfer function

$$H(z) = \frac{1}{1 - \alpha z^{-1}}$$

where

$$\alpha = 1 - \frac{1}{\tau}$$

where τ is the *timeConstant_in_samples* state.

SEE ALSO: Integrator dB

NAME: **`Printer`**

Prints out one sample from each input port per line If "fileName" is not equal to "<stdout>" (the default), it specifies the filename to write to. <stderr> prints on the standard error stream.

LOCATION: SDF main library

DOMAIN: **`SDF`** **(DERIVED FROM: `SDFStar`)**

VERSION: 2.11 (12/8/92)

AUTHOR: *D. G. Messerschmitt and J. Buck*

INPUTS: *input* (multiple), (ANYTYPE)

STATES: *fileName* (StringState): Filename for output.
Default = <stdout>

DESCRIPTION:

This star prints its input, which may be any supported type. There may be multiple inputs: all inputs are printed together on the same line, separated by tabs.

NAME: `QAM16`

Encode an input bit stream in a 16-QAM complex symbol sequence.

LOCATION: SDF comm library (galaxy)

DOMAIN: `SDF` **(DERIVED FROM:** `Galaxy`)

VERSION: 1.1 (12/17/92)

AUTHOR: *J. Buck*

INPUTS: *in* (int): input bits

OUTPUTS: *out* (complex): output symbols

DESCRIPTION:

This galaxy translates each pair of input bits int one of the following complex numbers where the real and imaginary parts are either ±1 or ±3.

SEE ALSO: bits QAM4

NAME: `QAM4`

Encode an input bit stream in a 4-QAM (or 4-PSK) complex symbol sequence.

LOCATION: SDF comm library (galaxy)

DOMAIN: `SDF` **(DERIVED FROM:** `Galaxy`)

VERSION: 1.1 (12/17/92)

AUTHOR: *J. Buck*

INPUTS: *in* (int): input bits

OUTPUTS: *out* (complex): output symbols

DESCRIPTION:

This galaxy translates each pair of input bits int one of the following complex numbers: {(1,1) (-1,1) (1,-1) (-1,-1)}.

SEE ALSO: bits QAM16

NAME: **`Quant`**

Quantizes input to one of N+1 possible output levels using N thresholds. For an input less than or equal to the n-th threshold, but larger than all previous thresholds, the output will be the n-th level. If the input is greater than all thresholds, the output is the N+1-th level. If level is specified, there must be one more level than thresholds; the default value for level is 0, 1, 2, ... N.

LOCATION: SDF main library

DOMAIN: **`SDF`** **(DERIVED FROM: `SDFStar`)**

VERSION: 1.6 (12/8/92)

AUTHOR: *E. A. Lee and J. Buck*

INPUTS: *input* (float)

OUTPUTS: *output* (float)

STATES: *thresholds* (FloatArrayState): Quantization thresholds, in increasing order
Default = 0.0
levels (FloatArrayState): Output levels. If empty, use 0, 1, 2, ...
Default =

NAME: **`RaisedCosCx`**

Output a complex raised-cosine Nyquist pulse.

LOCATION: SDF dsp library

DOMAIN: **`SDF`** **(DERIVED FROM: `SDFFIRCx`)**

VERSION: 1.9 (11/25/92)

AUTHOR: *J. T. Buck*

STATES: *N* (IntState): Number of taps
Default = 64
P (IntState): Distance from center to first zero crossing
Default = 16
excessBW (FloatState): Excess bandwidth, between 0 and 1
Default = 1.0

DESCRIPTION:

This star outputs a complex raised cosine pulse with excess bandwidth given by *excessbw* and pulse width (distance from center to first zero crossing) given by *P*. The length of the filter (number of taps) is *N*, and the output sample rate is *upsample* times the input. This star is implemented by deriving from the ComplexFIR star.

See "Digital Communication" by Lee and Messerschmitt for a discussion of raised cosine pulses in communications systems.

SEE ALSO: FIRCx and RaisedCos.

NAME: `RaisedCos`

An FIR filter with a magnitude frequency response shaped like the standard raised cosine used in digital communications. By default, the star upsamples by a factor of 16, so 16 outputs will be produced for each input unless the "interpolation" parameter is changed.

LOCATION: SDF dsp library

DOMAIN: `SDF` **(DERIVED FROM:** `SDFFIR`)

VERSION: 2.9 (11/25/92)

AUTHOR: *J. T. Buck*

STATES: *N* (IntState): Number of taps
Default = 64
P (IntState): Distance from center to first zero crossing
Default = 16
excessBW (FloatState): Excess bandwidth, between 0 and 1
Default = 1.0

DESCRIPTION:

This star implements an FIR filter with a raised cosine frequency response, with the excess bandwidth given by *excessbw* and the distance from center to first zero crossing given by *P*. The length of the filter (the number of taps) is *N*. Ideally, the impulse response of the filter would be

$$h(n) = \left(\frac{\sin(\pi n / P)}{\pi n / P} \right) \left(\frac{\cos(\alpha \pi n / P)}{1 minus\, 2 \alpha n / P} \right)$$

where α is *excessbw*. However, this pulse is centered at zero, and we can only implement causal filters in the SDF domain in Ptolemy. Hence, the inpulse response is actually

$$g(n) = h(n - M)$$

where $M = N/2$ if N is even, and $M = (N+1)/2$ if N is odd. Since the impulse response is simply truncated outside this range, note that if N is even the impulse response will not be symmetric. It will have one more sample to the left than to the right of center. Unless this extra sample is zero, the filter will not have linear phase if N is even.

The output sample rate is *upsample* times the input. This is set by default to 16 becuase in digital communication systems this pulse is used for line coding of symbols, and upsampling is necessary. The star is implemented by deriving from the FIR star.

REFERENCES

[1] E. A. Lee and D. G. Messerchmitt, "Digital Communication," Kluwer Academic Publishers, 1988.

SEE ALSO: FIR.

NAME: **`RampInt`**

Generates an integer ramp signal, starting at value (default 0) with step size step (default 1).

LOCATION: SDF main library

DOMAIN: **`SDF`** **(DERIVED FROM: `SDFStar`)**

VERSION: 2.5 (11/25/92)

AUTHOR: *D. G. Messerschmitt*

OUTPUTS: *output* (int)

STATES: *step* (IntState): Ramp step.
Default = 1
value (IntState): Initial (or latest) value output by Ramp.
Default = 0.0

NAME: **`Ramp`**

Generates a ramp signal, starting at "value" (default 0) with step size "step" (default 1).

LOCATION: SDF main library

DOMAIN: **`SDF`** **(DERIVED FROM: `SDFStar`)**

VERSION: 2.5 (11/25/92)

AUTHOR: *D. G. Messerschmitt*

OUTPUTS: *output* (float)

STATES: *step* (FloatState): Increment from one sample to the next.
Default = 1.0
value (FloatState): Initial (or latest) value output by Ramp.
Default = 0.0

NAME: **`ReadFile`**

Read ASCII data from a file.

LOCATION: SDF main library

DOMAIN: **`SDF`** **(DERIVED FROM: `SDFStar`)**

VERSION: 1.7 (12/8/92)

AUTHOR: *T. M. Parks*

OUTPUTS: *output* (float)

STATES: *fileName* (StringState): Input file.
Default = /dev/null
haltAtEnd (IntState): Halt the run at the end of the input file.
Default = NO
periodic (IntState): Make output periodic or zero-padded.

Default = YES

NAME: **`ReadImage`**

Read a sequence of PGM-format images from different files and send them out in Envelopes (containing data of type GrayImage).

If present, the character '#' in the 'fileName' state is replaced with the frame number to be read next. For example, if the 'frameId' state is set to 2 and if the 'fileName' state is 'dir.#/pic#' then the files that are read and output are 'dir.2/pic2', 'dir.3/pic3', etc.

LOCATION: SDF image library

DOMAIN: **`SDF`** **(DERIVED FROM: `SDFStar`)**

VERSION: 1.10 (11/25/92)

AUTHOR: *Paul Haskell*

OUTPUTS: *output* (message)

STATES: *fileName* (StringState): Name of file containing PGM-format image.
Default = ˜ptolemy/src/domains/sdf/demo/ptimage
frameId (IntState): Starting frame ID value.
Default = 0

DESCRIPTION:

NAME: **`ReadPCM`**

Read a binary mu-law encoded PCM file. Return one sample each time. The file format that is read is the same as the one written by the Play star.

LOCATION: SDF main library

DOMAIN: **`SDF`** **(DERIVED FROM: `SDFStar`)**

VERSION: 1.4 (12/8/92)

AUTHOR: *J. Buck*

OUTPUTS: *output* (int)

STATES: *fileName* (StringState): Input file.
Default = /dev/null
haltAtEnd (IntState): Halt the run at the end of the input file.
Default = NO
periodic (IntState): Make output periodic or zero-padded.
Default = YES

NAME: **`ReadRGB`**

Read a PPM-format image from a file and send it out in three inputs--a Red, Green, and Blue image. Each image is of type GrayImage.

If present, the character ’#’ in the ’fileName’ state is replaced with the frame number to be read next. For example, if the ’frameId’ state is set to 2 and if the ’fileName’ state is ’dir.#/pic#’ then the files that are read and output are ’dir.2/pic2’, ’dir.3/pic3’, etc.

LOCATION: SDF image library

DOMAIN: **`SDF`** **(DERIVED FROM: `SDFStar`)**

VERSION: 1.9 (11/25/92)

AUTHOR: *Sun-Inn Shih*

OUTPUTS: *output1* (message)
output2 (message)
output3 (message)

STATES: *fileName* (StringState): Name of file containing PPM-format image.
Default = ˜ptolemy/src/domains/sdf/demo/ppimage
frameId (IntState): Starting frame ID value.
Default = 0

DESCRIPTION:

NAME: **`Reciprocal`**

1/x, with an optional magnitude limit.

LOCATION: SDF main library

DOMAIN: **`SDF`** **(DERIVED FROM: `SDFStar`)**

VERSION: 1.5 (11/25/92)

AUTHOR: *E. A. Lee*

INPUTS: *input* (float)

OUTPUTS: *output* (float)

STATES: *magLimit* (FloatState): If non-zero, limits the magnitude of the output.
Default = 0.0

DESCRIPTION:

This star computes $1/x$, where x is the input. If the *magLimit* parameter is not 0.0, then the output is $\pm \max(magLimit, 1/x)$. In this case, x can be zero without causing an floating exception. The sign of the output is determined by the sign of the input.

NAME: **`Rect`**

Generates a rectangular pulse of height "height" (default 1.0). and width "width" (default 8). If "period" is greater than zero, then the pulse is repeated with the given period.

LOCATION: SDF main library

DOMAIN: **`SDF`** **(DERIVED FROM: `SDFStar`)**

VERSION: 2.6 (11/25/92)

AUTHOR: *J. T. Buck*

OUTPUTS: *output* (float)

STATES: *height* (FloatState): Height of the rectangular pulse.
Default = 1.0
width (IntState): Width of the rectangular pulse.
Default = 8
period (IntState): If greater than zero, the period of the pulse stream.
Default = 0
count (IntState): Internal counting state.
Default = 0

NAME: **`RectToCx`**

Convert real and imaginary parts to a complex output.

LOCATION: SDF main library

DOMAIN: **`SDF`** **(DERIVED FROM: `SDFStar`)**

VERSION: 1.6 (11/25/92)

AUTHOR: *E. A. Lee*

INPUTS: *real* (float)
imag (float)

OUTPUTS: *output* (complex)

DESCRIPTION:

NAME: **`RectToPolar`**

Convert two numbers to magnitude and phase. The phase output is in the range -PI to PI.

LOCATION: SDF main library

DOMAIN: `SDF` **(DERIVED FROM:** `SDFStar`)

VERSION: 1.7 (11/25/92)

AUTHOR: *E. A. Lee*

INPUTS: *x* (float)
y (float)

OUTPUTS: *magnitude* (float)
phase (float)

DESCRIPTION:

NAME: `Repeat`

Repeats each input sample the specified number of times.

LOCATION: SDF main library

DOMAIN: `SDF` **(DERIVED FROM:** `SDFStar`)

VERSION: 2.5 (12/8/92)

AUTHOR: *E. A. Lee*

INPUTS: *input* (ANYTYPE)

OUTPUTS: *output* (ANYTYPE)

STATES: *numTimes* (IntState): Repetition factor.
Default = 2

DESCRIPTION:

Repeat repeats each input Particle the specified number of times (*numTimes*) on the output. Note that this is a sample rate change, and hence affects the number of invocations of downstream stars.

NAME: `Reverse`

On each execution, reads a block of "N" samples (default 64) and writes them out backwards.

LOCATION: SDF main library

DOMAIN: `SDF` **(DERIVED FROM:** `SDFStar`)

VERSION: 1.3 (12/8/92)

AUTHOR: *J. T. Buck*

INPUTS: *input* (ANYTYPE)

OUTPUTS: *output* (ANYTYPE)

STATES: *N* (IntState): Number of particles read and written.
Default = 64

NAME: **`RGBToYUV`**

Read three GrayImages that describe a color image in RGB format and output three GrayImages that describe an image in YUV format. No downsampling is done on the U and V signals. The inputs and outputs are all GrayImages.

LOCATION: SDF image library

DOMAIN: **`SDF`** **(DERIVED FROM: `SDFStar`)**

VERSION: 1.8 (11/25/92)

AUTHOR: *Sun-Inn Shih*

INPUTS: *input1* (message)
input2 (message)
input3 (message)

OUTPUTS: *output1* (message)
output2 (message)
output3 (message)

DESCRIPTION:

NAME: **`RLattice`**

A Recursive (Backward) (IIR) Lattice filter. The default coefficient implement the synthesis filter for a particular 4th order AR random process. To read reflection coefficients from a file, replace the default coefficients with "<fileName", preferably specifying a complete path.

LOCATION: SDF dsp library

DOMAIN: **`SDF`** **(DERIVED FROM: `SDFStar`)**

VERSION: 1.6 (12/17/92)

AUTHOR: *Alan Kamas*

INPUTS: *signalIn* (float)

OUTPUTS: *signalOut* (float)

STATES: *reflectionCoefs* (FloatArrayState): Reflection or PARCOR coefficients.
Default = 0.804534 -0.820577 0.521934 -0.205

DESCRIPTION:

This star implements a recursive lattice filter. To load filter coefficients from a file, simply replace the default coefficients with the string "<filename". It is advisable not to use an absolute path name as part of the file name, especially if you are using the graphical interface. This will allow the Lattice filter to work as expected regardless of the directory in which the ptolemy process actually runs. It is best to use tilde's in the filename to reference them to user's home directory. This way, future filesystem reorganizations will have minimal effect.

The structure is as follows:

y[0] y[1] y[n-1] y[n]

```
X(n)  ---(+)->--o-->----(+)->--o--->-- ... ->--(+)->--o--->---o--->  Y(n)
          \    /          \    /               \    /        |
          +Kn /          +Kn-1 /               +K1 /         |
            X               X                    X           |
          -Kn \          -Kn-1 \               -K1 \         V
          /    \          /    \               /    \        |
        (+)-<--o--[z]--(+)-<--o--[z]- ... -<--(+)-<--o--[z]--/
              w[1]           w[2]                  w[n]
```

where the [z] are unit delays and the (+) are adders. and "y" and "z" are defined in the code.

The reflection (or PARCOR) coefficients should be specified left to right, K1 to Kn as above. Using exactly the same coefficients in the **`Lattice`** star will result in precisely the inverse transfer function. The default reflection coefficients give the following transfer function:

$$H(z) = \frac{1}{1 - 2z^{-1} + 1.91z^{-2} - 0.91z^{-3} + 0.205z^{-4}} .$$

Hence, the same coefficients in the **`Lattice`** star will give transfer function $H^{-1}(z)$.

Note that the definition of reflection coefficients is not quite universal in the literature. The reflection coefficients in references [2] and [3] are the negative of the ones used by this star, which correspond to the definition in most other texts, and to the definition of partial-correlation (PARCOR) coefficients in the statistics literature. The sign of the coefficients used in this star is appropriate for values given by the LevDur or Burg stars.

REFERENCES

[1] J. Makhoul, "Linear Prediction: A Tutorial Review", *Proc. IEEE*, Vol. 63, pp. 561-580, Apr. 1975.

[2] S. M. Kay, *Modern Spectral Estimation: Theory & Application*, Prentice-Hall, Englewood Cliffs, NJ, 1988.

[3] S. Haykin, *Modern Filters*, MacMillan Publishing Company, New York, 1989.

SEE ALSO: IIR and Lattice.

NAME: **`RunLenImageInv`**

Accept a GrayImage and inverse run-length encode it. Check to make sure we don’t write past unallocated memory.

LOCATION: SDF image library

DOMAIN: **`SDF`** **(DERIVED FROM: `SDFStar`)**

VERSION: 1.10 (11/25/92)

AUTHOR: *Paul Haskell*

INPUTS: *input* (message)

OUTPUTS: *outData* (message)

STATES: *meanVal* (IntState): Center value for thresholding.
Default = 0

DESCRIPTION:

SEE ALSO: RunLenImage.

NAME: `RunLenImage`

Accept a GrayImage and run-length encode it. All values less than "thresh" from "meanVal" are set to "meanVal" to help improve compression.

Runlengths are coded with a start symbol of "meanVal" and then a run-length between 1 and 255. Runs longer than 255 must be coded in separate pieces.

LOCATION: SDF image library

DOMAIN: `SDF` **(DERIVED FROM:** `SDFStar`)

VERSION: 1.12 (12/8/92)

AUTHOR: *Paul Haskell*

INPUTS: *input* (message)

OUTPUTS: *outData* (message)
compression (float)

STATES: *Thresh* (IntState): Threshold for run-length coding.
Default = 0
meanVal (IntState): Center value for thresholding.
Default = 0

DESCRIPTION:

NAME: `Sgn`

This star computes the signum of its input. The output is +- 1. Note that 0.0 maps into 1.

LOCATION: SDF main library

DOMAIN: `SDF` **(DERIVED FROM:** `SDFStar`)

VERSION: 2.6 (11/25/92)

AUTHOR: *E. A. Lee*

INPUTS: *input* (float)

OUTPUTS: *output* (int)

DESCRIPTION:

NAME: **`singen`**

Generate a sine wave with frequency "frequency" (given "sample_rate").

LOCATION: SDF sources palette (galaxy)

DOMAIN: **`SDF` (DERIVED FROM: `SDFGalaxy`)**

VERSION: 1.212/17/92 ()

AUTHOR: *E. A. Lee*

OUTPUTS: *out* (float)

STATES: *sample_rate* (FloatState): The sample rate, for reference to the frequency.
Default = 2*PI
frequency (FloatState): The frequency, relative to the sample rate.
Default = PI/50

DESCRIPTION:

The sample_rate parameter is used only locally to determine how to construct the output sinusoid. The frequency is given relative to the sample rate.

SEE ALSO: Sin

NAME: **`Sin`**

This star computes the sine of its input, in radians.

LOCATION: SDF main library

DOMAIN: **`SDF` (DERIVED FROM: `SDFStar`)**

VERSION: 1.6 (11/25/92)

AUTHOR: *J. T. Buck*

INPUTS: *input* (float)

OUTPUTS: *output* (float)

NAME: **`Sqrt`**

This star computes the square root of its input.

LOCATION: SDF main library

DOMAIN: **`SDF` (DERIVED FROM: `SDFStar`)**

VERSION: 1.6 (11/25/92)

AUTHOR: *J. T. Buck*

INPUTS: *input* (float)

OUTPUTS: *output* (float)

DESCRIPTION:

NAME: **`SubCx`**

Output the "pos" input minus the "neg" input, a complex value.

LOCATION: SDF main library

DOMAIN: **`SDF`** **(DERIVED FROM: `SDFStar`)**

VERSION: 1.3 (11/25/92)

AUTHOR: *J. Buck*

INPUTS: *pos* (complex)
neg (complex)

OUTPUTS: *output* (complex)

NAME: **`Sub`**

Output the "pos" input minus all "neg" inputs.

LOCATION: SDF main library

DOMAIN: **`SDF`** **(DERIVED FROM: `SDFStar`)**

VERSION: 2.6 (11/25/92)

AUTHOR: *E. A. Lee*

INPUTS: *pos* (float)
neg (multiple), (float)

OUTPUTS: *output* (float)

NAME: **`TableCx`**

Implements a complex-valued lookup table. The "values" state contains the values to output; its first element is element zero. An error occurs if an out of bounds value is received.

LOCATION: SDF main library

DOMAIN: **`SDF`** **(DERIVED FROM: `SDFStar`)**

VERSION: 1.4 (11/25/92)

AUTHOR: *J. T. Buck*

INPUTS: *input* (int)

OUTPUTS: *output* (complex)

STATES: *values* (ComplexArrayState): "table of values to output"
Default = (1, 0) (0, 1) (-1, 0) (0, -1)

DESCRIPTION:

NAME: `TableInt`

Implements a integer-valued lookup table. The "values" state contains the values to output; its first element is element zero. An error occurs if an out of bounds value is received.

