mu.psi
Connectable RT Signal Processing Blocks

Macro	Operands	Inputs	Outputs	Description
ada	fs	da1, da2	ad1, ad2	Analog input / output. Wait for sample. fs is in Hertz
angle	*name*	*name*_in	*name*	Arc Tangent function
cnorm	*name*	*name*_in	*name*	Bring a complex to module 1
compare	*name*,{gt|ge|eq|ne|le|lt},oT,oF	*name*_in,*name*_ref	*name*	Compare input to reference
decibel	*name*	*name*_in	*name*	Output = Log(input)/20 = 0 for 0dB and -1.0 for -200dB
delay	*name*,value,{abs|rel}	*name*_in	*name*	Implement delay line
divide	*name*	*name*_num, *name*_den	*name*	Divide variables output = num / den (|num| < |den|)
f_cos	*name*,points	*name*_in	*name*	Cosine function: output = Cos(input / π)
f_sin	*name*,points	*name*_in	*name*	Sine function: output = Sin(input / π)
f_sincos	*name,* points	*name*_in	*name*	Complex exponential function: name = exp(j.in / π)
fir	*name*,file	*name*_in	*name*	Implement FIR filter. Impulse response is given in *file*
fir2	*name*,size,flo,fhi,{abs|rel}	*name*_in	*name*	Bandpass or lowpass FIR filter, h(n) by windowing method
fitf2i	*name*,imin,imax	*name*_in	*name*	Adapt fract to integer so that -1→ imin +1 → imax
fiti2f	*name*,imin,imax	*name*_in	*name*	Adapt integer to fract so that imin →-1 imax → +1
g_bpr	*name*,bits	-	*name*	Pseudo random binary sequence generator
g_chirp	*name*,fr1,fr2,frep,{abs|rel}	-	*name*	Real or complex Chirp generator
g_gauss	*name,sigma[,N]*	-	*name*	Random number generator, gaussian distribution
g_noise	*name*	-	*name*	Random number generator, uniform distribution within [-1..+1[
g_rect	*name,*freq	-	*name*	Square wave generator
g_saw	*name,*freq,{abs|rel}	-	*name*	Sawtooth generator
g_sawd	*name,*freq,{abs|rel}	-	*name* (d)	Sawtooth generator, double precision
g_sin	*name*,freq,{abs|rel}	-	*name*	Sine wave generator
g_tri	*name,*freq,{abs|rel}	-	*name*	Triangle generator
gain	*name*,value	*name*_in	*name*	Constant gain (value may be > to 1.)
hilbert	*name*,points	*name*_in	*name* (c)	π/2 phase shifting filter. Complex output (analytic signal)
hp1	*name*, freq,{abs|rel}	*name*_in	*name*	1st order highpass filter
iir	*name*,b0,b1,b2,a1,a2	*name*_in	*name*	2nd order IIR filter
iir2	*name*,typ, freq,q, {abs|rel}	*name*_in	*name*	2nd order IIR filter with coeffs; typ = { lp | bp | hp | bs}
iirc1	*name*,freq,band, {abs|rel}	*name*_in (c)	*name* (c)	Complex 1 pole bandpass filter
integ	*name*,gain	*name*_in	*name*	Integrator with input gain
lp1	*name*, freq,{abs|rel}	*name*_in	*name*	1st order lowpass filter
lpabs	*name*, freq,{abs|rel}	*name*_in	*name*	Output = 1st order lowpass filtered absolute value of input
man	*name*,max,unit	-	*name*	Manual entry through RS232 port
mul	*name*	*name*_in1, *name*_in2	*name*	Multiplication
mulc	*name*	*name*_in1(c), *name*_in2 (c)	*name* (c)	Complex multiplication
mulcc	*name*	*name*_in1(cd) *name*_in2(cd)	*name* (cd)	Complex multiplication with cojuguate
oscpq	*name*, freq,{abs|rel}	-	*name* (c)	Complex oscillator
poly	name,'file'	*name*_in	*name*	Polynom. 'file' contains coefficients in ascending powers
pwm	name,n,per,min,max	*name*_in	_	Implement PWM on timer *n*. *n* = 0,1,2. Wait for PWM period Durations = *per, min, max* are in seconds in_pwm*n* = -1..+1
slopelim	name,slope[,slope_n]	*name*_in	*name*	Slope limiter (non linear filter)
snds	name	-	com_busy (f)	Send ASCII string *name* to com port. Uses 1 DMA channel
specan	fs,size,{pow|amp|dB},{r|c} ,{h|f}[,'file']	specan_in, da2	ad1, ad2	RT spectrum analyser on size points. Scope out on da2. If present, the program 'file' is executed at sampling rate
sqroot	*name*	*name*_in [d]	*name*	Square root. Output is 12b / 24b, if input is single / double
str **strcr** **strln**	*string*,'text'	-	-	Create / append ASCII string strcr = terminated by <cr> strlf = terminated by <cr / lf>
strd	*name,string*,digits,scaling	*name*_in	-	Convert fractional variable to decimal string
strh	*name,string*,digits,msb	*name*_in	-	Convert 4-24b data to hex string
stri	*name,string*,digits	*name*_in	-	Convert integer variable to decimal string
strnh	*name,string*, words,digits,msb,sep	*name*_in(n)	-	Convert multiple 4-24b data to hex string
tblrd	*name*,'file',{u|s},	*name*_in	*name*	Table read and interpolate. *file* contains table values
tblr2d	*name*,'file',xsize,{u|s},{u|s}	*name*_inx *name*_iny	*name*	2D table read and interpolate. *file* contains table values ordered in successive rows
vosc	samples,digits,var1,..var8	*1..8 variables*	-	Virtual oscilloscope
wsum2 **wsum3**	*name*,gain1,gain2 *name*,gain1,gain2,gain3	*name*_in1, .._in2,..in3	*name*	Make weighted sum of inputs. 2 inputs or 3 inputs Gains are constants between -1.0 and +1.0

mu.psi
Timing and Program Flow Control Macros

Macro	Operands	Inputs	Outputs	Description
clock	frequency	-	-	Change DSP clock frequency. Range: 12.8kHz .. 120MHz
flagclr	flagname	-	-	Reserve flag if not defined. Clear flag.
flagset	flagname	-	-	Reserve flag if not defined. Set flag.
flagtog	flagname	-	-	Reserve flag if not defined. Toggle flag.
goto	label	-	-	Jump to label
irq	irqname,level,file	-	-	Install and activate interrupt. Execute task 'file' on occurence
irqada	fs,file	da1, da2	ad1, ad2	At each sampling time do analog i/o and execute task *'file'*.
irqpwm	name,n,per,min,max,file	*name*_ in	_	PWM on timer *n*. (*n* = 0, 1, 2) Execute *'file'* at each period. Duration *per* ,*min*, *max* are in seconds. in_*name* = -1 .. +1
jcf	flagname,label	-	-	Test-and-clear, jump if flag was set
jcnf	flagname,label	-	-	Test-and-clear, jump if flag was clear
jf	flagname,label	-	-	Jump if flag is set
jnf	flagname,label	-	-	Jump if flag is clear
jtf	flagname,label			Toggle flag. