Chapter 17. C50 Domain

Authors: *Luis Gutierrez*

17.1 Introduction

The C50 domain generates assembly code for the Texas Instruments TMS320C5x series of digital signal processors. The graphs that we can describe in this domain follow the synchronous dataflow (SDF) model of computation. SDF allows us to schedule the `Blocks` and allocate all the resources at compile time. Refer to chapter "SDF Domain" on page 5-1 for a detailed description on the properties of SDF.

The TMS320C5x series are fixed-point digital signal processors which have 16 bit data and instructions and operate at a maximum rate of 50MIPS. The C50 domain has been tested on the TMS320C50 DSP Starter Kit board.

Since the C5x processors are fixed point, the floating point data type has no meaning in the C50 domain. Fixed-point values can take on the range [-1,1). The most positive value is $1 - 2^{-15}$. The domain defines a new constant `C50_ONE` set to this maximum positive value. In this chapter, whenever data types are not mentioned, fixed-point is meant. The complex data type means a pair of fixed-point numbers. The complex data type is supported for stars that have `anytype` inputs or outputs. Integers are the same length as the fixed-point representation. Matrix data types are not supported yet.

17.2 An overview of C50 stars

The "open-palette" command in pigi ("O") will open a checkbox window that you can use to open the standard palettes in all of the installed domains. For the C50 domain, the star library is large enough that it has been divided into sub-palettes as was done with the SDF main palette.

The top-level palette is shown in figure 17-1. The palettes are Signal Sources, I/O, Arithmetic, Nonlinear Functions, Logic, Control, Conversion, Signal Processing, and Higher Order Functions. The stars on the Higher Order Functions (HOF) palette are used to help lay out schematics graphically. The HOF stars are in the HOF domain, and not the C50 domain. Each palette is summarized in more detail below. More information about each star can be obtained using the on-line "profile" command (","), the on-line "man" command ("M"), or by looking in the *Star Atlas* volume of *The Almagest*.

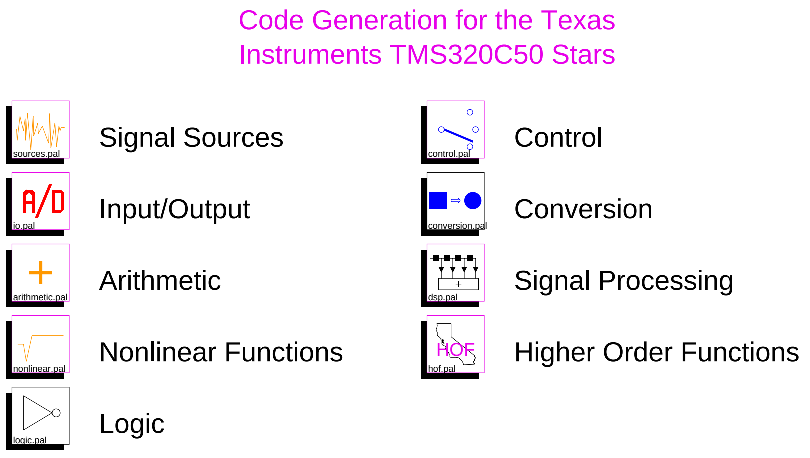

FIGURE 17-1: The palette of star palettes for the C50 domain.

At the top of each palette, for convenience, are instances of the delay icon, the bus icon, and the following star:

`BlackHole`	Discard all inputs. This star is useful for discarding signals that are not useful.

17.2.1 Source stars

Source stars are stars with only outputs. They generate signals, and may represent external inputs to the system, constant data, or synthesized stimuli. The palette of source stars

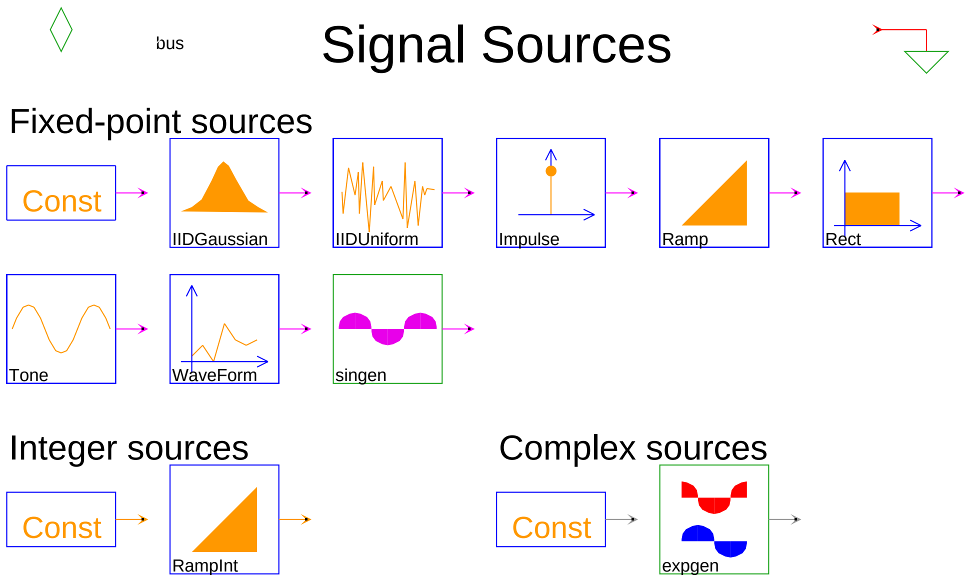

FIGURE 17-2: The palette of source stars for the C50 domain.

is shown in figure 17-2. Refer to 5.2.1 on page 5-5 for descriptions of the SDF equivalent stars: `Const`, `ConstCx`, `ConstInt`, `Ramp`, `RampInt`, `Rect`, `singen`, and `WaveForm`.