LOCATION: SDF main library

DOMAIN: `SDF` **(DERIVED FROM:** `SDFStar`)

VERSION: 1.4 (11/25/92)

AUTHOR: *J. T. Buck*

INPUTS: *input* (int)

OUTPUTS: *output* (int)

STATES: *values* (IntArrayState): "table of values to output"
Default = -1 1

DESCRIPTION:

NAME: `Table`

Implements a real-valued lookup table. The "values" state contains the values to output; its first element is element zero. An error occurs if an out of bounds value is received.

LOCATION: SDF main library

DOMAIN: `SDF` **(DERIVED FROM:** `SDFStar`)

VERSION: 1.5 (11/25/92)

AUTHOR: *J. T. Buck*

INPUTS: *input* (int)

OUTPUTS: *output* (float)

STATES: *values* (FloatArrayState): "table of values to output"
Default = -1 1

DESCRIPTION:

NAME: `telephoneChannel`

Simulate impairments commonly found on a telephone channel, including additive Gaussian noise, linear and nonlinear distortion, frequency offset, and phase jitter.

LOCATION: SDF comm palette (galaxy)

DOMAIN: `SDF` **(DERIVED FROM:** `SDFGalaxy`)

VERSION: 1.212/17/92 ()

AUTHOR: *E. A. Lee*

INPUTS: *in* (float)

OUTPUTS: *out* (float)

STATES: *linearDistortionTaps* (FloatArray): The taps of an FIR filter emulating the linear distortion on the telephone channel. The default value is a null filter that imposes no distortion at all.
Default = 1.0

noise (FloatState): The multiplier applied to the Gaussian noise source, which has a variance per sample of 1.0.
Default = 0.0

phaseJitterFrequency_Hz (FloatState): The phase jitter frequency in hertz (assuming a sample rate of 8 kHz).
Default = 0.0

phaseJitterAmplitude_Deg (FloatState): The phase jitter amplitude (peak) in degrees.
Default = 0.0

frequencyOffset_Hz (FloatState): The frequency offset in hertz (assuming a sample rate of 8 kHz).
Default = 0.0

secondHarmonic (FloatState): The gain applied to second harmonic distortion (gotten by squaring the signal).
Default = 0.0

thirdHarmonic (FloatState): The gain applied to third harmonic distortion (gotten by cubing signal).
Default = 0.0

DESCRIPTION:

This galaxy first applies an FIR filter to the input, then runs it through the **`freqPhase`** galaxy to introduce controlled amounts of frequency offset and phase jitter. Gaussian white noise is added to the result, as is the output of the **`nonLinearDistortion`** galaxy, which introduces non-linear distortion. All parameters are set assuming a sample rate of 8 kHz.

SEE ALSO: freqPhase nonLinearDistortion

NAME: `Thresh`

NOTE: THIS STAR SHOULD BE REPLACED BY Quantizer. Compares input values to "threshold" (default 0.5). Output is 0 if input <= threshold, otherwise it is 1.

LOCATION: SDF main library

DOMAIN: **`SDF` (DERIVED FROM: `SDFStar`)**

VERSION: 1.7 (11/25/92)

AUTHOR: *D. G. Messerschmitt*

INPUTS: *input* (float)

OUTPUTS: *output* (int)

STATES: *threshold* (FloatState): Threshold applied to input.
Default = 0.5

NAME: `Trainer`

Passes the "train" input to the output for the first "trainLength" samples, then passes the "decision" input to the output. Designed for use in decision feedback equalizers, but can be used for other purposes.

LOCATION: SDF main library

DOMAIN: **`SDF` (DERIVED FROM: `SDFStar`)**

VERSION: 1.4 (12/8/92)

AUTHOR: *J. T. Buck*

INPUTS: *train* (anytype)
decision (ANYTYPE)

OUTPUTS: *output* (ANYTYPE)

STATES: *trainLength* (IntState): Number of training samples to use
Default = 100

DESCRIPTION:

NAME: `Unwrap`

A very crude phase unwrapper. It basically assumes that the phase never changes by more than PI in one iteration, and that we "catch" all phase transitions. It also assumes that the input is in the [-PI,PI] range.

LOCATION: SDF dsp library

DOMAIN: `SDF` **(DERIVED FROM:** `SDFStar`)

VERSION: 2.10 (11/25/92)

AUTHOR: *J. T. Buck*

INPUTS: *input* (float)

OUTPUTS: *output* (float)

STATES: *outPhase* (FloatState): Current phase of the star.
Default = 0.0
prevPhase (FloatState): Previous phase input value.
Default = 0.0

DESCRIPTION:

NAME: `UpSample`

Upsample by a factor (default 2), filling with fill (default 0.0). The "phase" tells where to put the sample in an output block. The default is to output it first (phase = 0). The maximum phase is "factor" - 1.

LOCATION: SDF main library

DOMAIN: `SDF` **(DERIVED FROM:** `SDFStar`)

VERSION: 2.6 (12/8/92)

AUTHOR: *J. T. Buck*

INPUTS: *input* (ANYTYPE)

OUTPUTS: *output* (ANYTYPE)

STATES: *factor* (IntState): Number of samples produced.
Default = 2
phase (IntState): Where to put the input in the output block.
Default = 0
fill (FloatState): Value to fill the output block.
Default = 0.0

NAME: `Waterfall`

Plots a series of traces in the style of a "waterfall" plot. This is a type of three-dimensional plot used to show the evolution of signals or spectra. Optionally, each plot can be made opaque, so that lines that would appear behind the plot are eliminated. The star is derived from Xgraph.

LOCATION: SDF main library

DOMAIN: `SDF` **(DERIVED FROM:** `SDFXgraph`)

VERSION: 1.5 (12/8/92)

AUTHOR: *E. A. Lee*

STATES: *traceLength* (IntState): Number of samples per trace.
Default = 128
xUnits (FloatState): Number of horizontal units per input sample.
Default = 1.0
xInit (FloatState): Horizontal value of the first input sample.
Default = 0.0
xOffset (IntState): Amount of right shift of each successive trace, in number of samples.
Default = 4
yOffset (FloatState): Amount of upward shift of each successive trace, in vertical units.
Default = 4.0
showHiddenLines (IntState): Turns on or off hidden-line elimination.
Default = NO
showZeroPlane (IntState): Turns on or off display of the plane where y=0.
Default = YES

SEE ALSO: Xgraph, XMgraph, XYgraph, xgraph, Xhistogram and timeVarSpec.

NAME: `WaveFormCx`

Output a complex waveform as specified by the array state "value" (default "(1,0) (-1,0)"). To halt the simulation after exhausting the data, set "haltAtEnd" to YES. Otherwise, to get a periodic waveform, set "periodic" to YES. Then the value list will be cyclically repeated. If "periodic" is not YES, and "haltAtEnd" is NO, then the value list is output only once, and (0,0) values are output subsequently. This star may be used to read a file by simply setting "value" to something of the form "< filename".

LOCATION: SDF main library

DOMAIN: `SDF` **(DERIVED FROM:** `SDFStar`)

VERSION: 1.6 (12/8/92)

AUTHOR: *J. T. Buck and E. A. Lee*

OUTPUTS: *output* (complex)

STATES: *value* (ComplexArrayState): One period of the output waveform.
Default = (1,0) (-1,0)
haltAtEnd (IntState): Halt the run at the end of the given data.
Default = NO
periodic (IntState): Output is periodic if "YES" (nonzero).
Default = YES

DESCRIPTION:

Since this star can be used to read a complex waveform from a file, there is no other star dedicated to this purpose.

NAME: `WaveForm`

Output a waveform as specified by the array state "value" (default "1 -1"). You can get periodic signals with any period, and can halt a simulation at the end of the given waveform. The following table summarizes the capabilities:

haltAtEnd	periodic	period	operation
NO	YES	0	The period is the length of the waveform
NO	YES	N>0	The period is N
NO	NO	anything	Output the waveform once, then zeros
YES	anything	anything	Stop after outputting the waveform once

The first line of the table gives the default settings.

LOCATION: SDF main library

DOMAIN: `SDF` **(DERIVED FROM:** `SDFStar`)

VERSION: 2.12 (12/17/92)

AUTHOR: *J. T. Buck*

OUTPUTS: *output* (float)

STATES: *value* (FloatArrayState): One period of the output waveform.
Default = 1 -1
haltAtEnd (IntState): Halt the run at the end of the given data.
Default = NO
periodic (IntState): Output is periodic if "YES" (nonzero).
Default = YES
period (IntState): If greater than zero, gives the period of the waveform.
Default = 0

DESCRIPTION:

This star may be used to read a file by simply setting "value" to something of the form "< filename". The file will be read completely and its contents stored in an array. The size of the array is currently limited to 20,000 samples. To read longer files, use the **`ReadFile`** star. This latter star reads one sample at a time, and hence also uses less storage.

NAME: `Window`

Generates standard window functions: Rectangle, Hanning, Hamming, Blackman, and Steep-Blackman.

LOCATION: SDF dsp library

DOMAIN: `SDF` **(DERIVED FROM:** `SDFStar`)

VERSION: 1.9 (12/8/92)

AUTHOR: *Kennard White*

OUTPUTS: *output* (float)

STATES: *name* (StringState): Name of the window function to generate.
Default = Hanning
length (IntState): Length of the window function to produce.
Default = 256
period (IntState): Period of the output. Period 0 implies "length" period, and negative period is non-periodic (single cycle).
Default = 0

DESCRIPTION:

This star produces on its output values that are samples of a standard windowing function. The window function to be sampled is determined by the *name* string parameter. Possible values are: **Rectangle**, **Bartlett**, **Hanning**, **Hamming**, **Blackman**, and **SteepBlackman**. Upper and lower case characters in the names are equivalent.

The parameter *length* is the length of the window to produce. Note that most windows functions have zero value at the first and last sample. The parameter *period* specifies the period of the output signal: the window will be zero-padded if required. A *period* of 0 means a period equal to *length*. A negative period will produce only one window, and then outputs zero for all later samples. A period of less than window length will be equivalent to a period of window length (i.e, period=0).

One period of samples are produced on every firing.

NAME: `Xgraph`

Generate a plot with the xgraph program.

LOCATION: SDF main library

DOMAIN: `SDF` **(DERIVED FROM:** `SDFStar`)

VERSION: 1.12 (12/8/92)

AUTHOR: *J. T. Buck*

INPUTS: *input* (anytype)

STATES: *title* (StringState): Title for the plot.
Default = X graph
saveFile (StringState): File to save the input to the xgraph program.

Default =
options (StringState): Command line options for xgraph.
Default =
ignore (IntState): Number of initial values to ignore.
Default = 0
xUnits (FloatState): For labeling, horizontal increment between samples.
Default = 1.0
xInit (FloatState): For labeling, horizontal value of the first sample.
Default = 0.0

DESCRIPTION:

The input signal is plotted using the *xgraph* program. This program must be in your path, or this star will not work! The *title* parameter specifies a title for the plot. The *saveFile* parameter optionally specifies a file for storing the data in a syntax acceptable to xgraph. A null string prevents any such storage. The *options* string is passed directly to the xgraph program as command-line options. See the manual section describing xgraph for a complete explanation of the options.

NAME: `Xhistogram`

Generate a histogram with the xgraph program. 'binWidth' determines the bin width. 'options' passes extra options to xgraph.

LOCATION: SDF main library

DOMAIN: `SDF` **(DERIVED FROM:** `SDFStar`)

VERSION: 2.8 (12/8/92)

AUTHOR: *E. A. Lee*

INPUTS: *input* (anytype)

STATES: *title* (StringState): Title for the plot.
Default = Xhistogram
saveFile (StringState): File to save input.
Default =
binWidth (FloatState): Width of bins for histogram.
Default = 1.0
options (StringState): Extra Command line options for xgraph.
Default =

DESCRIPTION:

Creates a histogram with the xgraph program. It is assumed that "xgraph" is on your path, or this will not work! The *binWidth* parameter specifies how wide histogram bin will be. The number of bins is determined automatically from the input data.

By default, the xgraph program gets the options ‘‘-bar -nl -brw *halfw*’’ where *halfw* is half the bin width.

NAME: **`XMgraph`**

Generate a multi-signal plot with the xgraph program.

LOCATION: SDF main library

DOMAIN: **`SDF`** **(DERIVED FROM: `SDFStar`)**

VERSION: 2.11 (12/8/92)

AUTHOR: *J. T. Buck and E. A. Lee*

INPUTS: *input* (multiple), (float)

STATES: *title* (StringState): Title for the plot.
Default = X graph
saveFile (StringState): File name for saving plottable data.
Default =
options (StringState): Command line options for the xgraph program.
Default =
ignore (IntState): Number of initial values to ignore.
Default = 0
xUnits (FloatState): For labeling, horizontal increment between samples.
Default = 1.0
xInit (FloatState): For labeling, horizontal value of the first sample.
Default = 0.0

DESCRIPTION:

The input signal is plotted using the *xgraph* program. This program must be in your path, or this star will not work! The *title* parameter specifies a title for the plot. The *saveFile* parameter optionally specifies a file for storing the data in a syntax acceptable to xgraph. A null string prevents any such storage. The *options* string is passed directly to the xgraph program as command-line options. See the manual section describing xgraph for a complete explanation of the options.

SEE ALSO: Xgraph, xgraph, XYgraph and Xhistogram.

NAME: **`Xscope`**

Generate a multi-trace plot with the xgraph program.

LOCATION: SDF main library

DOMAIN: **`SDF`** **(DERIVED FROM: `SDFXgraph`)**

VERSION: 2.6 (11/25/92)

AUTHOR: *J. T. Buck*

STATES: *traceLength* (IntState): Number of samples per trace. If 0, only one trace.
Default = 0

DESCRIPTION:

This star is an enhanced version of Xgraph. It is identical except that it can plot multiple traces, like an oscilloscope. As for Xgraph, the *title* parameter specifies a title for the plot. The *saveFile* parameter optionally specifies a file for storing the data in a syntax acceptable to xgraph. A null string prevents any such storage. The *options* string is passed directly to the xgraph program as command-line options. See the manual section describing xgraph for a complete explanation of the options.

Multiple traces may be plotted by setting the *traceLength* state to a nonzero value. In this case, a new plot (starting at x value zero) is started every *traceLength* samples. The first *ignore* samples are not plotted; this is useful for letting transients die away.

NAME: `XYgraph`

Generates an X-Y plot with the xgraph program. The X data is on "xInput" and the Y data is on "input".

LOCATION: SDF main library

DOMAIN: `SDF` **(DERIVED FROM:** `SDFXgraph`)

VERSION: 1.8 (11/25/92)

AUTHOR: *J. T. Buck*

INPUTS: *xInput* (anytype)

DESCRIPTION:

The input signal is plotted using the *xgraph* program, with one input interpreted as the x-axis data, and the other input as y-axis data.

SEE ALSO: Xgraph, XMgraph, xgraph and Xhistogram.

NAME: `YUVToRGB`

Read three GrayImages that describe a color image in YUV format and output three GrayImages that describe an image in RGB format.

LOCATION: SDF image library

DOMAIN: `SDF` **(DERIVED FROM:** `SDFStar`)

VERSION: 1.8 (11/25/92)

AUTHOR: *Sun-Inn Shih*

INPUTS: *input1* (message)
input2 (message)
input3 (message)

OUTPUTS: *output1* (message)
output2 (message)
output3 (message)

DESCRIPTION:

NAME: **`ZigZagImageInv`**

This star inverse zig-zag scans a DCTImage.

LOCATION: SDF image palette

DOMAIN: **`SDF`** **(DERIVED FROM: `SDFStar`)**

VERSION: 1.7 (11/25/92)

AUTHOR: *Paul Haskell*

INPUTS: *inport* (message)

OUTPUTS: *outport* (message)

SEE ALSO: ZigZagImage.

NAME: **`ZigZagImage`**

This star zig-zag scans a DCTImage and outputs the result. This is useful before quantization.

LOCATION: SDF image palette

DOMAIN: **`SDF`** **(DERIVED FROM: `SDFStar`)**

VERSION: 1.8 (11/25/92)

AUTHOR: *Paul Haskell*

INPUTS: *inport* (message)

OUTPUTS: *outport* (message)

SDF Demos

This section contains explanations of the demo schematics included in the SDF distribution.

NAME: `KSchord`

Simulation of plucked strings using the Karplus-Strong algorithm

LOCATION: ˜ptolemy/src/domains/sdf/demo

DOMAIN: `SDF` **(DERIVED FROM:** `Universe`)

VERSION: 1.3 (12/17/92)

AUTHOR: *Joseph T. Buck*

DESCRIPTION:

This demo models the plucking of three strings; each string is modeled by an instance of the galaxy karplusStrongGal.

The Karplus-Strong algorithm models a string as a tapped delay line. The initial state of the string is determined by the IIDGaussian source and is random. The FIR filter in the feedback loop is a lowpass filter, so high-frequency energy is selectively reduced.

The initial burst yields the fairly broad spectrum characteristic of the onset of a complex sound, with high frequencies predominant. Relative to the fundamental and second harmonic, the higher frequencies die down fairly rapidly, so that the underlying lower tones prevail to the end.

REFERENCES

[1] Richard Moore, *Elements of Computer Music*, Prentice Hall, pp. 278-291, 1990

NAME: `adaptFilter`

A demo of the LMS adaptive filter star

LOCATION: ˜ptolemy/src/domains/sdf/demo

DOMAIN: `SDF` **(DERIVED FROM:** `Universe`)

VERSION: 1.4 (October 15, 1990)

AUTHOR: *Edward A. Lee*

DESCRIPTION:

A Gaussian white noise signal serves as input to an FIR filter and an adaptive filter using the LMS algorithm. The output of the two filters and the error are displayed after the run. By examining the parameters of the two filters, you can find the filenames for the FIR filter coefficients and the initial tap values. The final tap values can be saved by specifying a saveTapsFile in the adaptive filter star.

SEE ALSO: IIDGaussian, LMS, FIR.

NAME: **`allPole`**

A universe showing two ways to implement an all-pole filter.

LOCATION: ~ptolemy/src/domains/sdf/demo

DOMAIN: **`SDF`** **(DERIVED FROM: `Universe`)**

VERSION: 1.4"92/12/17" ()

AUTHOR: *E. A. Lee*

DESCRIPTION:

A noise signal is fed into two realizations of an all pole filter. The lower one uses an **`FIR`** filter in a feedback loop. The upper one uses the **`BlockAllPole`** star. The coefficients for the **`BlockAllPole`** star are supplied by the **`WaveForm`** star, and are identical to the coefficients of the FIR filter. The output of the two filters is identical. The number of samples run is determined in part by the *blockSize* parameter of the **`BlockAllPole`** star.

SEE ALSO: BlockAllPole, FIR

NAME: **`analytic`**

Analytic sample rate conversion.

LOCATION: ~ptolemy/src/domains/sdf/demo

DOMAIN: **`SDF`** **(DERIVED FROM: `Universe`)**

VERSION: 1.5 (October 15, 1990)

AUTHOR: *Edward A. Lee*

DESCRIPTION:

This system uses a ComplexFIR filter to reduce the sample rate of a sinusoid by a factor of 8/5, and at the same time produce a complex, analytic, signal. The magnitude spectrum of the original sinusoid and the filtered sinusoid are both displayed. Note the shift in the peak of the spectrum due to the sample-rate conversion. Also note that the symmetry of the original spectrum is lost in the filtered spectrum.

SEE ALSO: ComplexFIR.

NAME: **`askXmit`**

Synthesize an amplitude-shift keyed (ASK) signal

LOCATION: ˜ptolemy/src/domains/sdf/demo

DOMAIN: `SDF` **(DERIVED FROM:** `Universe`)

VERSION: 1.5 (12/17/92)

AUTHOR: *E. A. Lee*

DESCRIPTION:

An amplitude-shift keyed (ASK) signal is generated by the "ask" galaxy on the left. The modulation format is a simple, baseband, binary-antipodal signal with a 100% excess bandwidth raised cosine pulse. The sample rate is eight times the baud rate.

NAME: `broken`

An example of an inconsistent SDF system

LOCATION: ˜ptolemy/src/domains/sdf/demo

DOMAIN: `SDF` **(DERIVED FROM:** `Universe`)

VERSION: 1.4 (12/17/92)

AUTHOR: *J. Buck*

DESCRIPTION:

This is an example of an SDF universe that cannot be run because of sample rate inconsistencies. The FloatSum star requires a single input on both input arcs, but because of the UpSample star, its input arcs have different sample rates. Because of this, generating an SDF schedule is impossible.

NAME: `butterfly`

The butterfly curves.

LOCATION: ˜ptolemy/src/domains/sdf/demo

DOMAIN: `SDF` **(DERIVED FROM:** `Universe`)

VERSION: 1.4 (12/17/92)

AUTHOR: *E. A. Lee*

DESCRIPTION:

This system computes and plots a curve known as the butterfly curve [1]. In polar form, the magnitude r is given as a function of the phase θ by the equation

$$r = e^{\cos(\theta)} - 2\cos(4\theta) + \sin^5(\theta/12) .$$

REFERENCE

[1] T. Fay, "The Butterfly Curve," *American Math. Monthly*, 96(5), pp. 442-443.

SEE ALSO: XYgraph.

NAME: **cep**

Stabilization of IIR filters using the cepstrum

LOCATION: ˜ptolemy/src/domains/sdf/demo

DOMAIN: **SDF (DERIVED FROM: Universe)**

VERSION: 1.5 (12/17/92)

AUTHOR: *Joseph T. Buck*

DESCRIPTION:

Given a filter

$$H(z) = \frac{\sum_{n=0}^{q} b(n) z^{-n}}{\sum_{n=0}^{p} a(n) z^{-n}} = \frac{B(z)}{A(z)} ,$$

this demo serves as a tool to determine whether the filter is stable. If the filter is not stable, we wish to find a denominator $A_s(z)$ that is causal and stable and is equal to $|A(z)|$ on the unit circle. The filter $H_s(z) = B(z)/A_s(z)$ will therefore evaluate to the same values as $H(z)$ on the unit circle. Hence, in some sense, it has the frequency response that the unstable filter should have had.

This problem depends only on the $a(n)$ (we assume that roots of $A(z)$ and $B(z)$ do not cancel, so only the parameters $a(n)$ are provided as inputs. We have to find the minimum phase signal $a_s(n)$ that has the same magnitude spectrum as $a(n)$.

The demo has a "universe parameter", *ncoefs*. The parameter *ncoefs* should be set greater than or equal to the number of coefficients in $a(n)$ (including the 0'th coefficient). Since 256-point FFT's are used, for good results *ncoefs* should be a good deal smaller than this.

The input coefficients are supplied as the *value* parameter of the WaveForm star on the far left. To read the parameters from a file instead, give "< filename" as the parameter value. The values are the coefficients of $a(n)$, starting with $a(0)$.

The cepstrum of the autocorrelation function is computed and plotted. This signal is multiplied by a window function. This window function should have value 1/2 for $n = 0$, unity for $n > 0$, and zero for $n < 0$, where n is the time index; however, the cepstrum "wraps around" (negative time appears above positive time because of the periodicity of the DFT) so the application must compensate for this. The resulting signal is the cepstrum of a minimum phase signal $\hat{a}_{s(n)}$, and it can be shown that the autocorrelation of $a_{s(n)}$ is the same as the autocorrelation of $a(n)$. So we convert it from the cepstrum domain back to the time domain to obtain our result.

The original signal is plotted in red; the modified signal is plotted in green. If the original signal was minimum phase (equivalently, if the original filter was stable) you'll see only one plot.

REFERENCES

[1] J. Lim, *Two-dimensional signal and image processing*, Prentice-Hall, Englewood Cliffs, NJ, 1990, pp. 298-300

SEE ALSO: WaveForm, ComplexFFT

NAME: **`chaos`**

A simple demonstration of chaos.

LOCATION: ˜ptolemy/src/domains/sdf/demo

DOMAIN: **`SDF`** **(DERIVED FROM: `Universe`)**

VERSION: 1.312/17/92 ()

AUTHOR: *E. A. Lee*

DESCRIPTION:

This universe generates the so called Henon map, using the following nonlinear recurrence:

$$x(n+1) = 1 + y(n) - 1.4x^2(n)$$

$$y(n+1) = 0.3x(n)\ .$$

Initialized with zero, this system exhibits chaotic behavior.

NAME: **`chirpplay`**

Chirp generator that plays on the workstation speaker.

LOCATION: ˜ptolemy/src/domains/sdf/demo

DOMAIN: **`SDF`** **(DERIVED FROM: `Universe`)**

VERSION: 1.2 (February 8, 1991)

AUTHOR: *E. A. Lee*

DESCRIPTION:

This universe generates a chirp (a sinusoid with slowly increasing frequency) and plays it over the workstation speaker. It does this by writing its output to a temporary file and invoking the "play" program on that file. On a sparcstation equipped with a speaker, this will produce sound. This will also work on any other workstation if a compatible "play" program is in the users path.

NAME: **`ColorImage`**

A universe that displays a color image.

LOCATION: ~ptolemy/src/domains/sdf/demo

DOMAIN: **`SDF`** **(DERIVED FROM: `Universe`)**

VERSION: 1.5 (August 19, 1991)

AUTHOR: *P. E. Haskell*

DESCRIPTION:

This universe reads a color image in RGB format from a file. The image is transformed into YUV color format, and then back to RGB format. Finally, the RGB image is displayed.

SEE ALSO: DisplayRgb ReadRgb Rgb2Yuv Yuv2Rgb

NAME: **`CompareMedian`**

A universe that median filters an image.

LOCATION: ~ptolemy/src/domains/sdf/demo

DOMAIN: **`SDF`** **(DERIVED FROM: `Universe`)**

VERSION: 1.4 (August 19, 1991)

AUTHOR: *P. E. Haskell*

DESCRIPTION:

This universe reads an image from a file and median filters the image. The original image, median filtered image, and difference between the two are all displayed.

SEE ALSO: DisplayImage ReadImage MedianImage

NAME: **`complexExponential`**

Generate and plot a complex exponential

LOCATION: ~ptolemy/src/domains/sdf/demo

DOMAIN: **`SDF`** **(DERIVED FROM: `Universe`)**

VERSION: 1.212/17/92 ()

AUTHOR: *E. A. Lee*

DESCRIPTION:

A complex exponential with frequency $\pi/50$ (radians per second) is generated and plotted. The sampling frequency is normalized to 2π radians per second.

SEE ALSO: expgen CxReal XMgraph

NAME: **`DctImage`**

A universe that does discrete cosine transform (DCT) coding.

LOCATION: ˜ptolemy/src/domains/sdf/demo

DOMAIN: **`SDF`** **(DERIVED FROM: `Universe`)**

VERSION: 1.4 (August 19, 1991)

AUTHOR: *P. E. Haskell*

DESCRIPTION:

This universe reads an image from a file and performs DCT compression on the image. Next, the transform is inverted, and the result is displayed.

SEE ALSO: DisplayImage ReadImage Dct DctInv

NAME: **`dft`**

Compute a discrete Fourier transform of a finite signal

LOCATION: ˜ptolemy/src/domains/sdf/utilities

DOMAIN: **`SDF`** **(DERIVED FROM: `Universe`)**

VERSION: 1.712/17/92 ()

AUTHOR: *Edward A. Lee*

DESCRIPTION:

This demo computes a DFT (using the FFT star) of a finite signal. It can be used, for example, to plot the frequency response of an FIR filter given its impulse response. This system is so useful that it is included in the Utilities menu of pigi. The magnitude (in dB) and phase (unwrapped) are plotted.

The time-domain signal is read from a file using the WaveForm star. The filename is given by the universe parameter "signal". Notice how the syntax "< filename" is used to specify that the signal should come from a file. Alternatively, the signal could be specified directly. For example, a square pulse with with 20 could be specified using the syntax "1.0 [20] 0.0" for the value of "signal".

Immediately after the signal is read it is multiplied by another signal consisting of alternating + and - one. This trick shifts the d.c. component of the signal to the Nyquist frequency, and hence is an efficient way to get a spectral plot with the d.c. component in the center of the plot. Without this multiplication, d.c. would be at the left, and the sampling frequency at the right. With it, the output plot is centered at d.c. (frequency 0.0), and ranges from -PI to PI. The proper labeling of the x axis is achieved by using the FloatRamp star to generate the X input of the XYgraph stars.

Note that the conversion from the FLOAT output of the FIR filter to the COMPLEX input of the ComplexFFT is implicit. The default time-domain signal is an impulse response of an equiripple lowpass filter designed using the optfir command.

SEE ALSO: FFT, ComplexFFT, optfir, Unwrap, WaveForm, XYgraph.

NAME: `doppler`

Doppler shift

LOCATION: ~ptolemy/src/domains/sdf/demo

DOMAIN: `SDF` **(DERIVED FROM:** `Universe`)

VERSION: 1.5 (12/17/92)

AUTHOR: *Edward A. Lee*

DESCRIPTION:

A sine wave of frequency 2*PI/40 radians is subjected to four successive amounts of doppler shift. The first 100 samples are not shifted at all (in the first 64 samples transients are dying out). The next 160 samples are shifted up in frequency by 2*PI/160 radians (so that there is one extra cycle for every four cycles of the original sinusoid). The next 160 samples are shifted down in frequency by 2*PI/160 radians (so that there is one fewer cycle for every four cycles of the original sinusoid). The remaining samples are not shifted at all.

The doppler shift is accomplished by the phaseShift galaxy, which forms an analytic signal (using a Hilbert transform) which modulates a complex exponential. The input is assumed to be real, and the output produced is real. The frequency shift produced by the phaseShift galaxy is the slope of the phase control input. This input is designed to be flat for the first 100 samples, rising for the next 160 samples, falling for the next 160 samples, and flat thereafter. To get this waveform, the WaveForm star is combined with an Integrator. The outputs of the WaveForm star, therefore, give the frequency offset in radians.

Note that because the Hilbert filter in the phaseShift galaxy is a 64-tap, linear-phase FIR filter, there is a transient of 64 samples while the delay line of the filter fills. Since the group-delay of this filter is 32 samples, the original sinusoid is delayed by 32 samples before being displayed.

SEE ALSO: phaseShift.

NAME: `DpcmImage`

A universe that does differential pulse code modulation (DPCM).

LOCATION: ~ptolemy/src/domains/sdf/demo

DOMAIN: `SDF` **(DERIVED FROM:** `Universe`)

VERSION: 1.7 (August 19, 1991)

AUTHOR: *P. E. Haskell*

DESCRIPTION:

This universe reads images from a file and performs DPCM compression on the image sequence. The difference-coded image sequence is run-length encoded. The coded image, final image, and difference between the original and final images are all displayed.

The DisplayVideo star needs programs from the Utah Raster Toolkit to be in your $path variable to work. These programs are not included with Ptolemy. The manual page for the DisplayVideo star tells how to get the Utah Raster Toolkit for free.

SEE ALSO: DisplayVideo ReadImage Dpcm RunLen

NAME: **`dtft`**

Demonstrate the DTFT star.

LOCATION: ~ptolemy/src/domains/sdf/demo

DOMAIN: **`SDF`** **(DERIVED FROM: `Universe`)**

VERSION: 1.5 (October 28, 1990)

AUTHOR: *E. A. Lee*

DESCRIPTION:

This demo compares the function of the **`DTFT`** star to that of the **`ComplexFFT`** star. While both stars output samples of the discrete-time Fourier transform of a finite-length signal, the **`FFT`** star can only produce 2^N samples, where N is the integer *order*. By contrast, the **`DTFT`** star can produce any number of samples, without any constraint that the number be a power of two. Furthermore, the FFT samples must be uniformly spaced, and cover the entire spectrum from d.c. to the sampling frequency. The DTFT samples can be spaced at arbitrary intervals, and can cover any subset of the spectrum. An outside signal source (in this case a **`FloatRamp`** star) provides the values of ω at which to sample the DTFT.

In both cases, the magnitude (in dB) and phase are plotted. The DTFT output is plotted using the **`XYgraph`** star, so that the frequency axis of the plot can be properly labeled. The labels are normalized frequency, where 2π is the sampling frequency.

SEE ALSO: ComplexFFT, DTFT, XYgraph.

NAME: **`filterBank`**

Discrete Wavelet Transform Filter Bank

LOCATION: ~ptolemy/src/domains/sdf/demo

DOMAIN: **`SDF`** **(DERIVED FROM: `Universe`)**

VERSION: %I% (November 18, 1992)

AUTHOR: *Alan Peevers*

DESCRIPTION:

This universe implements an 8-level perfect reconstruction filter bank based on a biorthogonal wavelet decomposition. It is inspired by the paper "Wavelets and Filter Banks: Theory and Design", *IEEE Transactions on Signal Processing,* **40(9)**, September, 1992. This is a form of sub-band coder, where the sample rate is decreasing by powers of two, forming a dyadic decomposition of the frequency axis. At each stage of the analysis section, the incoming signal is split into highpass and lowpass components via two FIR filters. These components are then decimated by a factor of two. The decimated high-pass signal proceeds directly to the synthesis filterbank, while the decimated low-pass signal gets sent down to the next level of analysis. The effect is to split the frequency domain into progressively finer bins, with the highest resolution at the lowest frequencies.This can be seen as a binary 'tree' structure, where successive representations have progressively better frequency resolution at the expense of progressively lower time resolution.

The multi-resolution representation at the analysis bank outputs is actually the Discrete Wavelet Transform (DWT) decomposition of the original signal. The DWT is characterized by having this power-of-2 resolution decomposition.

The synthesis filter bank performs the inverse function of the analysis bank. That is, incoming signals from two levels of the analysis section are each upsampled by 2, passed through a different set of FIR filters, and added together, yielding a new signal at twice the component signals' sampling rate. This then gets combined with the next highest rate signal from the analysis bank, and so on, until the original sampling rate is restored.

For specially designed FIR filters, the reconstructed signal will be an exact replica of the input (to within coefficient roundoff error). This property, known as the 'perfect reconstruction' property, has been studied in great detail in the literature. See, for example, "Quadrature Mirror Filter Banks, M-Band Extensions, and Perfect-Reconstruction Techniques", *IEEE ASSP Magazine* , **37(12)**, July, 1987.

All analysis stages are identical, as are all synthesis stages. There are, in all, only 4 unique FIR filters used. The impulse response of the analysis filters form a pair of biorthogonal wavelets. Biorthogonality is not quite as strong as orthogonality, and the reader is referred to Vetterli et al for more details.

anais synthesis block. The source used is a sinewave burst. Note the sharp edges in the output image at the precise onset and offset times of the burst. By typing 'e' in the image window, the color map can be edited to enhance details. The galaxy prescaler has gain and offset parameters which can be adjusted to optimize the dynamic range of the image.

The *sfb* universe is a four-level decomposition using the same filters. It runs much more quickly.

SEE ALSO: sfb

NAME: `fm`

Sound generator using FM modulation.

LOCATION: ~ptolemy/src/domains/sdf/demo

DOMAIN: **`SDF` (DERIVED FROM: `Universe`)**

VERSION: 1.6 (October 15, 1990)

AUTHOR: *E. A. Lee and Tom Parks*

DESCRIPTION:

This universe implements the FM-based sound synthesis system invented by H. M. Chowning and described in "The Synthesis of Complex Audio Spectra by Means of Frequency Modulation", *Journal of the Audio Engineering Society*, **21(7)**, September, 1973. This technique produces a sound that is rich in harmonics but relatively inexpensive to compute.

A **`FloatDC`** star controls the center frequency, relative to 2π, the sampling frequency. The default value, 1.0, produces a spectrum centered at $1/(2\pi)$ or about 0.159 times the sampling frequency.

The galaxy fmGalaxy actually computes the waveform. It has three inputs, *pitch*, which controls the perceived pitch, *index*, which controls the modulation index, and *amp*, which controls the amplitude of the sound. In addition, there are three parameters. The *Nc* parameter multiplies the pitch input to give the carrier frequency of the FM waveform. The *Nm* parameter multiplies the pitch input to give the frequency of the sinusoid that modulates the carrier. The *Imax* parameter multiplies the *index* input. Since this input must be smaller than one in magnitude, the *Imax* parameter controls the maximum modulation index. The modulation index controls the relative strength of the harmonics. For instance, a modulation index of zero will yield a pure tone. For more information, see [1].

REFERENCES

[1] C. Dodge and T. Jerse, *Computer Music: Synthesis, Composition, and Performance*, Schirmer Books, New York, 1985.

NAME: `fmplay`

Sound generator using FM modulation that plays on the workstation speaker.

LOCATION: ~ptolemy/src/domains/sdf/demo

DOMAIN: **`SDF` (DERIVED FROM: `Universe`)**

VERSION: 1.2 (February 8, 1991)

AUTHOR: *E. A. Lee and Tom Parks*

DESCRIPTION:

This universe implements the FM-based sound synthesis system invented by H. M. Chowning and described in "The Synthesis of Complex Audio Spectra by Means of Frequency Modulation", *Journal of the Audio Engineering Society*, **21(7)**, September, 1973. The structure is identical to the *fm* demo system, which plots its output on the screen, but instead of plotting the output on the screen, it writes it to a file and invokes the "play" program on that file. On a sparcstation equipped with a speaker, this will produce sound. This will also work on any other workstation if a compatible "play" program is in the users path.

NAME: **`freqPhaseOffset`**

Impose frequency jitter and phase offset on a sinusoid.

LOCATION: ~ptolemy/src/domains/sdf/demo

DOMAIN: **`SDF`** **(DERIVED FROM: `Universe`)**

VERSION: 1.212/17/92 ()

AUTHOR: *E. A. Lee*

DESCRIPTION:

A 400 Hz sinusoid, sampled at 8kHz, is generated. On the upper path, 15 degrees (peak) phase jitter is imposed on the sinusoid, and the result displayed using overlaid traces, so that the jitter can be easily seen. On the lower path, frequency offset of -200 Hz is imposed on the signal, resulting in a 200 Hz sinusoid. The "freqPhase" galaxy is used to accomplish this.

SEE ALSO: freqPhase Xscope

NAME: **`freqsample`**

Filter design via frequency sampling.

LOCATION: ~ptolemy/src/domains/sdf/demo

DOMAIN: **`SDF`** **(DERIVED FROM: `Universe`)**

VERSION: 1.3 (October 20, 1990)

AUTHOR: *J. Buck and E. A. Lee*

DESCRIPTION:

This system designs filters using the frequency sampling method. The frequency domain specification is given using the **`WaveForm`** star, and using some of the more elaborate features of the parameter interpreter. The filter specification in this demo reads:

1 [6] 0.3904 0 [19] 0.3904 1 [5]

which translates to six samples with magnitude 1.0, one transition sample with magnitude 0.3904, 19 samples with magnitude 0.0, the same transition sample again, and five sample with magnitude 1.0. This specifies a lowpass filter with the cutoff at about 1/3 of the Nyquist frequency. The frequency specification is first modulated by a sequence of alternating ±1. This effectively translates the impulse response so that it is symmetric about the center sample, number 15. Modulation in the frequency domain yields translation in the time domain. It is an interesting experiment to remove this modulation and compare the resulting frequency response.

After this modulation, an inverse FFT gives the impulse response of the filter. Both the real and imaginary parts of this are plotted. Symmetry in the specification will result in the imaginary parts being zero. To compute the frequency response, a forward FFT of much higher order is computed (256, in this case). The plot therefore has enough resolution to see how the frequency response behaves in-between the specified samples. Note that the *size* parameter of the FFT is 32, indicating that only 32 input samples should be read. Of course, as is standard with the frequency sampling method, the magnitude response of the resulting filter exactly passes through the given samples at discrete frequency intervals. This can be easily verified with the plot.

To obtain a filter with a real-valued impulse response, the frequency domain specification must be symmetric, as with the default parameters. To understand the symmetry precisely, note that the frequency domain specification is actually only one cycle of a periodic specification. With this observation, and the fact that the first sample specifies the d.c. response, the symmetry is obvious.

To design a filter of a different order from that in the demo, you must modify the *size* and *order* parameters of the first **`ComplexFFT`** star, and the *size* parameter of the second.

SEE ALSO: WaveForm, ComplexFFT.

NAME: **`gaussian`**

Histogram and autocorrelation of a Gaussian noise source

LOCATION: ~ptolemy/src/domains/sdf/demo

DOMAIN: **`SDF`** **(DERIVED FROM: `Universe`**)

VERSION: 1.5 (October 20, 1990)

AUTHOR: *E. A. Lee*

DESCRIPTION:

This system plots a histogram and estimated autocorrelation of samples generated by an **`IIDGaussian`** star. Note that since the **`Autocor`** star does block processing, this demo should be run for one iteration. The number of Gaussian samples processed will be determined by the *noInputsToAvg* parameter of the autocorrelation block.

SEE ALSO: Autocor, Xhistogram, IIDGaussian.

NAME: **`integrator`**

Demonstrate the features of the integrator star.

LOCATION: ~ptolemy/src/domains/sdf/demo

DOMAIN: **`SDF`** **(DERIVED FROM: `Universe`)**

VERSION: 1.5 (October 15, 1990)

AUTHOR: *Edward A. Lee*

DESCRIPTION:

This system shows the various features of the **`Integrator`** star. The leftmost integrator has limits so that its output is a sawtooth. The **`Quantizer`** star resets the rightmost integrator when the sawtooth is at its peaks. The rightmost integrator also has leakage, so its output signal is bowed. Note that a **`FloatDC`** star is connected to the INT reset input of the leftmost integrator. The type conversion is automatic, converting the FLOAT 0.0 into the INT 0.

SEE ALSO: Integrator.

NAME: **`interp`**

Linear interpolation

LOCATION: ~ptolemy/src/domains/sdf/demo

DOMAIN: **`SDF`** **(DERIVED FROM: `Universe`)**

VERSION: 1.4 (October 15, 1990)

AUTHOR: *Edward A. Lee*

DESCRIPTION:

This system uses an FIR filter to upsample by a factor of 8 and linearly interpolate between samples.

SEE ALSO: FIR.

NAME: **`lattice`**

Demonstrate the use of lattice filters.

