Chapter 14. CGC Domain

Authors: *Joseph Buck*
Soonhoi Ha
Christopher Hylands
Edward A. Lee
Thomas M. Parks

Other Contributors: *Kennard White*
Mary Stewart

14.1 Introduction

The CGC domain generates code for the C programming language. This domain supports both synchronous dataflow (SDF, see "SDF Domain" on page 5-1) and Boolean-controlled dataflow (BDF, see "BDF Domain" on page 8-1) models of computation. The model associated with a particular program graph is determined by which target is selected. The `bdf-CGC` target supports the BDF model, while all other targets in the CGC domain support only the SDF model. Code can be generated for both single-processor and multi-processor computers. The targets that support single processors include `default-CGC`, `Makefile_C`, `TclTk_Target`, and `bdf-CGC`. The multi-processor targets are `unixMulti_C` and `NOWam`.

14.2 CGC Targets

The targets of the CGC domain generate C code from dataflow program graphs. Code generation is controlled by the *host*, *directory*, and *file* parameters as described in "Targets" on page 13-1. The command used to compile the code is determined by the *compileCommand*, *compileOptions*, and *linkOptions* parameters. Compilation and execution are controlled by the *display?*, *compile*?, and *run?* parameters, also described in "Targets" on page 13-1. The other parameters common to all CGC targets are listed below. Not all of these parameters are made available to the user by every target, and some targets define additional parameters.

staticBuffering (`INT`) Default = `TRUE`
If `TRUE`, then attempt to use static, compile-time addressing of data buffers between stars. Otherwise, generate code for dynamic, run-time addressing.

funcName (`STRING`) Default = `main`
The name of the main function. The default value of `main` is suitable for generating stand-alone programs. Choose another name if you wish to use the generated code as a procedure that is called from your own main program.

compileCommand (`STRING`) Default = `cc`
Command name of the compiler.

compileOption (STRING) Default =
Options passed to the compiler. The default is the empty string.

linkOptions (STRING) Default = -lm
Options passed to the linker.

resources (STRING) Default = STDIO
List of abstract resources that the host computer has.

14.2.1 Single-Processor Targets

The default-CGC target generates C code for a single processor from a SDF program graph. The parameters available to the user are shown in Table 14-1, "Parameters of the default-CGC target," on page 14-2. See "Targets" on page 13-1 and "CGC Targets" on page 14-1 for detailed descriptions of these parameters.

compile?	file	Looping Level
compileCommand	funcName	resources
compileOptions	host	run?
directory	linkOptions	staticBuffering
display?		

TABLE 14-1: Parameters of the default-CGC target

The Makefile_C target compiles CGC binaries with makefiles so that compile time architecture and site dependencies can be handled. The Makefile_C target generates a small makefile that is rcp'd over to the remote machine. The generated makefile is named after the universe. If the universe is called bigBang, then the makefile will be called bigBang.mk. We name the generated makefiles so that more than one makefile can exist in the users' directory.

The generated makefile uses $PTOLEMY/lib/cgc/makefile_C.mk as a starting point, and then appends lines to it. The generated makefile includes $PTOLEMY/mk/config-$PTARCH.mk, which determines architecture and site dependencies, such as which compiler to use, or where the X11 include files are. The user may modify makefile_C.mk and add site-dependent rules and variables there. If the user wants to have site dependent include files on the remote machines, then they could add include $(ROOT)/mk/mysite.mk to makefile_C.mk, and that file would be included on the remote machines at compile time.

On the remote machine, the Makefile_C target assumes:

- $PTOLEMY and $PTARCH are set on the remote machine when rshing.
- $PTOLEMY/mk/config-$PTARCH.mk and any makefile files included by that file are present.
- A make binary is present. The Makefile_C target does not assume GNU make, so the default makefile_C.mk does not include mk/common.mk. The reason not to assume GNU make is that we are not sure what the user's path is like when they log in. The user can require that GNU make be used by setting the *skeletonMakefile* target parameter to the name of a makefile that requires GNU make.

  If the remote machine does not fulfill these constraints, then the user should use the

Default_C target.

skeletonMakefile (STRING) Default=
The default value of this target parameter is the empty string, which means that we use $PTOLEMY/lib/cgc/makefile_C.mk as our skeleton makefile. If this parameter is not empty then the value of the parameter refers to the skeleton makefile to be copied into the generated makefile.

appendToMakefile (INT) Default = 1
This target parameter controls whether we append rules to the generated makefile or just copy it over to the remote machine. In the default situation, *appendToMakefile* is true and we append our rules after copying $PTOLEMY/lib/cgc/makefile_C.mk

The parent target of the Makefile_C target is default-CGC. If the parent target parameter *compileOptions* is set, then we process any environment variables in that string, and then add it to the end of the generated makefile as part of OTHERCFLAGS=. In a similar fashion, the parent target parameter *linkOptions* ends up as part of the right-hand side of LOADLIBES=.

The TclTk_Target target, which is derived from the Makefile_C target, must be used when Tcl/Tk stars are present in the program graph. The initial default of one parameter differs from that of the parent target.

skeletonMakefile (STRING) Default=$PTOLEMY/lib/cgc/TclTk_Target.mk
The TclTk_Target overrides this parent target parameter and sets it to the name of a skeleton makefile to be copied into the generated makefile.

The bdf-CGC target supports the BDF model of computation. It must be used when BDF stars are present in the program graph. It can also be used with program graphs that contain only SDF stars. The bdf-CGC target has the same parameters as the default-CGC target with the exception that the *Looping Level* parameter is absent. This is because a loop-generating algorithm is always used for scheduling. See “BDF Domain” on page 8-1 for details.

14.2.2 Multi-Processor Targets

Currently, the CGC domain supports two multi-processor targets: unixMulti_C and NOWam. The unixMulti_C target generates code for multiple networked workstations using a shared bus configuration for scheduling purposes. Inter-processor communication is implemented by splicing send/receive stars into the program graph. These communication stars use the TCP/IP protocol. In addition to the target parameters described in “CGC Targets” on page 14-1 and “Targets” on page 13-1, this target defines the user parameters listed below. Table 14-2, “Parameters of the unixMulti_C target,” on page 14-4 gives the complete list of parameters for the unixMulti_C target.

```
adjustSchedule  ignoreIPC        overlapComm
amortizedComm   inheritProcessors portNumber
childType       logFile          relTimeScales
compile?        machineNames     resources
directory       manualAssignment run?
display?        nameSuffix       sendTime
file            nprocs           userCluster
ganttChart      oneStarOneProc   tabular
```

TABLE 14-2: Parameters of the `unixMulti_C` target

portNumber (`INT`) Default = `7654`
The starting TCP/IP port number used by send/receive stars. The port number is incremented for each send/receive pair. It is the responsibility of the user to ensure that the port number does not conflict with any that may already be in use.

machineNames (`STRING`) Default = `herschel`
The host names of the workstations which form the multi-processor. The names should be separated by a comma (‘,’).

nameSuffix (`STRING`) Default =
The default is the empty string. The domain suffix for the workstations named in *machineNames*. If left blank, which is the default, then the workstations are assumed to be part of the local domain. Otherwise, specify the proper domain name, including a leading period. This string is appended to the names in *machineNames* to form the fully qualified host names.

The `NOWam` target uses Networks Of Workstations (NOW) active messages to communicate between machines. The NOW project is an effort to use many commodity workstations to create a building-wide supercomputer. For more information about the NOW project, see `http://now.cs.berkeley.edu`. Currently, the `NOWam` target is still experimental, and only a proof of concept. The `NOWam` target has the following target parameters:

machineName (`STRING`) Default = `lucky, babbage`
The host names of the workstations which form the multi-processor. The names should be separated by a comma (‘,’). The `NOWam` target will not work on the local machines, the machines named by this parameter must be remote machines. Note that the default of this parameter differs from the default in the `UnixMulti_C` target.

nameSuffix (`STRING`) Default =
The default is the empty string. See the description of *nameSuffix* in `UnixMulti_C` above.

14.2.3 Setting Parameters Using Command-line Arguments

The pragma facility allows users to identify any parameters that the user would like to

be able to change on the command line of the CGC binary. CGC command line arguments has not been extensively tested yet. Currently, it is only supported for scalar parameters with `FLOAT` and `INT` values. Also, it is only working for parameters of Stars at the top level, i.e. it will not work with Galaxies' and Universes' parameters, or parameters of Stars in Galaxies.

To specify a parameter for setting via the command-line, place the cursor over the Star and invoke the *edit-pragmas* command ('`a`'). In the dialog box, enter the name of the parameter to be made settable, follow by white space, then the name of the command-line option with which to set the parameter. This parameter /option-name pair should be entered for each of the required parameters, with pairs separated by white space.

Now, the generated program will take the new options each followed by a value with which to set the corresponding parameters. If the command-line option is not specified for a parameter, it will be initialized to its default value, which will be the value set by the *edit-params* command ('`e`'). In addition, if the '`-h`', '`-help`' or '`-HELP`' option is specified, the program will print the option-names corresponding to the settable parameters with their default values.

14.3 An Overview of CGC Stars

Figure 14-1 shows the top-level palette of CGC stars. The stars are divided into categories: sources, sinks, arithmetic functions, nonlinear functions, control, Sun UltraSparc VIS-conversion, signal processing, boolean-controlled dataflow, Tcl/Tk and higher-order function (HOF) stars. Icons for `delay`, `bus`, and `BlackHole` appear in most palettes for easy access. Many of the stars in the CGC domain have equivalent counterparts in the SDF domain. See "An overview of SDF stars" on page 5-4 for brief descriptions of these stars. Brief descriptions of the stars unique to the CGC domain are given in the following sections.

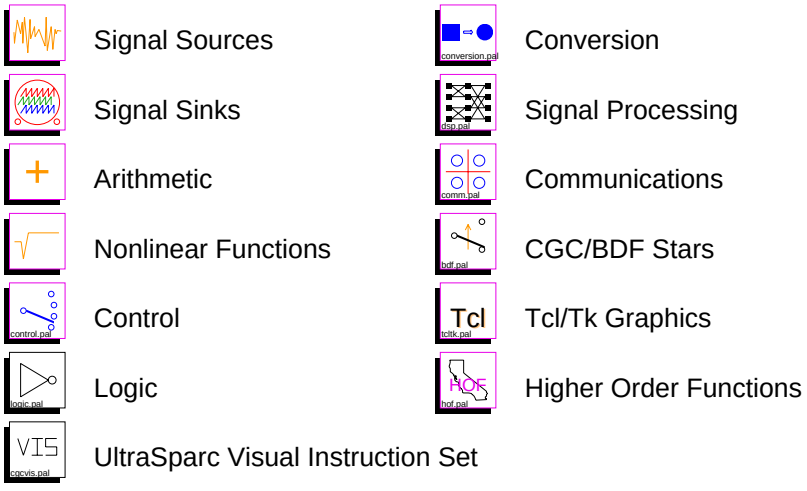


FIGURE 14-1: Top-level palette of stars in the CGC domain

14.3.1 Source Stars

Source stars have no inputs and produce data on their outputs. Figure 14-2 shows the palette of CGC source stars. The following stars are equivalent to the SDF stars of the same

name (see “Source stars” on page 5-5): `Const`, `IIDUniform`, `Ramp`, `Rect`, `singen`, `WaveForm`, `TclScript`, `TkSlider`, `RampFix`, `RectFix`, `RampInt`, `expgen`. Stars that are unique to the CGC domain are described briefly below.

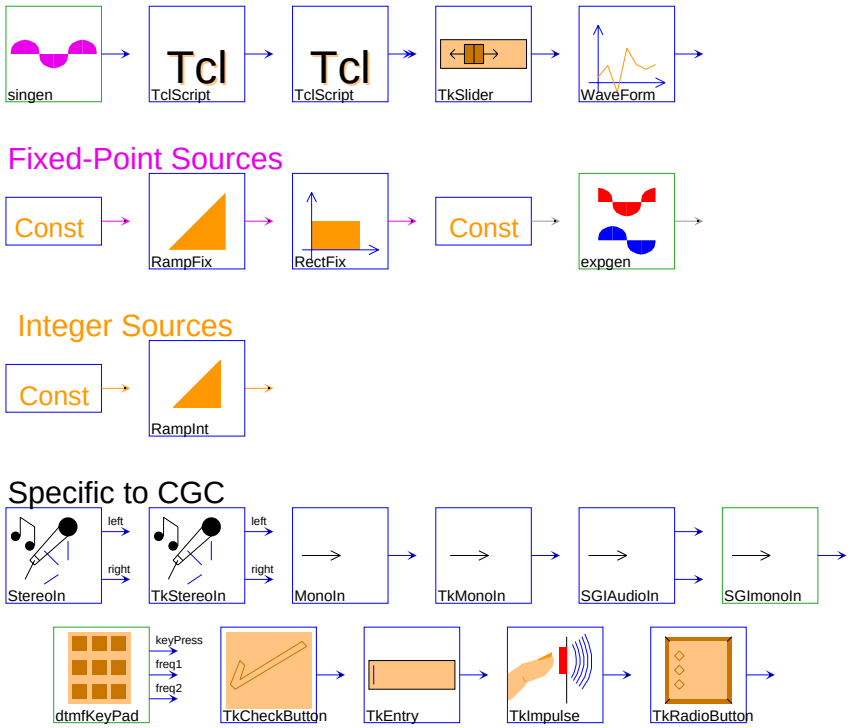


FIGURE 14-2: Source stars in the CGC domain

`StereoIn`	Reads Compact Disc format audio data from a file given by `fileName`. The file can be the audio port `/dev/audio`, if supported by the workstation. The data read is linear 16 bit encoded and stereo (2 channel) format.
`TkStereoIn`	Just like `StereoIn`, except that a Tk slider is put in the master control panel to control the volume.
`MonoIn`	`Reads` `mono` (1 channel) data with either linear16 or ulaw8 encoding from a file given by `fileName`. The file can be the audio port `/dev/audio` , if supported by the workstation.
`TkMonoIn`	Just like `MonoIn`, except that a Tk slider is put in the master control panel to control the volume.
`SGImonoIn`	(SGI only) Average the stereo audio output of an `SGIAudioIn` star into one mono output.
`SGIAudioIn`	(SGI only) Get samples from the audio input port on an Silicon Graphics workstation.
`dtmfKeyPad`	Generate a Dual-Tone Modulated Frequency (DTMF) signal.

`TkCheckButton`	A simple Tk on/off input source.
`TkEntry`	Output a constant signal with value determined by a Tk entry box (default 0.0).
`TkImpulse`	Output a specified value when a button is pushed. Optionally synchronize by halting until the button is pushed.
`TkRadioButton`	Graphical one-of-many input source.

14.3.2 Sink Stars

Sink stars have no outputs and consume data on their inputs. Figure 14-3 shows the palette of CGC sink stars. The following stars are equivalent to the SDF stars of the same name (see “Sink stars” on page 5-9): `XMgraph`, `XYgraph`, `Xscope`, `TkBarGraph`, `TkPlot`, `TKXYPlot`, `TclScript`, `Printer`. Stars that are unique to the CGC domain are described briefly below.

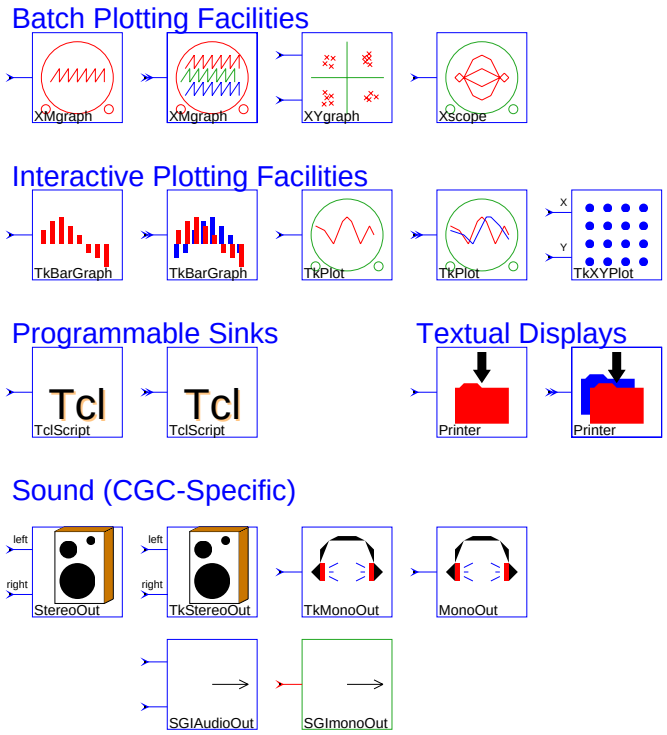


FIGURE 14-3: Sink stars in the CGC domain

`StereoOut`	Writes Compact Disc audio format to a file given by `file-Name`. The file can be the audio port `/dev/audio`, if supported by the workstation. The data written is linear 16 bit encoded and stereo (2 channel) format.

`TkStereoOut`	Just like `StereoOut` except that Tk sliders are put in the master control panel to control the volume and balance.
`TkMonoOut`	Just like `MonoOut` except that Tk sliders are put in the master control panel to control the volume.
`MonoOut`	Writes mono (1 channel) data with either linear16 or ulaw8 encoding to a file given by `fileName`. The file can be the audio port `/dev/audio`, if supported by the workstation. If the *aheadlimit* parameter is non-negative, then it specifies the maximum number of samples that the program is allowed to compute ahead of real time.
`SGIAudioOut`	(SGI Only) Put samples into an audio output port.
`SGIMonoOut`	(SGI Only) A galaxy that takes a mono output and drives the stereo `SGIAudioOut` star below.

14.3.3 Arithmetic Stars

Arithmetic stars perform simple functions such as addition and multiplication. Figure 14-4 shows the palette of CGC arithmetic stars. All of the stars are equivalent to the SDF stars of the same name (see “Arithmetic stars” on page 5-12): `Add`, `Gain`, `Integrator`, `Mpy`, `Sub`.

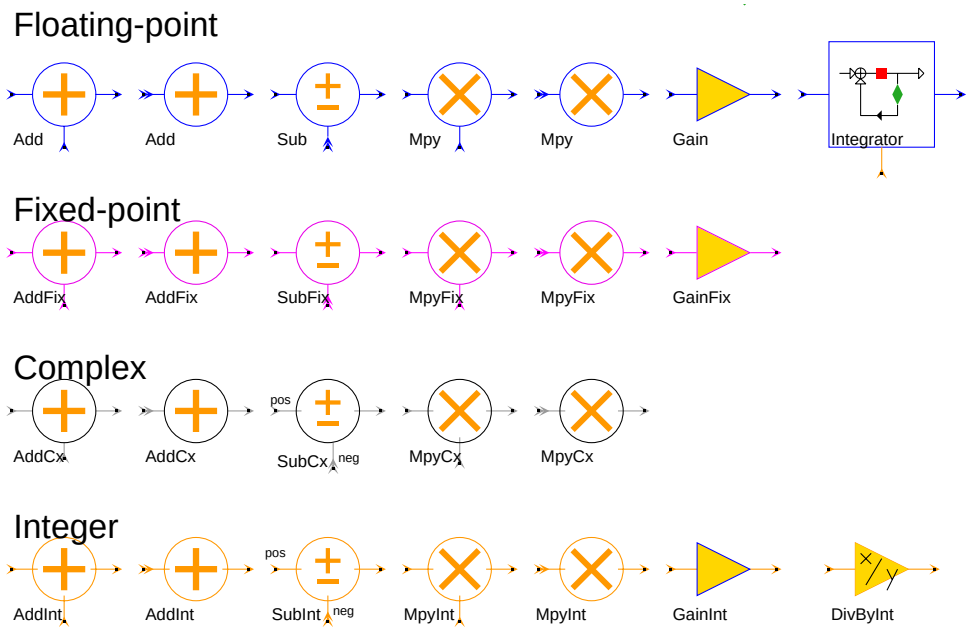


FIGURE 14-4: Arithmetic stars in the CGC domain

14.3.4 Nonlinear Stars

Nonlinear stars perform simple functions. Figure 14-5 shows the palette of CGC nonlinear stars. The following stars are equivalent to the SDF stars of the same name (see “Nonlinear stars” on page 5-13): `Abs`, `cexp`, `conj`, `Cos`, `Dirichlet`, `Exp`, `expjx`, `Floor`,

`Limit`, `Log`, `MaxMin`, `Modulo`, `ModuloInt`, `OrderTwoInt`, `Reciprocal`, `Sgn`, `Sin`, `Sinc`, `Sqrt`, `powerEst`, `Quant`, `Table`, `TclScript`. Stars that are unique to the CGC domain are described briefly below.

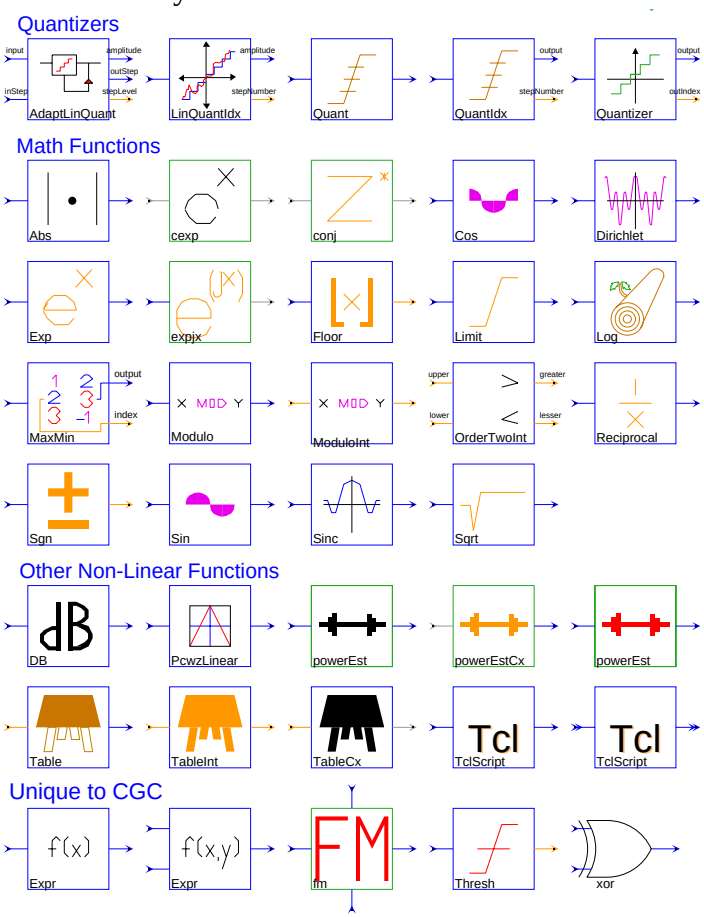


FIGURE 14-5: Nonlinear stars in the CGC domain

`Expr`	(Two icons) General expression evaluation. This star evaluates the expression given by the *expr* parameter and writes the result on the output. The default expression, which is `$ref(in#1)`, simply copies the first input to the output.
`fm`	Modulate a signal by frequency.
`Thresh`	Compares input values to threshold. The output is `0` if input <= threshold, otherwise it is `1`.
`xor`	Exclusive-OR two signals.

14.3.5 Control Stars

Control stars are used for routing data and other control functions. Figure 14-6 shows the palette of CGC control stars. The following stars are equivalent to the SDF stars of the same name (see “Conversion stars” on page 5-20): `Fork`, `Chop`, `ChopVarOffset`, `Commutator`, `DeMux`, `Distributor`, `DownSample`, `Mux`, `Repeat`, `UpSample`. Stars that are

unique to the CGC domain are described briefly belowp.

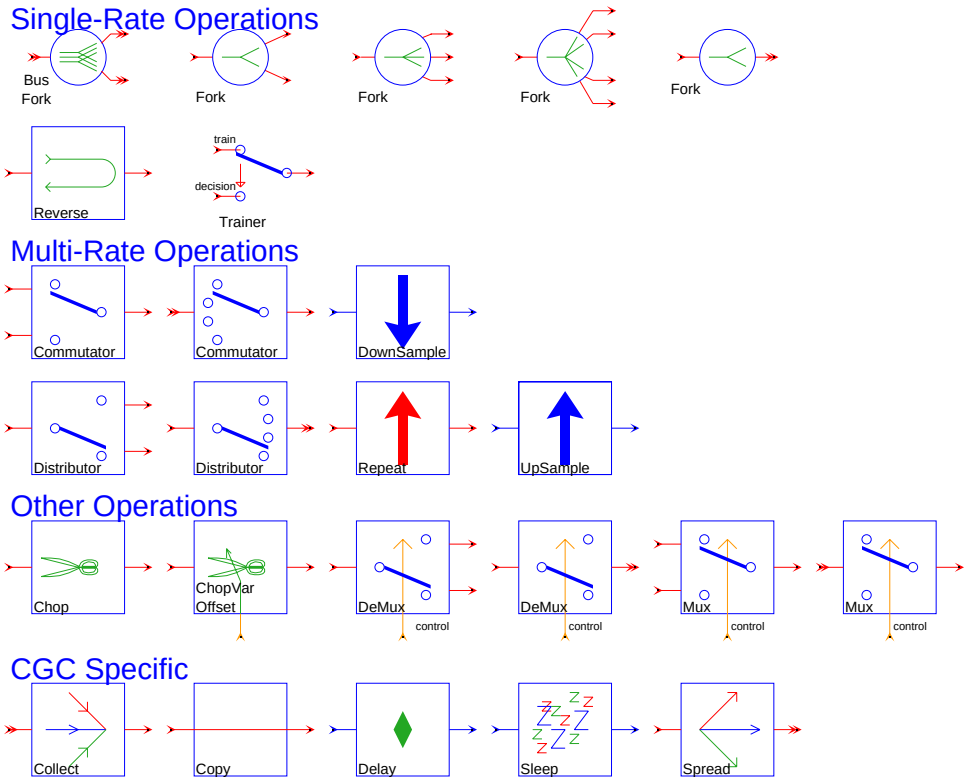


FIGURE 14-6: Control stars in the CGC domain

`Collect`	Takes multiple inputs and produces one output. This star does not generate code. In multiprocessor code generation, it is automatically attached to a porthole if it has multiple sources. Its role is just opposite to that of the `Spread` star.
`Copy`	‘Copy’ stars are added if an input/output PortHole is a host/embedded PortHole and the buffer size is greater than the number of Particles transferred.
`Delay`	Delay an input by *delay* samples.
`Sleep`	Suspend execution for an interval (in milliseconds). The input is passed to the output when the process resumes.
`Spread`	Takes one input and produces multiple outputs. This star does not generate any code. In multiprocessor code generation, this star is automatically attached to a porthole whose outputs are passed to more than one destination (one ordinary block and one `Send` star, more than one `Send` star, and so on.)

14.3.6 Logic Stars

Figure 14-7 shows the palette of CGC Logic stars.

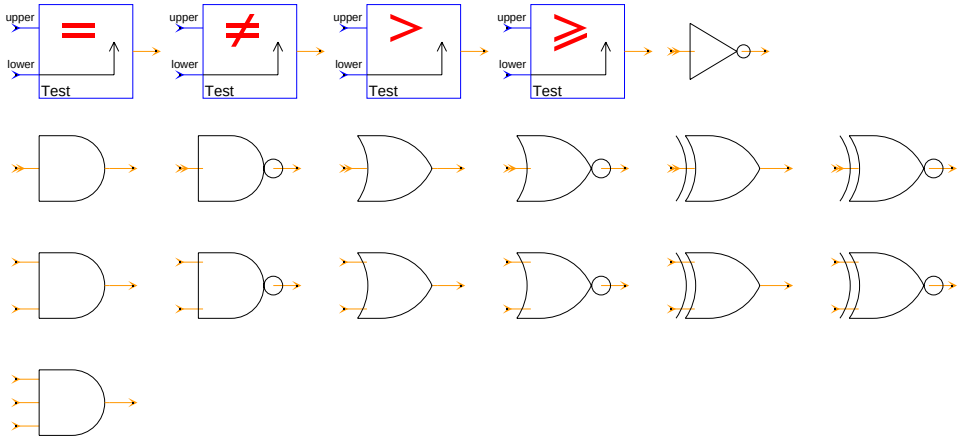
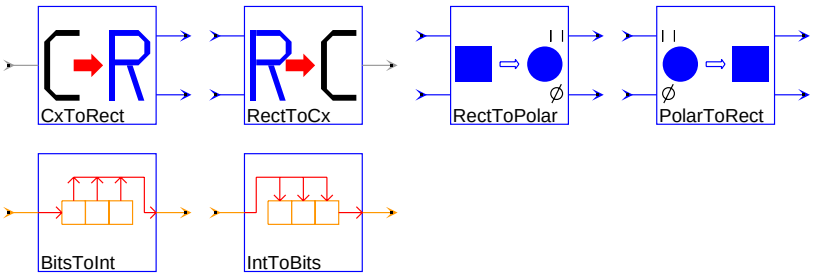


FIGURE 14-7: Logic stars in the CGC domain

14.3.7 Conversion Stars

Conversion stars are used to convert between complex and real numbers. Figure 14-8 shows the palette of CGC conversion stars. All of the stars are equivalent to the SDF stars of the same name (see “Conversion stars” on page 5-20): `CxToRect`, `PolarToRect`, `RectToCx`, `RectToPolar`.



For explicit (vs. automatic) type conversion:

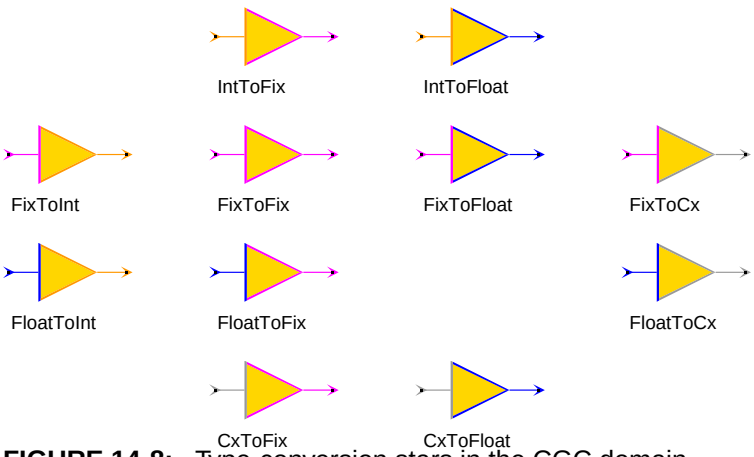


FIGURE 14-8: Type-conversion stars in the CGC domain

14.3.8 Signal Processing Stars

Figure 14-9 shows the palette of CGC signal processing stars. The following stars are equivalent to the SDF stars of the same name (see “Signal processing stars” on page 5-30): `DB`, `FIR`, `FIRFix`, `FFTCx`, `GAL`, `GGAL`, `Goertzel`, `LMS`, `LMSOscDet`, `LMSTkPlot`. The `IIR`, `RaisedCosine` and `Window` CGC stars are not present in Ptolemy0.6. Stars that are unique to the CGC domain are described briefly below.

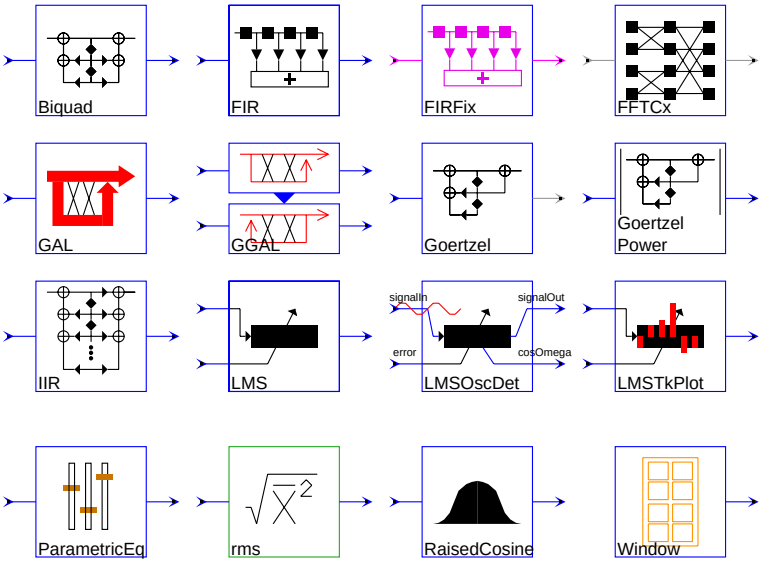


FIGURE 14-9: Signal processing stars in the CGC domain

`GoertzelPower`	Second-order recursive computation of the power of the kth coefficient of an N-point DFT using Goertzel’s algorithm. This form is used in touchtone decoding.
`ParametricEq`	A two-pole, two-zero parametric digital IIR filter (a biquad).
`rms`	Calculate the Root Mean Squared of a signal.

14.3.9 Communications Stars

Figure 14-10 shows the communications stars in the CGC domain. The following stars

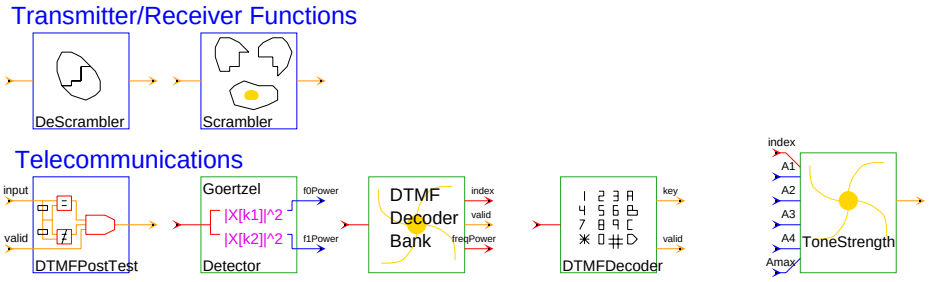


FIGURE 14-10: Communications stars in the CGC domain

are equivalent to the SDF stars of the same name in the communications palette, (See “Communication stars” on page 5-36): `DeScrambler`, `Scrambler`. The following stars are equivalent to the SDF stars of the same name in the telecommunications palette, (see “Telecommunications” on page 5-39): `DTMFPostTest`, `GoertzelDetector`, `DTMFDe-`

coderBand, DTMFDecoder, ToneStrength.

14.3.10 BDF Stars

BDF stars are used for conditionally routing data. Figure 14-11 shows the palette of BDF stars in the CGC domain. These stars require the use of the bdf-CGC target (see “Single-Processor Targets” on page 14-2). Unlike their simulation counterparts (see “An overview of BDF stars” on page 8-2), these stars can only transfer single tokens in one firing.

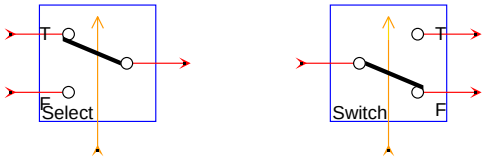


FIGURE 14-11: BDF stars in the CGC domain

Select	This star requires a BDF scheduler. If the value on the *control* line is nonzero, *trueInput* is copied to the output; otherwise, *falseInput* is.
Switch	This star requires a BDF scheduler. Switches *input* events to one of two outputs, depending on the value of the *control* input. If *control* is true, the value is written to *trueOutput*; otherwise it is written to *falseOutput*.

14.3.11 Tcl/Tk Stars

Tcl/Tk stars require the use of the TclTk_Target target. They can be used to provide an interactive user interface with Tk widgets. Figure 14-12 shows the palette of Tcl/Tk stars available in the CGC domain. Most of these stars are described in the sources, sinks and non-

linear palettes.

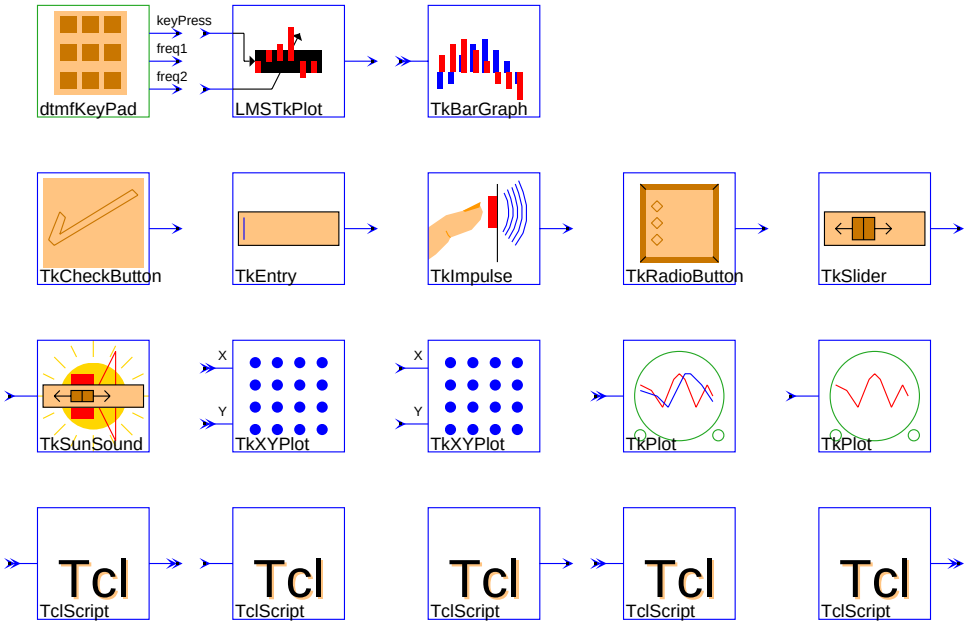


FIGURE 14-12: Tcl/Tk stars in the CGC domain

`TkParametricEq` Just like `ParametricEq` star, except that a Tk slider is put in the master control panel to control the gain, bandwidth, and center and cut-off frequencies.

14.3.12 Higher Order Function Stars

For information on the HOF stars, please see “An overview of the HOF stars” on page 6-15.

14.3.13 UltraSparc VIS (Visual Instruction Set) Stars

These stars generate code that includes instructions for the UltraSparc’s Visual Instruction Set (VIS). These stars only run on Sun UltraSparc workstations (see “UltraSparc VIS Demos” on page 14-26 for more information about using CGCVIS demos.) All of these stars process data in “quad-words”—64-bit words, each containing four, 16-bit signed integers. All of the stars exhibit some speed improvement over the equivalent stars written in floating-point, although a substantial effort is needed in coding them to realize this performance gain.

CGC VIS universes that create standalone applications, such as the 256fft demo, should use the CGC `Makefile_C` target and set the *skeletonMakefile* target parameter to `$PTOLEMY/lib/cgc/makefile_VIS.mk`. Universes that use CGC VIS stars and TclTk stars should use `makefile_TclTk_VIS.mk`. The `CGCVISSim` target can be used to simulate

VIS stars, but this target is very experimental.

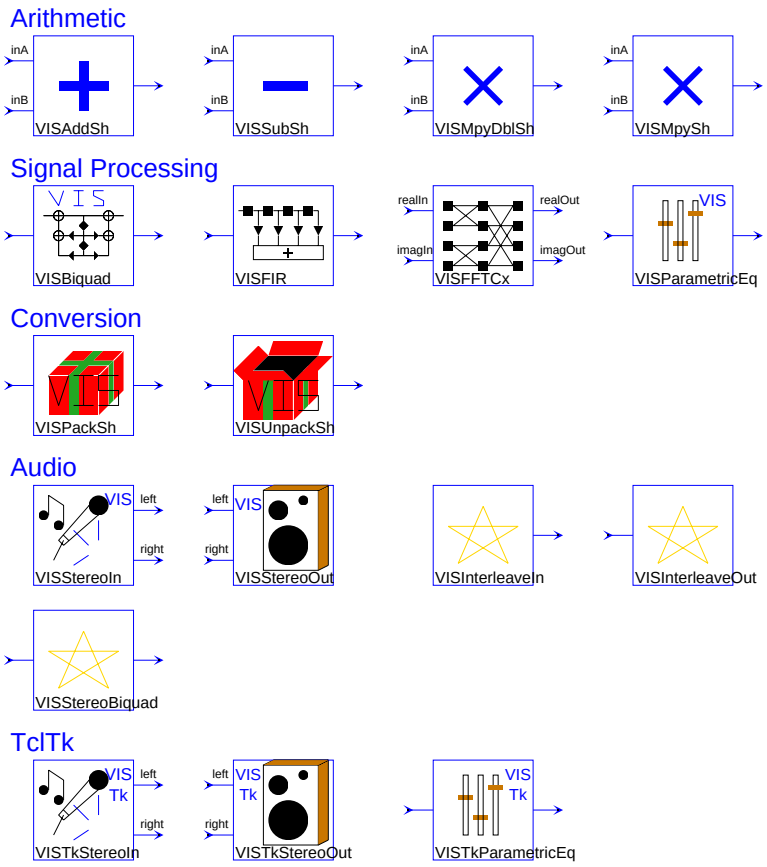


FIGURE 14-13: UltraSparc Visual Instruction Set

`VISAddSh` Add the corresponding 16-bit fixed point numbers of two partitioned float particles. Four signed 16-bit fixed point numbers of a partitioned 64-bit float particle are added to those of another 64-bit float particle. The result is returned as a single 64-bit float particle. There is no saturation arithmetic so that overflow results in wrap around.

`VISSubSh` Subtract the corresponding 16-bit fixed point numbers of two-partitioned float particles. Four signed 16-bit fixed point numbers of a partitioned 64-bit float particle are subtracted from those of another 64-bit float particle. The result is returned as a single 64-bit float particle. There is no saturation arithmetic so that overflow results in wrap around.

`VISMpyDblSh` Multiply the corresponding 16-bit fixed point numbers of two-partitioned float particles. Four signed 16-bit fixed point numbers of a partitioned 64-bit float particle are multiplied to those of another 64-bit float particle. Each multiplication produces a 32-bit result. Each 32-bit result is then left-shifted to fit within

a certain dynamic range and truncated to 16 bits. The final result is four 16-bit fixed point numbers that are returned as a single float particle.

`VISMpySh` Multiply the corresponding 16-bit fixed point numbers of two-partitioned float particles. Four signed 16-bit fixed pointnumbers of a partitioned 64-bit float particle are multiplied to those of another 64-bit float particle. Each multiplication produces a 32-bit result, which is then truncated to 16 bits. The final result is four 16-bit fixed point numbers that are returned as a single float particle.

`VISBiquad` An IIR biquad filter. In order to take advantage of the 16-bit partitioned multiplies, the VIS biquad reformulates the filtering operation to that of a matrix operation (Ax=y), whereVIS A is a matrix calculated from the taps, x is an input vector, and y is an output vector.The matrix A is first calculated by substituting the biquad equation y[n] = -a*y[n-1]-b*y[n-2]+c*x[n]+d*x[n-1]+e*x[n-2] into y[n-1], y[n-2], and y[n-3]. The matrix A is then multiplied with the 16-bit partitioned input vector. The final result is accumulated in four 16-bit fixed point numbers which are concatenated into a single 64-bit float particle.

`VISFIR` A finite impulse response (FIR) filter. In order to take advantage of the 16-bit partitioned multiplies, the VIS FIR reformulates the filtering operation to that of a matrix operation (Ax=y), where A is a tap matrix, x is an input vector, and y is an outputvector. The matrix A is first constructed from the filter taps. Each row is filled by copying the filter taps, zero-padding so that its length is a multiple of 4, and shifting to the right by one. Four of these rows are used to build up matrix A. The matrix A is then multiplied with the 16-bit partitioned input vector. This is equivalent to taking four sum of products. The final result is accumulated in four 16-bit fixed point numbers which are concatenated into a single 64-bit float particle.

`VISFFTCx` A radix-2 FFT of a complex input. The radix-2 decimation-in-time decomposes the overall FFT operation into a series of smaller FFT operations. The smallest operation is the “FFT butterfly” which consists of a single addition and subtraction. Graphically, the full decomposition can be viewed as N stages of FFT butterflies with twiddle factors between each of the stages. One standard implementation is to use three nested for loops to calculate the FFT. The innermost loop calculates all the butterflies and performs twiddle factor multiplcations within a particular stage; the next outer loop calculates the twiddle factors; and the outermost loop steps through all the stages. In order to take advantage of the 16-bit partitioned multiplications

and additions, the basic operation of the VIS FFT is actually doing four “FFT butterflies” at once. The implementation is similar to the standard three nested for loops, but the last two stages are separated out. In order to avoid packing and unpacking, the basic operation of the last two stages switches from four to two to eventually just one “FFT butterfly”. After the FFT is taken, the order of the sequence is bit-reversed.

`VISParametricEq` The user supplies the parameters such as *Bandwidth*, *`Center Frequency`*, and *Gain*. The digital biquad coefficients are quickly calculated based on the procedure defined by Shpak.

`VISPackSh` Takes four float particles, casts them into four signed 16-bit-fixed point numbers, and packs them into a single 64-bit float-particle. The input float particles are first down cast into 16-bit fixed point numbers. The location of the binary point of the fixed point number can be placed anywhere by adjusting the scale parameter. The fixed point numbers are then concatenated to produce a 64-bit result. The order of the fixed point numbers can be reversed so that the most current input either leads or trails the pack, ie reverse equals `FALSE` produces (x[n],x[n-1],x[n-2],x[n-3]) and reverse equals `TRUE` produces (x[n-3],x[n-2],x[n-1],x[n]).

`VISUnpackSh` Takes a single 64-bit float particle, unpacks them into four 16-bit fixed point numbers, and casts them into four float particles. The input float particle is first separated into four 16-bit fixed point numbers. Once again, the order of the fixed point numbers can be reversed. The fixed point numbers are then up cast to float particles. The exponent value of each float particle can be adjusted by the scaledown parameter.

`VISStereoIn` Reads Compact Disc audio format from a file given by *fileName*. The file can be the audio port `/dev/audio`, if supported by the workstation. The star reads *`blockSize`* 16-bit samples at each invocation. The *blocksize* should be a multiple of 4.

`VISStereoOut` Writes Compact Disc audio format to a file given by *fileName.* The file can be the audio port `/dev/audio`, if supported by the workstation. The star writes *blockSize* 16-bit samples at each invocation. The *blocksize* should be a multiple of 4.

`VISInterleaveIn` Reads Compact Disc audio format from a file given by *fileName*. The file can be the audio port `/dev/audio`, if supported by the workstation. The star reads *blockSize* 16-bit samples at each invocation. The *blocksize* should be a multiple of 4.

`VISInterleaveOut` Reads Compact Disc audio format from a file given by *fileName*. The file can be the audio port `/dev/audio`, if supported by the workstation. The star reads *blockSize* 16-bit samples at

each invocation. The *blocksize* should be a multiple of 4.

`VISStereoBiquad` A two-pole, two-zero IIR filter.

`VISTkStereoIn` Just like StereoIn, except that a Tk slider is put in the master-control panel to control the volume.

`VISTkStereoOut` Just like StereoOut, except that a Tk slider is put in the master-control panel to control the volume and balance.

`VISTkParametricEq` Just like VISParametricEq, except that a Tk slider is put in the master control panel to control the gain.

14.3.14 An Overview of CGC Demos

Figure 14-14 shows the top-level palette of CGC demos. The demos are divided into categories: basic, multirate, signal processing, multi-processor, sound, Tcl/Tk, BDF,HOF and SDF-CGC wormhole demos[1].Many of the demos in the CGC domain have equivalent counterparts in the SDF or BDF domains. See “An overview of SDF demonstrations” on page 5-51, or “An overview of BDF demos” on page 8-3 for brief descriptions of these demos. Brief

1. In Ptolemy0.6, the SDF-CGC Wormhole icon is not present in the CGC demo palette. The SDF-CGC Wormhole demos can be found in the Mixed Domain demo palette, located in the top level Ptolemy palette at $PTOLEMY/demo/init.pal.

descriptions of the demos unique to the CGC domain are given in the sections that follow.

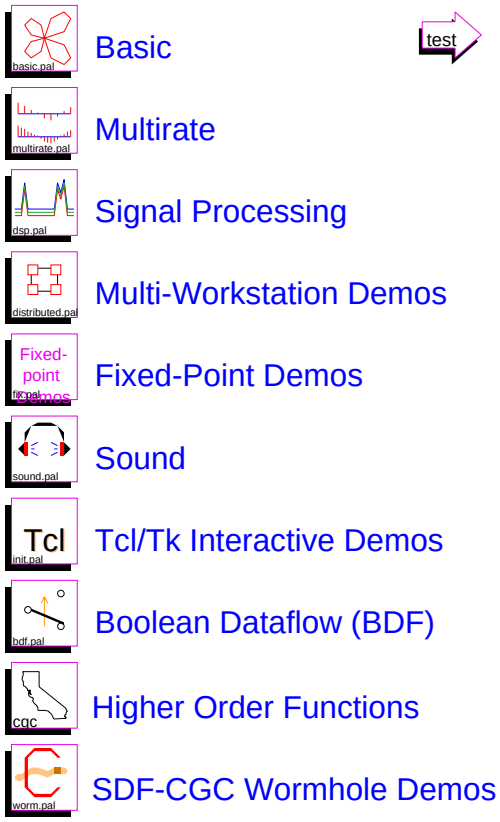


FIGURE 14-14: Top-level palette of demos in the CGC domain

14.3.15 Basic Demos

Figure 14-15 shows the palette of basic demos that are available in the CGC domain. The following demos are equivalent to the SDF demos of the same name (see “An overview of SDF demonstrations” on page 5-51): `butterfly`, `chaos`, `integrator`, `quantize`. The other demos in this palette are described briefly below.

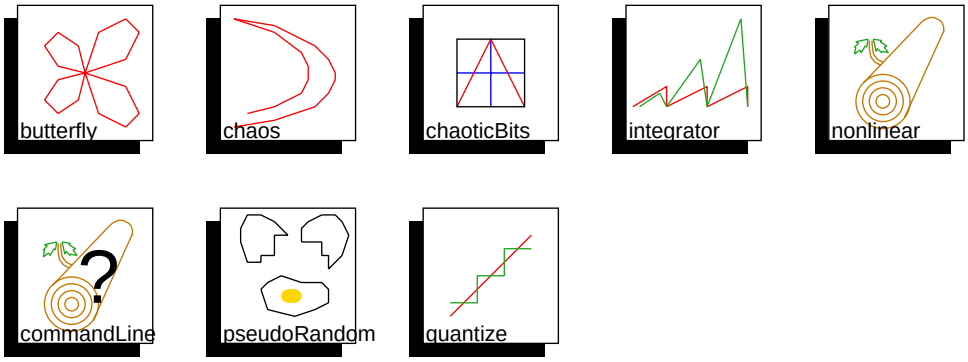


FIGURE 14-15: Basic CGC demos

`chaoticBits`	Chaotic Markov map with quantizer to generate random bit sequence.
`nonlinear`	This simple system plots four nonlinear functions over the range 1.0 to 1.99. The four functions are exponential, natural logarithm, square root, and reciprocal.
`commandline`	This demo is a slight modification of the nonlinear demo. It uses the pragma mechanism to indicate the parameters that are to be made settable from the command-line.
`pseudoRandom`	Generate pseudo-random sequences.

14.3.16 Multirate Demos

Figure 14-16 shows the palette of multirate demos available in the CGC domain. The following demos are equivalent to the SDF demos of the same name (see “An overview of SDF demonstrations” on page 5-51): `interp`, `filterBank`. The other demos in this palette are described briefly below.