`Impulse`	Generate a single impulse or an impulse train. The size is determined by *impulseSize* (default ONE). If *period* (default is 0) is positive an impulse train with this period is generated, otherwise a single impulse is generated. If *delay* (default 0) is positive the impulse (or impulse train) is delayed by this amount.
`IIDUniform`	Generate an i.i.d. uniformly distributed pseudo-random process. Output is uniformly distributed between *-range* and *range* (default ONE).
`IIDGaussian`	Generate a white Gaussian pseudo-random process with mean 0 and standard deviation 0.1. A Gaussian distribution is realized by summing *noUniforms (default* 16) number of uniform random variables. According to the central limit theorem, the sum of N random variables approaches a Gaussian distribution as N approaches infinity.
`Tone`	Generate a sine or cosine wave using a second order oscillator. The wave will be of *amplitude* (default 0.5), *frequency* (default 0.2), and *calcType* (default “sin”)

17.2.2 I/O Stars

I/O stars are target specific stars that allow input and output of stimuli to a target architecture. Currently there are I/O stars only for the C50 DSK board so these stars should only be used with the DSKC50 target. These stars are located on the TI 320C5x IO palette inside the Input/Output palette.

`AIn`	This is an interrupt driven star to receive samples from the A/D converter in the Analog Interface Chip. The sample rate is determined by *sampleRate*. The actual conversion rate is 285.7KHz/N where N is an integer from 4 to 64. This star supports an internal buffer to hold the received samples. The size of this buffer can be set manually by changing the *interruptBufferSize* parameter. Setting *interruptBufferSize* to a negative value will set the size of the buffer equal to the number of times the star is fired on each iteration of the universe.
`AOut`	This is an interrupt driven star to send samples to the D/A converter in the AIC chip. The parameters are identical to those of the `AIn` star.

17.2.3 Arithmetic stars

The arithmetic stars that are available are shown in figure 17-4.

`Add`	(2 icons) Output the sum of the inputs. If *saturation* is set to yes, the output will saturate.
`Sub`	Outputs the “pos” input minus all of the “neg” inputs.
`Mpy`	(2 icons) Outputs the product of all of the inputs.
`Gain`	The output is set the input multiplied by a *gain* term. The gain must be in [-1,1).
`AddCx`	(2 icons) Output the complex sum of the inputs. If *saturation* is set to yes, the output will saturate.
`SubCx`	Outputs the “pos” input minus all of the “neg” inputs.
`MpyCx`	(2 icons) Outputs the product of all of the inputs.
`AddInt`	(2 icons) Output the sum of the inputs. If *saturation* is set to yes, the output will saturate.
`SubInt`	Outputs the “pos” input minus all of the “neg” inputs.

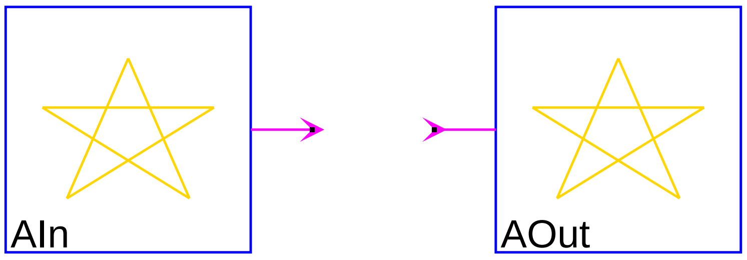

FIGURE 17-3: TI DSK 320C5x IO Palette

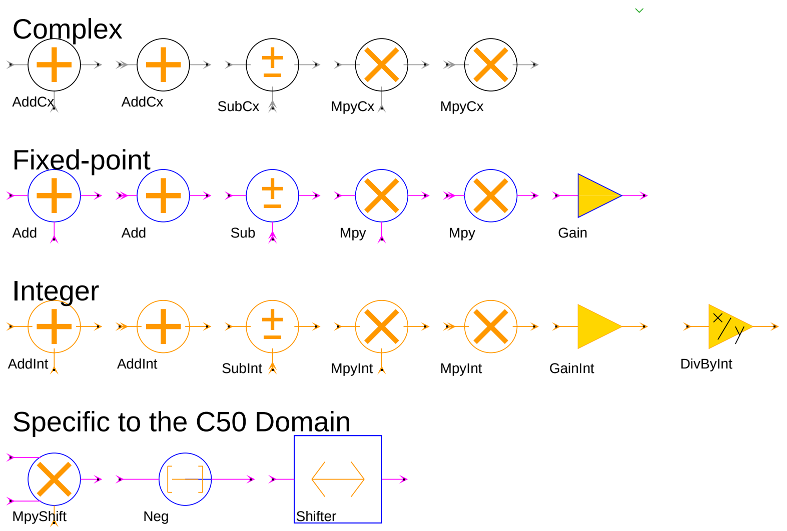

FIGURE 17-4: C50 Arithmetic Palette

`MpyInt`	(2 icons) Outputs the product of all of the inputs.
`GainInt`	The output is set the input multiplied by an integer *gain* term.
`DivByInt`	This is an amplifier. The integer output is the integer input divided by the integer *divisor* (default 2). Truncated integer division is used.
`MpyShift`	Multiply and shift.
`Neg`	Output the negation of the input.
`Shifter`	Scale by shifting left *leftShifts* bits. Negative values of *leftShifts* implies right shifting.

17.2.4 Nonlinear stars

The nonlinear palette (figure 17-5) in the C50 domain includes transcendental functions, quantizers, table lookup stars, and miscellaneous nonlinear functions.

`Abs`	Output the absolute value of the input.
`ACos`	Output the inverse cosine of the input, which is in the range -1.0 to 1.0. The output, in the principle range of 0 to π, is scaled

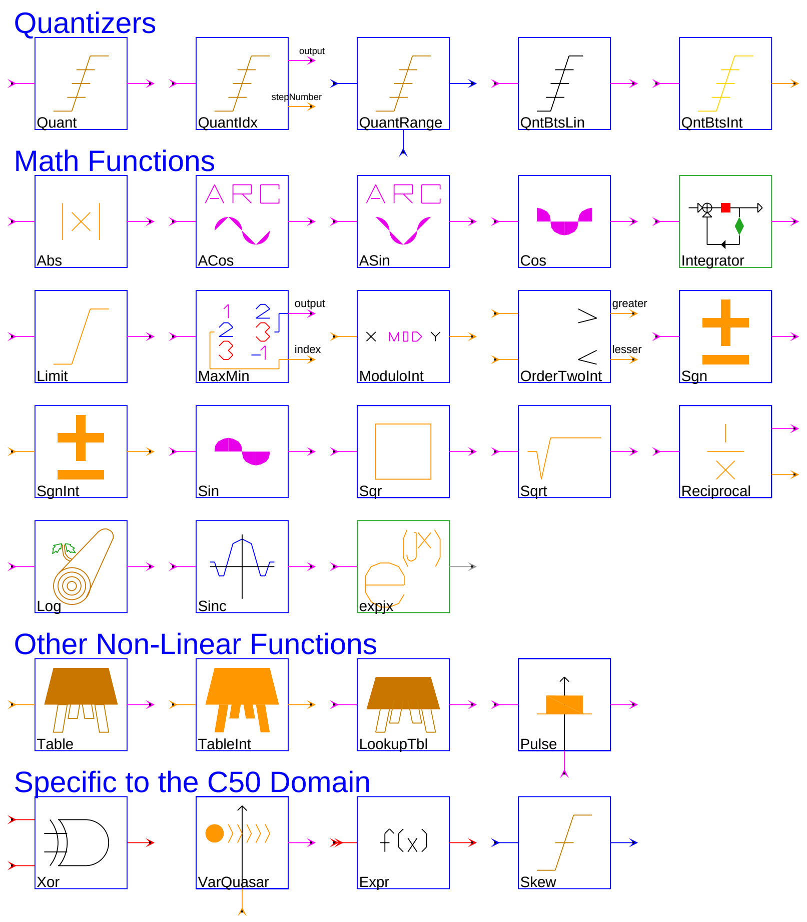

FIGURE 17-5: C50 Nonlinear Palette

	down by π.
`ASin`	Output the inverse sine of the input, which is in the range -1.0 to 1.0. The output, in the principle range of $-\frac{\pi}{2}$ to $\frac{\pi}{2}$, is scaled down by π.
`Cos`	Output the cosine, calculated the table lookup. The input range is [-1,1] scaled by π.
`expjx`	Output the complex exponential of the input.
`Intgrtr`	An integrator with leakage set by *feedbackGain*. If there is an overflow, the *onOverflow* parameter will designate a wrap around, saturate or reset operation.
`Limit`	Limits the input between the range of [*bottom*, *top*].
`Log`	Outputs the base two logarithm.
`MaxMin`	Output the maximal or minimal (*MAX*) sample out of the last *N* input samples. This can either *compareMagnitude* or take into account the sign. If *outputMagnitude* is `YES` the magnitude of the result is written to the output, otherwise the result itself is written.
`ModuloInt`	Output the remainder after dividing the integer input by the integer *modulo* parameter.
`OrderTwoInt`	Takes two inputs and outputs the greater and lesser of the two integers.
`Quant`	Quantizes the input to one of N+1 possible output *levels* using N *thresholds*.
`QuantIdx`	The star quantizes the input to one of N+1 possible output *levels* using N *thresholds*. It also outputs the index of the quantization level used.
`QuantRange`	Quantizes the input to one of N+1 possible output *levels* using N *thresholds*.
`Reciprocal`	Outputs the reciprocal to *Nf* precision in terms of a fraction and some left shifts.
`Sgn`	Outputs the sign of the input.
`SgnInt`	Outputs the sign of the integer input.
`Sin`	Outputs the sine, calculated using a table lookup. The input range is [-1,1) scaled by π.
`Sinc`	Outputs the sinc functions calculated as sin(x)/x.
`Sqrt`	Outputs the square root of the input.
`Table`	Implements a real-valued lookup table. The *values* state contains the values to output; its first element is element zero. An