LOCATION: ~ptolemy/src/domains/sdf/demo

DOMAIN: **`SDF`** **(DERIVED FROM: `Universe`)**

VERSION: 1.512/17/92 ()

AUTHOR: *E. A. Lee*

DESCRIPTION:

This demo generates an auto-regressive (AR) random process by filtering Gaussian white noise with an all-pole filter with transfer function

$$H(z) = \frac{1}{1 - 2z^{-1} + 1.91z^{-2} - 0.91z^{-3} + 0.205z^{-4}} .$$

The filter is implemented two ways. First, it is impmented with an FIR filter in a feedback loop. Then, in parallel, it is implemented with a recursive lattice filter, using the **`RLattice`** star. The transfer function of the lattice filter is also $H(z)$, so the outputs should be identical. The resulting AR process is then filtered using an FIR lattice (the **`Lattice`** star) with the same coefficients as in the recursive lattice. This has transfer function $H^{-1}(z)$, which whitens the random process. The power spectrum for each process is then estimated using the **`autocorrelation`** galaxy.

SEE ALSO: latticeDesign, Lattice, RLattice.

NAME: **`latticeDesign`**

Use of Levinson-Durbin algorithm to design a lattice filter with a known transfer function.

LOCATION: ~ptolemy/src/domains/sdf/demo

DOMAIN: **`SDF`** **(DERIVED FROM: `Universe`)**

VERSION: 1.14/18/92 ()

AUTHOR: *E. A. Lee*

DESCRIPTION:

This demo generates the impulse response of an all-pole (auto-regressive, AR) filter using an FIR filter in a feedback loop, and the uses the Levinson-Durbin algorithm to design a lattice filter with the same transfer function and the inverse transfer function. The transfer function of the AR filter is

$$H(z) = \frac{1}{1 - 2z^{-1} + 1.91z^{-2} - 0.91z^{-3} + 0.205z^{-4}} .$$

A biased autocorrelation estimate is computed and fed into the **`LevDur`** star. The FIR and lattice predictor coefficient outputs of the LevDur star are plotted. The FIR predictor coefficients should correspond exactly to the negative of all but the first coefficient in the denominator of $H(z)$ above. The lattice predictor coefficients are loaded into the **`BlockLattice`** and **`BlockRLattice`** stars The BlockLattice star therefore has transfer function $H^{-1}(z)$. To verify this, the impulse response of the AR filter is fed into it, producing an impulse at the output. The BlockRLattice star will have transfer function $H(z)$. This is verified by plotting its impulse response together with that of the AR filter. They should be identical, so the two plots should exactly overlap.

SEE ALSO: Autocor, BlockLattice, BlockRLattice, Lattice, lattice, LevDur, RLattice.

NAME: `levinsonDurbin`

Use the Levinson Durbin algorithm to estimate the parameters of an AR process.

LOCATION: ~ptolemy/src/domains/sdf/demo

DOMAIN: `SDF` **(DERIVED FROM:** `Universe`)

VERSION: 1.7 (October 31, 1990)

AUTHOR: *E. A. Lee*

DESCRIPTION:

This demo generates an auto-regressive (AR) random process by filtering Gaussian white noise with an all-pole filter. Then it estimates the autocorrelation of the process and uses the Levinson Durbin algorithm to estimate the parameters of the process. The transfer function of the all-pole filter is:

$$H(z) = \frac{1}{1 + d_1 z^{-1} + d_2 z^{-2}}$$

where in this case $d_1 = -0.9$ and $d_2 = 0.5$. The *lp* output of the **`LevDur`** star is estimates of the negative of these parameters, as can be verified using the plot produced by this demo. These are called the autoregressive or linear predictor parameters. Thus, if $F(z)$ is the Z-transform of the *lp* output of the **`LevDur`** star, then

$$G(z) = 1 + z^{-1} F(z)$$

is an estimate of the whitening filter, or the filter that will convert the filtered noise sequence into white noise.

The reflection coefficients are also plotted, but it is difficult in this demo to verify that they are correct. However, the fact that they become very small after the second one is reassuring because the AR noise is second order. Note that the definition of reflection coefficients is not universal in the literature. The reflection coefficients defined in references [2] and [3] are the negative of the ones generated by the **`LevDur`** star. The ones generated correspond to the definition in most other texts, and correspond to the definition of partial-correlation (PARCOR) coefficients in the statistics literature.

The prediction error power is plotted as a function of the predictor order. The first sample corresponds to the zero-th order predictor, so it is simply the estimate of the power of the input signal. The predictor error power fails to decrease further when the order exceeds two, which again is reassuring because of the order of the input process. The last plot is the autocorrelation estimate used in the Levinson-Durbin algorithm. Note that the center lag is the estimate of the power of the input process, and hence is equal to the zero sample of the prediction error power.

REFERENCES

[1] J. Makhoul, "Linear Prediction: A Tutorial Review", *Proc. IEEE*, Vol. 63, pp. 561-580, Apr. 1975.

[2] S. M. Kay, *Modern Spectral Estimation: Theory & Application*, Prentice-Hall, Englewood Cliffs, NJ, 1988.

[3] S. Haykin, *Modern Filters*, MacMillan Publishing Company, New York, 1989.

SEE ALSO: Autocor, BiQuad, IIDGaussian, LevDur.

NAME: **`linearPrediction`**

Two mechanisms for linear prediction are compared.

LOCATION: ~ptolemy/src/domains/sdf/demo

DOMAIN: **`SDF`** **(DERIVED FROM: `Universe`)**

VERSION: 1.5 (October 31, 1990)

AUTHOR: *E. A. Lee*

DESCRIPTION:

This demo performs linear prediction on a test signal consisting of three sinusoids in colored, Gaussian noise. The first (upper) method uses Burg's algorithm and second (lower) method uses an adaptive filter, with the LMS stochastic gradient adaptation algorithm. The plots that are generated show the original signal, the predicted signal, and the difference between these, or the prediction error.

Burg's method is a block method. The **`Burg`** star collects 128 input samples, from which it estimates the autoregressive (AR) parameters of the input process. These parameters are then loaded into the **`BlockFIR`** star, which filters the input signal, delayed by one, in order to construct the minimum mean-square-error prediction. An initial transient, where the prediction is poor, occurs because the tapped delay line of the **`BlockFIR`** needs to fill up with input samples before accurate predictions can be made. If the system is run for two iterations (two blocks), there will be no transient at the start of the second block because the tapped delay line is full.

The *blockSize* parameter of the **`BlockFIR`** star and the *numInputs* parameter of the **`Burg`** star must be equal. Together they determine how many samples are processed by each iteration of the overall schedule. In this demo, they are both 256, so 256 input samples are processed on each iteration. Because of the relatively large number of samples plotted, you may wish to zoom in on the plots for more detail.

The adaptive filter method is more dynamic than the Burg method. Inputs are examined one-by-one and the current FIR filter coefficients are used to predict the next sample. The difference between the prediction and the actual signal is then fed back to the **`LMS`** star, which adjusts its FIR filter coefficients to try to improve the next prediction. The plots show that that the prediction is not as good as that of the Burg method, and that the initial transient is considerably longer, giving time for the LMS filter to converge. However, the adaptive filter method may perform better on some non-stationary signals, such as those with slowly varying statistics. Furthermore, parameters like filter order, step-size, and the length of the run can be varied to improve the performance.

REFERENCES

[1] J. Makhoul, "Linear Prediction: A Tutorial Review", *Proc. IEEE*, Vol. 63, pp. 561-580, Apr. 1975.

[2] S. M. Kay, *Modern Spectral Estimation: Theory & Application*, Prentice-Hall, Englewood Cliffs, NJ, 1988.

[3] S. Haykin, *Modern Filters*, MacMillan Publishing Company, New York, 1989.

SEE ALSO: Burg, LevDur, levinsonDurbin, LMS, powerSpectrum.

NAME: `MC_DCT`

A universe that does motion compensation and DCT image coding.

LOCATION: ~ptolemy/src/domains/sdf/demo

DOMAIN: `SDF` **(DERIVED FROM:** `Universe`)

VERSION: 1.4 (November 12, 1992)

AUTHOR: *P. E. Haskell*

DESCRIPTION:

This universe reads images from a file and performs motion compensation, the discrete cosine transform (DCT), zig-zag scanning of the DCT coefficients, and run-length coding of the DCT coefficients. The final video sequence is displayed.

The DisplayVideo star needs programs from the Utah Raster Toolkit to be in your $path variable to work. These programs are not included with Ptolemy. The manual page for the DisplayVideo star tells how to get the Utah Raster Toolkit for free.

SEE ALSO: DisplayVideo ReadImage MotionCmp DCTImage DCTImageCode

---

NAME: `MotionComp`

A universe that does motion compensation image coding.

LOCATION: ~ptolemy/src/domains/sdf/demo

DOMAIN: `SDF` **(DERIVED FROM:** `Universe`)

VERSION: 1.4 (August 19, 1991)

AUTHOR: *P. E. Haskell*

DESCRIPTION:

This universe reads images from a file and performs motion compensation on the image sequence. The difference-coded sequence is run-length encoded. The final image is displayed.

SEE ALSO: DisplayImage ReadImage MotionCmp RunLen

NAME: **`multirate`**

A universe that upsamples a signal by a ratio of 5/2.

LOCATION: ˜ptolemy/src/domains/sdf/demo

DOMAIN: **`SDF`** **(DERIVED FROM: `Universe`)**

VERSION: 1.6 (October 15, 1990)

AUTHOR: *E. A. Lee*

DESCRIPTION:

This universe generates a sine wave using the **`singen`** galaxy and feeds it into an **`FIR`** star with parameters set to upsample by a ratio of 5/2 (the lower path). The FIR star internally uses a polyphase filter structure for efficient computation. Along the upper path, a functionally equivalent, but less efficient implementation uses **`UpSample`** and **`DownSample`** stars. Because of properties of synchronous dataflow scheduling, when you run this universe for 25 iterations, you get 50 samples of the input signal and 125 samples (5/2 times as many) of the output signal. The filter in the FIR star is a lowpass with its cutoff frequency at 1/20th of the sample rate. In this case, the sample rate in question is five times the input sample rate, which is equal to twice the output sample rate. The low cutoff conservatively prevents aliasing due to the sample rate conversions. The filter was designed using the optfir program, which can be invoked through the pigi menu.

SEE ALSO: optfir, interp, analytic, FIR.

NAME: **`muxDeMux`**

Demonstrate the Mux and DeMux stars

LOCATION: ˜ptolemy/src/domains/sdf/demo

DOMAIN: **`SDF`** **(DERIVED FROM: `Universe`)**

VERSION: 1.5 (12/17/92)

AUTHOR: *Edward A. Lee*

DESCRIPTION:

This system shows how to use the **`Mux`** and **`DeMux`** stars. The control signal for each is generated by the **`WaveForm`** star. A ramp is demultiplexed, so the first two signals displayed are the two outputs of the demultiplexer. The control signal is chosen to repeat the pattern "0 0 0 1 1 1 2 2 2", so the first three ramp outputs go to the top demultiplexer output, the next three go to the bottom demultiplexer output, and the final three go nowhere. When the value of the control signal exceeds the number of outputs, then all outputs of the demultiplexer produce 0.0. The first control signal is periodic with period 9 and the second is periodic with period 6, so the overall control pattern has period 18.