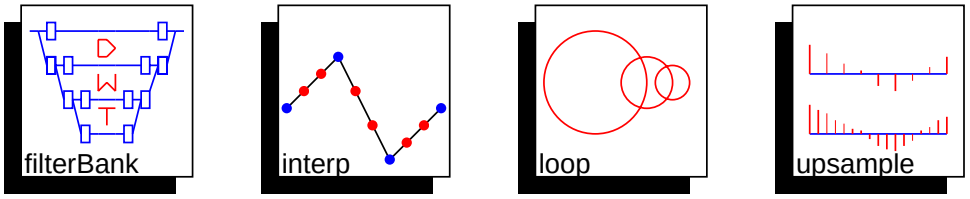


FIGURE 14-16: CGC Multirate demos

`upsample`	This simple up-sample demo tests static buffering. Each invocation of the `XMgraph` star reads its input from a fixed buffer location since the buffer between the `UpSample` star and the `XMgraph` star is static.
`loop`	This demo demonstrates the code size reduction achieved with a loop-generating scheduling algorithm.

14.3.17 Signal Processing Demos

Figure 14-17 shows the palette of signal processing demos that are available in the CGC domain. The following demos are equivalent to the SDF demos of the same name (see “An overview of SDF demonstrations” on page 5-51): `adaptFilter`, `dft`. The `animat-`

edLMS demo is described in “Tcl/Tk Demos” on page 14-24.

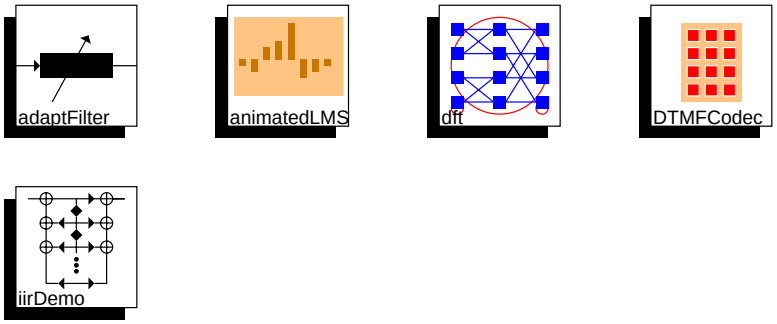


FIGURE 14-17: Signal processing demos in the CGC domain

DTMFCodec	Generate and decode touch tones.
iirDemo	Two equivalend implementations of IIR filtering. One of the implementations uses the IIR star. This demo is not present in Ptolemy0.6.

14.3.18 Multi-Processor Demos

Figure 14-18 shows the top-level palette of multi-processor demos available in the CGC domain. Ptolemy contains two multi-processor targets, unixMulti_C and NOWam. The demos in each target subpalette are the same. These demos would actually run faster on a single processor, but they do serve as a ‘proof of concept’.

FIGURE 14-18: Multi-processor demos in the CGC domain

Figure 14-19 shows the palette of multi-processor demos that use the unixMulti_C target to communicate between workstations.

FIGURE 14-19: Multi-workstation CGC demos

adaptFilter_multi	This is a multi-processor version of the adaptFilter demo. The graph is manually partitioned onto two networked workstations.
spread	This system demonstrates the Spread and Collect stars. It

shows how multiple invocations of a star can be scheduled onto more than one processor.

Figure 14-20 shows the demos that use the `NOWam` target to communicate between workstations. The demos in this palette are the same as the demos in the `UnixMulti_C` palette above.

FIGURE 14-20: Networks of Workstations (NOW) CGC demos

14.3.19 Fixed-Point Demos

Figure 14-21 shows the fixed-point demonstrations.

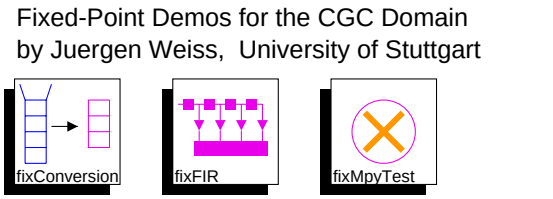


FIGURE 14-21: Fixed-point demos in the CGC domain

`fixConversion`	Demonstrate fixed-point conversion and overflow effects.
`fixFIR`	Demonstrate tap quantization effects on the transfer function of FIR filters.
`fixMpyTest`	Demonstrate retargeting of a SDF fixed-point multiply demo to CGC.

14.3.20 Sound-Making Demos

Figure 14-22 shows the palette of sound demos available in the CGC domain. Your workstation must be equipped with an audio device that can accept 16-bit linear or μ-law encoded PCM data, for these demos to work.For information about how to use the audio capa-

bilities of a workstation, see “Sounds” on page 2-38.

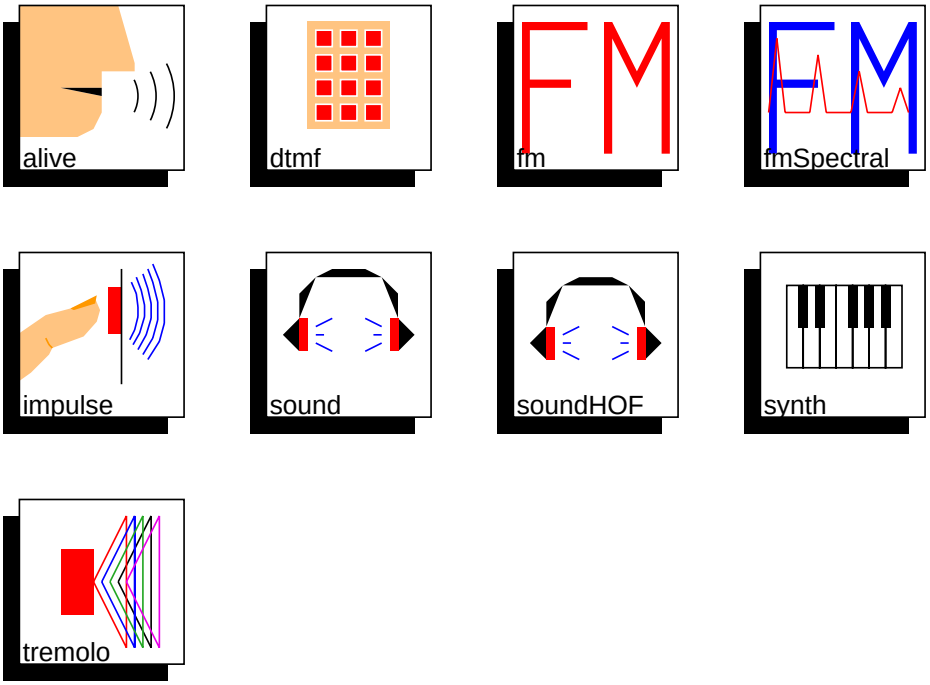


FIGURE 14-22: Sound-making demos in the CGC domain

`alive`	(SGI Only) Processes audio in real time, with an effect similar to the effects Peter Frampton used in the late 70’s rock album ‘Frampton Comes Alive’.
`dtmf`	This demo generates the same dual-tone multi-frequency tones you hear when you dial your telephone. The interface resembles the keypad of a telephone.
`fm`	This demo uses frequency modulation (FM) to synthesize a tone on the workstation speaker. You can adjust the modulation index, pitch, and volume in real time.
`fmSpectral`	FM synthesis with a spectral display.
`impulse`	This demo generates tones on the workstation speaker with decaying amplitude envelopes using frequency modulation synthesis. You can make tones by pushing a button. You can adjust the pitch, modulation index, and volume in real time.
`sound`	Generate a sound to play over the workstation speaker (or headphones).
`soundHOF`	Produce a sound made by adding a fundamental and its harmonics in amounts controlled by sliders.
`synth`	This demo generates sinusoidal tones on the workstation

speaker. You can control the pitch with a piano-like interface.

`tremolo` This demo produces a tremolo (amplitude modulation) effect on the workstation speaker. You can adjust the pitch, modulation frequency, and volume in real time.

14.3.21 Tcl/Tk Demos

These demos show off the capabilities of the Tcl/Tk stars, which must be used with the `TclTk_Target` target. Graphical user interface widgets are used to control input parameters and to produce animation. Many of these demos also produce sound on the workstation speaker with the `TkMonoOut` star (see “Tcl/Tk Stars” on page 14-13). Due to the overhead of processing Tk events, you must have a fast workstation (SPARCstation 10 or better) in order to have continuous sound output. You may be able to get continuous sound output on slower workstations if you avoid moving your mouse. Figure 14-23 shows the demos that are available. The following audio demos are documented in the previous section: `dtmf`, `fm`, `audioio`, `impulse`, `synth`, `tremolo`.