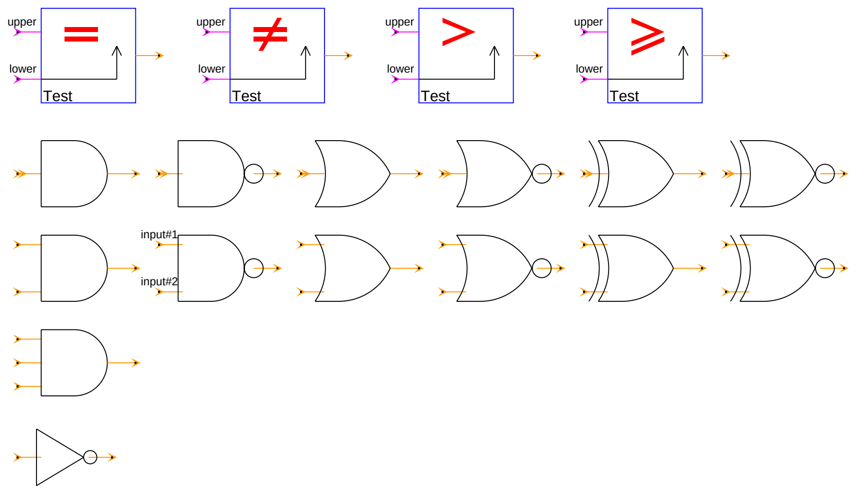

FIGURE 17-6: C50 Logic Palette

	error occurs if an out of bounds value is received.
`TableInt`	Implements an 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.
`Expr`	General expression evaluation.
`LookupTbl`	The input accesses a lookup table. The *interpolation* parameter determines the output for input values between table-entry points. If *interpolation* is "linear" the star will interpolate between table entries; if *interpolation* is set to "none", it will use the next lowest entry.
`Pulse`	Generates a variable length pulse. A pulse begins when a non-zero trigger is received. The pulse duration varies between 1 and *maxDuration* as the control varies between [-1,1).
`QntBtsInt`	Outputs the two's complement number given by the top *noBits* of the input (for integer output).
`QntBtsLin`	Outputs the two's complement number given by the top *noBits* of the input, but an optional *offset* can be added to shift the output levels up or down.
`Skew`	Generic skewing star.
`Sqr`	Outputs the square of the input.
`VarQuasar`	A sequence of values(*data*) is repeated at the output with period N (integer input), zero-padding or truncating the sequence to N if necessary. A value of O for N yields an aperiodic sequence.
`Xor`	Output the bit-wise exclusive-or of the inputs.

17.2.5 Logic stars

The Logic stars are discussed below:

`Test`	(4 icons) Test to see if two inputs are equal, not equal, greater than, and greater than or equal. For less than and less than or equal, switch the order of the inputs.
`And`	(3 icons) True if all inputs are non-zero.
`Nand`	(2 icons) True if all inputs are not non-zero.
`Or`	(2 icons) True if any input is non-zero.
`Nor`	(2 icons) True if any input is zero.
`Xor`	(2 icons) True if its inputs differ in value.
`Xnor`	(2 icons) True if its inputs coincide in value.
`Not`	Logical inverter.

17.2.6 Control stars

Control stars (figure 17-7) manipulate the flow of tokens. All of these stars are polymorphic; they operate on any data type. Refer to 5.2.6 on page 5-17 for descriptions of the SDF equivalent stars: `Fork`, `DownSample`, `Commutator`, `Distributor`, `Mux`, `Repeat`, `Reverse`, and `UpSample`.

`ChopVarOffset`	This star has the same functionality as the `Chop` star except now the *offset* parameter is determined at run time through a control input.
`Cut`	On each execution, this star reads a block of *nread* samples (default 128) and writes *nwrite* of these samples (default 64),