SEE ALSO: Distributor, Commutator, Switch, Select.

NAME: **`phasedArray`**

Phased array antenna simulation.

LOCATION: ˜ptolemy/src/domains/sdf/demo

DOMAIN: **`SDF`** **(DERIVED FROM: `Universe`)**

VERSION: 1.4 (October 15, 1990)

AUTHOR: *E. A. Lee*

DESCRIPTION:

This demo simulates a plane wave approaching an array of four antennas from angles starting from head on and slowly rotating 360 degrees. The response of the antenna is plotted as a function of direction of arrival in polar form. The outputs of the four antennas are simply added together, so the maximal response occurs when the excitation is incident to the plane of the antennas. A more elaborate system would introduce programmable delays after each antenna element in order to steer the antenna. The magnitude response is measured by computing the complex envelope of the signal. This is done with a phase splitter, which produces an analytic signal. The magnitude of the analytic signal is the complex envelope. The **`Cut`** star is used to discard initial transients.

SEE ALSO: Cut and FIR.

NAME: **`plldemo`**

Simulation of an optical phase-locked loop.

LOCATION: ˜ptolemy/src/domains/sdf/demo

DOMAIN: **`SDF`** **(DERIVED FROM: `Universe`)**

VERSION: 1.212/17/92 ()

AUTHOR: *John R. Barry*

DESCRIPTION:

This is a discrete-time model of a continuous-time system. Specifically, the system is a fourth-power optical phase-locked loop suitable for synchronous detection of heterodyne optical QPSK. Simulation of such a phase-locked loop is hampered by two obstacles: (1) the signals are continuous-time, and (2) the signals are passband, some as high as 200 THz. We hide the passband nature of the signals by representing each signal by its complex-valued baseband envelope. Moreover, we use the samples of each envelope to convert to discrete time. The continuous-time filters were converted to discrete time using the impulse-invariant method. For example, a continuous-time filter with impulse response $h(t)$ is represented by a discrete-time filter h_k, where $h_k = Th(kT)$ and T is the time step. Similarly, a continuous-time integrator is represented by a discrete-time integrator scaled by T. A continuous-time white noise with power-spectral density N_0 is represented by a discrete-time white random process with power-spectral density N_0/T.

There are two sources of noise in an optical phased-lock loop: laser phase noise and shot noise. The laser phase noise is modeled as a Brownian motion process, which is generated by passing a white Gaussian noise process through an integrator. The shot noise in heterodyne detection is of extremely high intensity and thus is accurately-modeled as white Gaussian noise. The phase detector simply raises the QPSK signal to the fourth power, effectively wiping out the QPSK modulation and isolating the phase error. This phase-error estimate is passed through a loop filter to generate a control signal which is fed into the VCO (voltage controlled oscillator).

The first graph plots the input phase (laser phase noise) and output phase versus time. The input phase takes a random walk, so its variance grows linearly with time. The output phase settles near zero degrees, with a standard deviation of about 3 degrees. The standard deviation of the steady-state phase error is a critical performance parameter, and the primary purpose of these simulations is to determine this standard deviation. The last graph plots the estimated standard deviation versus time; for a sufficiently large number of iterations, this estimate will get arbitrarily close to its actual value.

The histogram estimates the probability density function of the phase error, which (at high signal-to-noise ratios) should approach Gaussian for a sufficiently large number of iterations. The performance of the entire QPSK system can be estimated visually by inspecting the in-phase and quadrature eye diagrams.

REFERENCES

[1] J. Barry and J. Kahn, *Carrier Synchronization for Heterodyne Detection of Optical Quadriphase-Shift Keying,* submitted to J. Lightwave Technology, manuscript 1213, February 1992.

SEE ALSO:

NAME: **`powerSpectrum`**

Compare three methods for estimating a power spectrum.

LOCATION: ~ptolemy/src/domains/sdf/demo

DOMAIN: **`SDF`** **(DERIVED FROM: `Universe`)**

VERSION: 1.4 (12/17/92)

AUTHOR: *E. A. Lee*

DESCRIPTION:

This demo generates a random process that is a real-valued version of the test process described in [2] (page 11). It consists of three sinusoids, two of which are closely spaced, and colored Gaussian noise. The power spectrum of this process is estimated using three methods, the periodogram, the autocorrelation method (which uses the Levinson Durbin algorithm), and the Burg method. Since the input process is Gaussian, the latter two methods produce approximate *maximum entropy* spectral estimates. If the autocorrelation of the input process were known exactly, then these methods would produce exact maximum entropy spectra. However, the autocorrelation is estimated from observations of the input, so the two methods yield slightly different results.

The universe parameters are *log2NumInputs*, the log base 2 of the number of input samples to generate, and *order*, the order of the AR model to use in the autocorrelation and Burg's method estimates.

The periodogram method amounts to computing a direct DFT of the observations of the input process. The number of observations used is 512. This type of power spectrum estimate has very high variance, and increasing the number of observations does not reduce the variance. This high variance can be observed in the extreme jaggedness of the output plot.

The autocorrelation method first estimates the autocorrelation of the input from the observations. Per the classical technique, a biased estimate is used (see references). Based on this autocorrelation estimate, the parameters of an all-pole filter that could have produced the observations given a white noise input are estimated using the Levinson-Durbin algorithm. The transfer function of the all-pole filter is:

$$H(z) = \frac{1}{1 + \sum_{n=1}^{N} d_n z^{-n}}$$

The **`LevDur`** star outputs the d_n parameter estimates, also called the AR parameters. The autocorrelation method power spectrum estimate is simply

$$|H(e^{j\omega})|^2$$

so the rest of the autocorrelation galaxy is devoted to computing this quantity at various values of ω. The number of values of ω to use (512, in this case) is specified by the *resolution* parameter of the galaxy. The final output is scaled by an estimate of the power of the input process, extracted from the autocorrelation estimate using the **`Cut`** star. The **`Repeat`** star is needed to maintain consistent sample rates. The **`FloatPad`** star is used to prepend the AR coefficients with a one, the value of the first coefficient, which need not be computed.

The Burg method does not require first estimating the autocorrelation. It estimates the AR parameters directly from the input samples. The parameters are then processed exactly as above to produce a spectral estimate.

REFERENCES

[1] J. Makhoul, "Linear Prediction: A Tutorial Review", *Proc. IEEE*, Vol. 63, pp. 561-580, Apr. 1975.

[2] S. M. Kay, *Modern Spectral Estimation: Theory & Application*, Prentice-Hall, Englewood Cliffs, NJ, 1988.

[3] S. Haykin, *Modern Filters*, MacMillan Publishing Company, New York, 1989.

SEE ALSO: Burg, LevDur, levinsonDurbin, linearPrediction.

NAME: **`qam`**

A demo that produces a QAM signal

LOCATION: ˜ptolemy/src/domains/sdf/demo

DOMAIN: **`SDF` (DERIVED FROM: `Universe`)**

VERSION: 1.5 (12/17/92)

AUTHOR: *Joseph T. Buck*

DESCRIPTION:

This universe produces a 16-point quadrature amplitude modulated signal and displays the eye diagram for the in-phase part, the constellation, and the modulated transmit signal. The excess bandwidth is 100% and the carrier frequency is twice the symbol rate. Run it for about 100 iterations.

This is a port of the ‘‘transmitter’’ demo from Gabriel version 0.7.

REFERENCES

[1] E. A. Lee and D. G. Messerschmitt, *Digital Communication*, Kluwer Academic Publishers, Boston, MA, 1988, pp. 154-172

SEE ALSO: RaisedCos, Xscope

NAME: **`quantize`**

Demonstrate the Quantizer star.

LOCATION: ˜ptolemy/src/domains/sdf/demo

DOMAIN: **`SDF` (DERIVED FROM: `Universe`)**

VERSION: 1.4 (12/17/92)

AUTHOR: *Edward A. Lee*

DESCRIPTION:

This trivial system shows how to use the **`Quantizer`** star. Three thresholds, at -1.0, 0.0, and 1.0 are used to produce one of four levels, -1.5, -0.5, 0.5, 1.5. The **`FloatRamp`** star is used to provide inputs over the range of interest.

SEE ALSO: Quantizer, Sgn.

NAME: **`scramble`**

Simple commutator-distributor demo

LOCATION: ˜ptolemy/src/domains/sdf/demo

DOMAIN: **`SDF` (DERIVED FROM: `Universe`)**

VERSION: 1.4 (12/17/92)

AUTHOR: *J. T. Buck*

DESCRIPTION:

This system is a simple demonstration of the Commutator and Distributor stars.

SEE ALSO: Commutator, Distributor

NAME: **`sinMod`**

Sinusoidal modulator

LOCATION: ~ptolemy/src/domains/sdf/demo

DOMAIN: **`SDF`** **(DERIVED FROM: `Universe`)**

VERSION: 1.4 (October 15, 1990)

AUTHOR: *Edward A. Lee*

DESCRIPTION:

This system uses galaxy to produce a sinusoid, and another galaxy to modulate it. It serves primarily as a simple demonstration of nested galaxies.

NAME: **`telephoneChannelTest`**

A sinusoidal test signal is sent over a telephone channel simulation.

LOCATION: ~ptolemy/src/domains/sdf/demo

DOMAIN: **`SDF`** **(DERIVED FROM: `Universe`)**

VERSION: 1.412/17/92 ()

AUTHOR: *Edward A. Lee*

DESCRIPTION:

Assuming a sampling rate of 8 kHz, a sinusoid at 500 Hz is transmitted through a simulation of a telephone channel with additive Gaussian noise, nonlinear distortion, and phase jitter. The time domain plot of the output clearly shows the phase jitter and the noise, but only the most discerning eye can see the nonlinear distortion. A periodogram power spectrum estimate, however, clearly shows the second harmonic distortion, plus a DC offset.

SEE ALSO: TelephoneChannel

NAME: **`timeVarSpec`**

A time varying spectrum is displayed.

LOCATION: ˜ptolemy/src/domains/sdf/demo

DOMAIN: `SDF` **(DERIVED FROM:** `Universe`)

VERSION: 1.412/17/92 ()

AUTHOR: *Edward A. Lee*

DESCRIPTION:

A signal is generated by filtering white Gaussian noise and then modulating it with a chirp (a sinusoid with steadily increasing frequency). Spectral estimates are repeatedly computed using the autocorrelation method, and displayed using the Waterfall star. Note that in the display, the first plot shows a roughly baseband spectrum, while in subsequent displays the spectrum is centered at successively higher frequencies. Note also that hidden points are not displayed. This can be changed by altering the parameters of the Waterfall star.

SEE ALSO: autocorrelation, powerSpectrum, Waterfall.