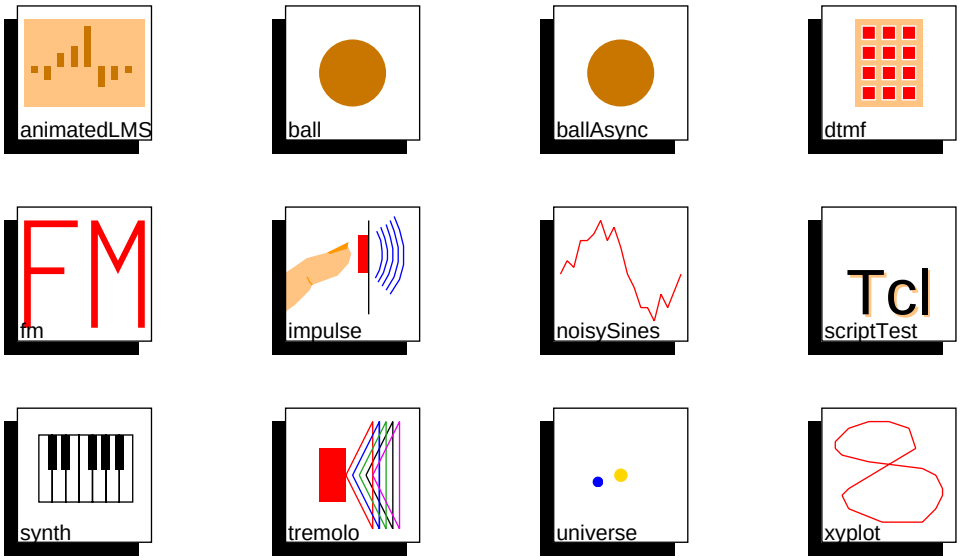


FIGURE 14-23: Tcl/Tk demos in the CGC domain

`animatedLMS` This demo is a simplified version of the SDF demo of the same name.

`ball` This demo exhibits sinusoidal motion with a ball moving back and forth.

`ballAsync` This demo is the same as the `ball` demo except that animation is updated asynchronously.

`noisySines` Generate a number of sinusoids with controllable additive noise.

`scriptTest` This demo shows the use of several kinds of Tk widgets for user

input. Push buttons generate tones or noise, and sliders adjust the frequency and volume in real time.

`universe` This demo shows the movements of the Sun, Venus, Earth, and Mars in a Ptolemaic (Earth-centered) universe.

`xyplot` Demonstrate the `TkXYPlot` star.

14.3.22 BDF Demos

Figure 14-24 shows the palette of systems that demonstrate the use of BDF stars in the CGC domain. The `timing` demo is equivalent to the BDF simulation demo of the same name. The demos `bdf-if` and `bdf-doWhile` are equivalent to the BDF simulation demos named `ifThenElse` and `loop`. See “An overview of BDF demos” on page 8-3 for short descriptions of these demos.

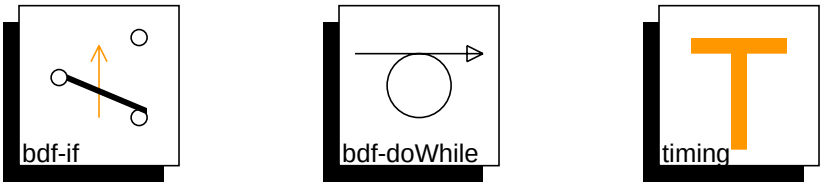


FIGURE 14-24: BDF demos in the CGC domain

14.3.23 Higher Order Function Demos

For information on the HOF demos, see “An overview of HOF demos” on page 6-18.

14.3.24 SDF-CGC Wormhole demos

Figure 14-25 shows that palette of systems that demonstrate the use of the `CreateS-`

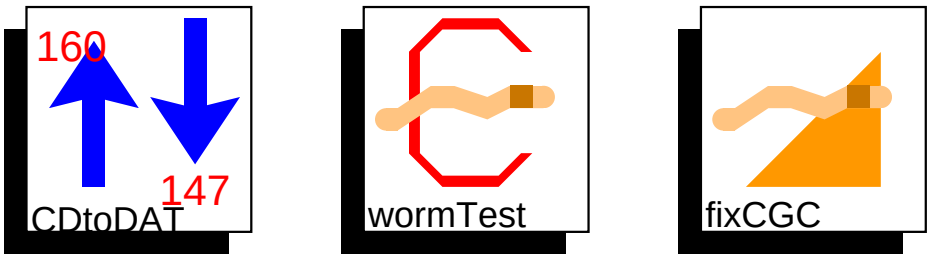


FIGURE 14-25: SDF-CGC Wormhole demos.

`DFStar` CGC target, which allows cgc stars that are reloaded back into Ptolemy for use inside the SDF domain. See “Interface Synthesis between Code Generation and Simulation Domains” on page 13-10 for more information about `CreateSDFStar`. The SDF-CGC Wormhole demos are found under the “Mixed Domain Demos” palette. The Mixed domain Demos palette is in the top level palette that is first visible when pigi starts up.

`CDtoDAT` Convert two sine waves sampled at CD sample rate to DAT sample rate. The outer galaxy is in the SDF domain, while the

	`cd2dat` galaxy is in the CGC domain. `cd2dat` uses the `CreateSDFStar` target.
`wormTest`	A simple test of the `CreateSDFStar` target.
`fixCGC`	Another simple test of the `CreateSDFStar` target.

14.3.25 UltraSparc VIS Demos

Figure 14-26 show the palette of systems that demonstrate the use of the Sun UltraSparc Visual Instruction Set demos.

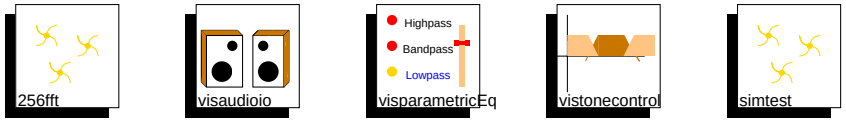


FIGURE 14-26: UltraSparc VIS demos in the CGC domain.

The Visual Instruction Set (VIS) demos only run on Sun UltraSparc workstations with the Sun unbundled CC compiler. The VIS demos will not compile with the Gnu compilers. Note that it is possible to generate VIS code if you don't have the Sun CC compiler, you just won't be able to compile it. You must have the Sun Visual Instruction Set Development kit installed, see `http://www.sun.com/sparc/vis/vsdkfaq.html`.

The VIS development kit and the CGC VIS stars require that following two environment variables be set:

```
setenv VSDKHOME /opt/SUNWvsdk
setenv INCASHOME /opt/SUNWincas
```

`256fft`	Plots the real and imaginary parts of a FFT. Note that this demo uses the CGC `Makefile_C` target and sets the *skeletonMakefile* target parameter to a special CGC VIS makefile at `$PTOLEMY/lib/cgc/makefile_VIS.mk`
`visaudioio`	Reads in audio from line-in and plays back from line-out. This demo uses the `makefile_TclTk_VIS.mk` file.
`parametricEQ`	Parametric equalizer.
`vistonecontrol`	Tone control using high, low and bandpass filters.
`simtest`	VIS simulator test universe. This demo illustrates a use of the `CGCVISSim` target. Note that this target is very experimental.

14.3.26 EECS20 demos

The Mixed domain demos palette also contains a palette of demos that were designed for EECS20, "Introduction to Real-Time Systems". These demos are used in a new lower-division course at UC Berkeley. For more information about this course, see `http://www-inst.eecs.berkeley.edu/~ee20`. The demos in this palette are in the CGC and CG56 domains. Most of these demos run on any Sparcstation with audio output. A few of the demos

require an S56X DSP card. At this time, these demos are not documented in this manual, see the individual demos on-line for documentation.

14.3.27 Tycho Demos

These demos demonstrate the use of the TychoTarget to create customized Control Panels. Graphical user interface widgets are used to control input and output parameters and to produce animation. The demos make use of the `TkStereoIn` and `TkStereoOut` (see “Tcl/Tk Stars” on page 14-13) to record and play sound on the workstation speaker, so these demos will probably only work on a Sun Ultrasparc. For information about how to use the audio capabilities of a workstation, see “Sounds” on page 2-38.

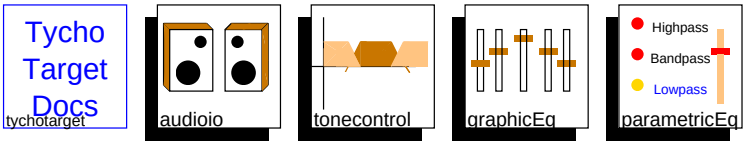


FIGURE 14-27: Tycho Target demos in the CGC domain

`audioio`	This is a simple real time audio demonstration which illustrates Ptolemy’s ability to support CD quality audio.
`graphicEq`	This demo consists of 10 band-pass filters with center frequencies spaced out by octaves. Using the customized control panel, you can adjust the gain of each band-pass filter, the record and play volumes and balance in real time.
`parametricEq`	In this demo, there is a single band of parametric equalisation, with control over the band frequency, band width, and band gain. The frequency range is settable; in the future, it will also be possible to select low-pass, band-pass, or high-pass filtering as well.
`tonecontrol`	The demo consists of one of each of the high, band and low-pass filters. There is a single control panel, with control over the band gain for each filter.