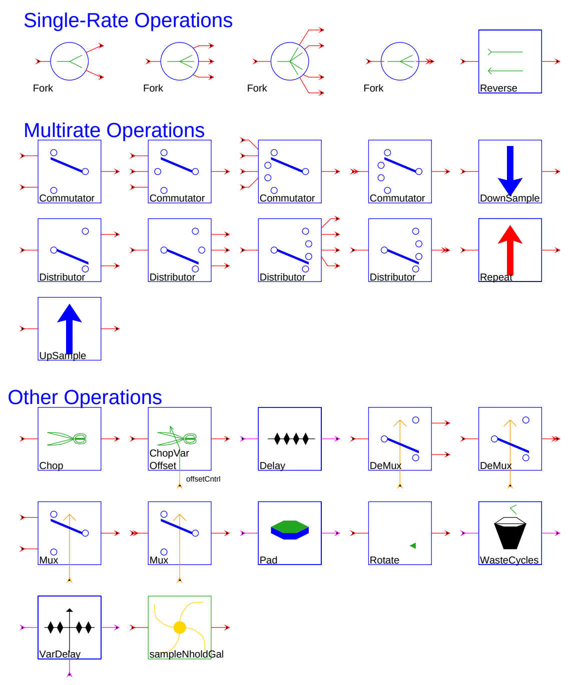

FIGURE 17-7: C50 Control Palette

	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.
`Delay`	A delay star of parameter *totalDelay* unit delays.
`Pad`	On each execution, Pad reads a block of *nread* samples and writes a block of *nwrite* samples. The first *offset* samples have value *fill*, the next *nread* output samples have values taken from the inputs, and the last *nwrite* - *nread* - *offset* samples have value *fill* again.
`Rotate`	The star reads in an input block of a certain *length* and performs a circular shift of the input. If the *rotation* is positive, the input is shifted to the left so that ouput[0] = input[*rotation*]. If the *rotation* is negative, the input is shifted to the right so that output[*rotation*] = input[0].
`sampleNholdGalaxy`	This sample-and-hold galaxy is more memory efficient than using a downsample star for the same purpose. This star is not present in Ptolemy0.6.
`VarDelay`	A variable delay that will vary between 0 and *maxDelay* as the control input varies between -1.0 and 1.0.
`WasteCycles`	Stalls the flow of data for *cyclesToWaste* number of cycles.

17.2.7 Conversion stars

The palette in figure 17-8 shows stars for format conversions from fixed point to complex fixed point.

`CxToRect`	Output the real part and imaginary part of the input of separate output ports.
`RectToCx`	Output a complex signal with real and imaginary part inputs.

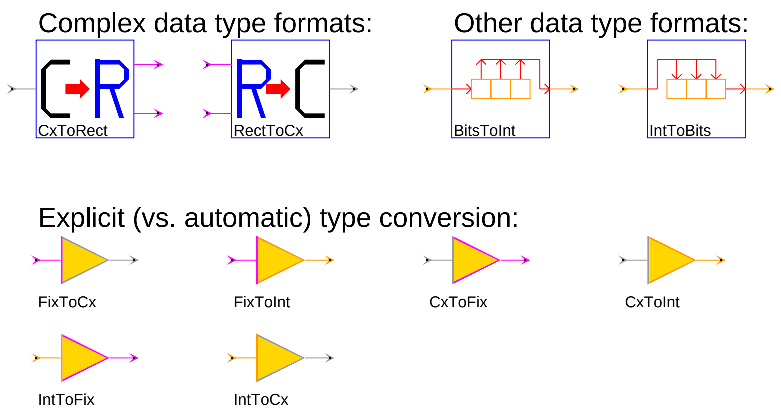

FIGURE 17-8: C50 Conversion Palette

`BitsToInt`	Convert a stream of bits to an integer.
`IntToBits`	Convert an integer into a stream of bits.
`FixToCx`	Convert fixed-point numbers to complex fixed-point numbers.
`FixToInt`	Convert fixed-point numbers to integer numbers.
`CxToFix`	Output the magnitude squared of the complex number.
`CxToInt`	Output the magnitude squared of the complex number.
`IntToFix`	Convert an integer input to a fixed point output.
`IntToCx`	Convert an integer input to a complex output.

17.2.8 Signal processing stars

The palette shown in figure 17-9 has icons for the library of signal processing functions. The filter stars follow. The `Goertzel` and `IIR` stars are identical to their SDF counterparts.

`Allpass`	An allpass filter with one pole and one zero. The location of these is given by the “polezero” input.
`Biquad`	A two-pole, two-zero IIR filter (a biquad).

$$H(z) = \frac{1 + n_1 z^{-1} + n_2 z^{-2}}{1 + d_1 z^{-1} + d_2 z^{-2}}$$

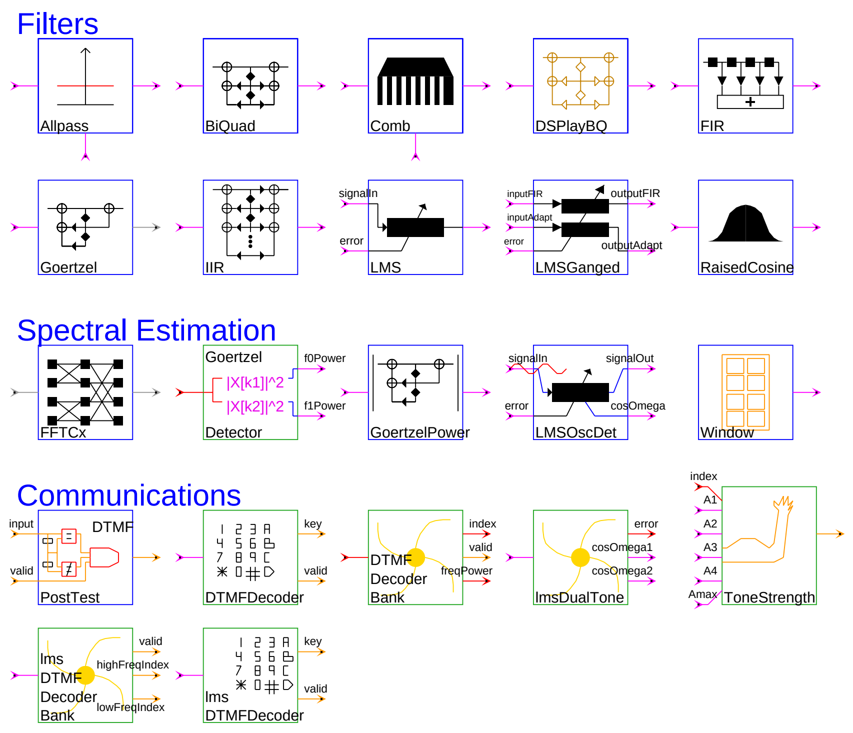

FIGURE 17-9: C50 Signal processing Palette

`Comb`	A comb filter with a one-pole lowpass filter in the delay loop.
`BiquadDSPlay`	A two-pole, two zero IIR filter (a biquad). This biquad is tailored to use the coefficients from the DSPlay filter design tool. If DSPlay gives the coefficients: A B C D E then define the parameters as follows: a=A, b=B, c=C, d=-(D+1), e = -E. This only works if a, b, c, d, and e, are in the range [-1,1). The default coefficients implement a low pass filter. $$H(z) = \frac{a + bz^{-1} + cz^{-2}}{1 - (d+1)z^{-1} - ez^{-2}}$$
`FIR`	A finite impulse response (FIR) filter. Coefficients are specified by the *taps* parameter. The default coefficients give an 8th order, linear-phase, lowpass filter. To read coefficients from a file, replace the default coefficients with `< filename`, preferably specifying a complete path. Polyphase multirate filtering is not yet supported.
`LMS`	An adaptive filter using the LMS adaptation algorithm. The initial coefficients are given by the *coef* parameter. The default initial coefficients give an 8th order, linear phase lowpass filter. To read default coefficients from a file, replace the default coefficients with `< filename`, preferably specifying a complete path. This star supports decimation, but not interpolation.
`LMSGanged`	A LMS filter were the coefficients from the adaptive filter are used to run a FIR filter in parallel. The initial coefficients default to a lowpass filter of order 8.
`RaisedCos`	An FIR filter with a magnitude frequency response shaped like the standard raised cosine used in digital communications.See the `SDFRaisedCosine` star for more information.

The spectral estimation stars follow. The `GoertzelDetector, GoertzelPower`, and `LMSOscDet` are identical to their SDF counterparts.

`FFTCx`	Compute the discrete-time Fourier transform of a complex input using the fast Fourier transform (FFT) algorithm. The parameter *order* (default 8) is the transform size. The parameter *direction* (default 1) is 1 for forward, -1 for the inverse FFT.
`Window`	Generate standard window functions or periodic repetitions of standard window functions. The possible functions are `Rectangle`, `Bartlett`, `Hanning`, `Hamming`, `Blackman`, `SteepBlackman`, and `Kaiser`. One period of samples is produced on each firing.

The communications stars are exactly like their SDF counterparts.

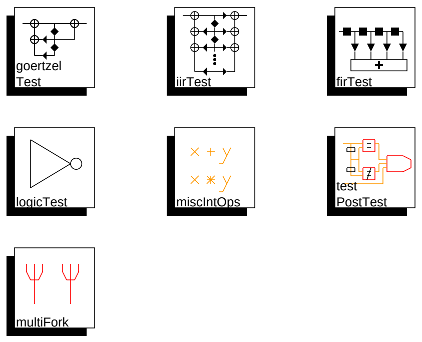

FIGURE 17-10: Basic/Test Demo Palette

17.3 An overview of C50 Demos

A set of C50 demonstration programs have been developed. The demos are meant to be run on the C50DSK board. If you do not have the required DSK tools, then you can still run the demos to see the generated code. To do this make sure that the *run* and *compile* target parameters are to `NO`. By default, the generated code is written to `$HOME/PTOLEMY_SYSTEMS/C50` directory.

17.3.1 Basic/Test demos

The Basic/Test palette contains 7 demonstrations.

`goertzelTest`	Test the Goertzel filters for computing the discrete Fourier transform.
`firTest`	Test the finite impulse response (FIR) filters.
`iirTest`	Test the infinite impulse response (IIR) filters.
`logicTest`	Test various comparison tests and Boolean functions.
`miscIntOps`	Test integer arithmetic operations.
`multiFork`	Test the `AnyAsmFork` star. An `AnyAsmFork` star is one of a group of stars that do produce any code at compile time.
`testPostTest`	Test the `DTMFPostTest` star used in touchtone decoding.

17.3.2 DSK 320C5x demos

The DSK 320C5x demo palette contains demonstrations meant to be run on the Texas Instruments DSP Starter Kit board.

`chirp`	This system uses two integrators and a cosine to generate a chirp signal.
`DTMFCodec`	Demonstration of touchtone detection using the discrete Fourier transform implemented by using Goertzel filters.
`lms`	A noise source is connected to an eighth-order least-mean squares (LMS) adaptive filter with initial taps specifying a low-pass filter. The taps adapt to a null filter (the impulse response is

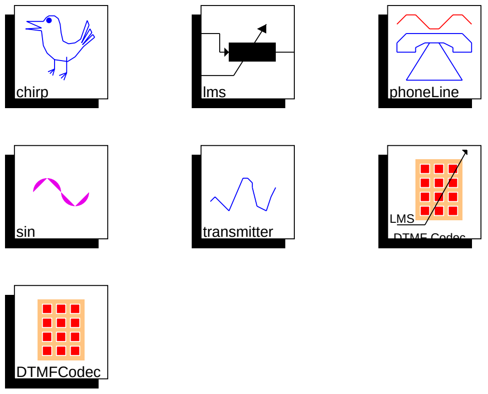

FIGURE 17-11: DSK 320C5x Palette

	an impulse) and the error signal is displayed.
`lmsDTMFCodec`	Demonstration of touchtone detection using Normalized Direct Frequency Estimation implemented by using Least-Mean Squares (LMS) adaptive filters.
`phoneLine`	A telephone channel simulator. A tone is passed through some processing which implements various distortions on a telephone channel. The parameters that are controllable are: noise, channel filter, second harmonic, third harmonic, frequency offset, phase jitter frequency, and phase jitter amplitude.
`sin`	A sine wave is generated by using two integrators in a feedback loop.
`transmitter`	A simple 4-level PAM transmitter

17.4 Targets

Three C50 targets are included in the Ptolemy distribution. To choose one of these targets, with your mouse cursor in a schematic window, execute the Edit:edit-target command (or just type “T”). You will get a list of the available `Targets` in the C50 domain. The `default-C50` target is the default value. When you click `OK`, the dialog box appears with the parameters of the target. You can edit these, or accept the defaults. The next time you run the schematic, the selected target will be used.

17.4.1 Default C50 (default-C50) target

The default target is used only for code generation. It has the following set of options:

host	(`STRING`) Default = The default is the empty string. Host machine to compile or assemble code on. All code is written to and compiled and run on the computer specified by this parameter. If a remote computer is specified here then `rsh` commands are used to place files on that computer and to invoke the compiler. You should verify that your .rhosts file is properly configured so that `rsh`

	will work.
directory	(`STRING`) Default = `$HOME/PTOLEMY_SYSTEMS/C50` This is the directory to which all generated files will be written to.
file	(`STRING`) Default = The default is the empty string. This represents the prefix for file names for all generated files.
Looping Level	Specifies if the loop scheduler should be used. Please refer to the section “default-CG” on page 13-2 for more details on this option. Refer to “Default SDF target” on page 5-65 and “The loop-SDF target” on page 5-67 for more details on loop scheduling.
display?	(`INT`) Default = `YES` If this flag is set to `YES`, then the generated code will be displayed on the screen.
compile?	This is a dummy flag since the default target only generates code.
run?	This is a dummy flag since the default target only generates code.
bMemMap	(`STRING`) Default = `768-1279` Address range for C50 Dual Access RAM blocks. C50 Instructions that operate on data run faster if the data is stored in one of the DARAM blocks. Disjoint segments of memory can be specified by separating the contiguous ranges with spaces, e.g. “768-800 1200-1279.”
uMemMap	(`STRING`) Default = `2432-6848` Data address range in the C50 Single Access RAM block. This can also specify a valid address range in external memory.
subroutines?	(`INT`) Default = `-1` Setting this parameter to N makes the target attempt to generate a subroutine instead of in-line code for a star if the number of repetitions of that star is greater than N (use N=0 to generate subroutines even for stars with just 1 repetition). Set *subroutines?* to `-1` (or any other negative integer) to disable the feature.

17.4.2 C50 Subroutine (sub-C50) target

This target is used to generate subroutines that can be called from hand-written C50 code. The options are identical to those of `default-C50` target.

17.4.3 C50 DSP Starter Kit (DSKC50) target

This target is used to generate C50 code to be run on Texas Instruments’ DSP Starter

Kit board. In addition to the regular `file.asm` generated by the other targets, this target will produce a second file (`fileDSK.asm`) which is the same as the original file but with all lines truncated to 80 characters. This is done because the TI DSK assembler will give false error messages if lines in the input file exceed 80 characters. The options are identical to those of `default-C50` target with four exceptions:

compile?	If this flag is set the target will issue the command `asmc50 fileDSK.asm` where `fileDSK.asm` is the name of the file containing the generated code. This should run the DSK assembler and produce a file `fileDSK.dsk`. Note that `asmc50` can be a shell script that invokes the user's DSK assembler. Scripts to use the TI DSK assembler and loader in Linux are presented at the end of this section.
run?	If this flag is set the target will issue the command `loadc50 fileDSK.dsk` which should load `fileDSK.dsk` to the DSK board. Note that `loadc50` can be a shell script that invokes the user's DSK loader.
bMemMap	(`STRING`) Default = `768-1270` Valid addresses on the Dual Access RAM block 1. The last 9 words in this (addresses 1271 - 1279) are reserved by the target to store configuration information for the Analog Interface Chip.
uMemMap	(`STRING`) Default = `2432-6847` Valid addresses on the Single Access RAM memory. Locations 6848 - 11263 are reserved to store the user's program and locations 2048-2431 are reserved by the TI DSK debugger kernel.

The following scripts invoke the TI DSK assembler and loader from Linux through `dosemu` (a DOS emulator). Note that before invoking the assembler and loader Ptolemy executes a cd to the *directory* target parameter. Since you need to unmount the DOS partition to run `dosemu` you can not have *directory* set to the DOS partition. One solution is to set *directory* to your home directory and set *file* to include the path to the directory where you want the file written. For example, if your home directory is `/ptuser`, the dos partition `dosemu` will use is `/dos/c` and you want the output files written to `/dos/c/dsk/src` the you could set *directory* to `/users/ptdesign` and *file* to `/dos/c/dsk/filename` where `filename` is the name of the output file. These scripts are also included in `$PTOLEMY/src/domains/c50.`

```
#!/bin/sh
# Version: @(#)asmc501.604/07/97
# Copyright (c) 1996-1997 The Regents of the University of California.
# All Rights Reserved.
#
# asmc50
# script to assemble files with TI's DSK assembler(dsk5a.exe)
# Uses dosemu to run dsk5a.exe.  The person running it must be root to
# mount/unmount the dos partition.
# This script was tested on a machine running linux (red-hat 3.0.3
# distribution) with dosemu-0.63.1.33 installed.
#
# Written by Luis Gutierrez.
# Converted from csh to sh by Brian L. Evans

# User’s home directory.
homedir=/root

# User’s dos partition.
dospartition=/dos/c

# The root path of DOS drive where DSK files and DOS binaries are
stored.
dosroot=c:

# The DOS directory(relative to dosroot)where the *.asm and *.dsk
files
# are stored. Replace the \ in the DOS path with \\.
dsksrc=dsk\\src

# The DOS directory(relative to dosroot) where the DSK
# executables(dsk5a.exe, dsk5l.exe) are stored.
# Replace the \ in the DOS path with \\.
dskbin=dsk

# The file used to temporarily save autoexec.emu.
autoexecsave=autoexec.bak

cd $dospartition
mv autoexec.emu $autoexecsave

# The text between the first xxxx and the second xxxx will be
# piped to unix2dos and will end up in autoexec.emu.

unix2dos > $dospartition/autoexec.emu << xxxx
path $dosroot\\$dskbin;$dosroot\\dos
cd $dosroot\\$dsksrc
dsk5a.exe $1:t
exitemu
xxxx
cd $homedir

# Unmount DOS partition to run dosemu
umount $dospartition
dos > /dev/null

# Mount DOS partition after running dosemu
mount -t msdos /dev/sda1 $dospartition

# Restore autoexec.emu
cd $dospartition
mv -f $dospartition/$autoexecsave  $dospartition/autoexec.emu
```

The following script is used to load files.

```
#!/bin/csh
# Version: @(#)loadc501.5 03/29/97
# Copyright (c) 1996-1997 The Regents of the University of California.
# All Rights Reserved.
#
# loadc50
# script to load files with TI’s DSK loader(dsk5l.exe)
# Uses xdos to run dsk5l.exe.  The person running it must be root to
# mount/unmount the dos partition.
# This script was tested on a machine running linux(red-hat 3.0.3
# diistribution) with dosemu-0.63.1.33 installed.
# Written by Luis Gutierrez.
#
# Converted from csh to sh by Brian L. Evans

# User’s home directory.
homedir=/root

# User’s dos partition.
dospartition=/dos/c

# The root path of DOS drive where DSK files and DOS binaries are
stored.
dosroot=c:

# The DOS directory(relative to dosroot\)where the *.asm and *.dsk
files
# are stored. Replace the \ in the DOS path with \\.
dsksrc=dsk\\src

# The DOS directory(relative to dosroot) where the DSK
# executables(dsk5a.exe, dsk5l.exe) are stored.
# Replace the \ in the DOS path with \\.
dskbin=dsk

# The file used to temporarily save autoexec.emu
autoexecsave=autoexec.bak

cd $dospartition
mv autoexec.emu $autoexecsave

# The text between the first xxxx and the second xxxx will be
# piped to unix2dos and will end up in autoexec.emu.

unix2dos  > $dospartition/autoexec.emu << xxxx
path $dosroot\\$dskbin;$dosroot\\dos
cd $dosroot\\$dsksrc
dsk5l.exe  $1:t
exitemu
xxxx
cd $homedir

# Unmount DOS partition to run xdos
```

```
umount $dospartition
xdos

# After running xdos mount DOS partition
mount -t msdos /dev/sda1 $dospartition

# Restore autoexec.emu
cd $dospartition
mv -f $dospartition/$autoexecsave  $dospartition/autoexec.emu
```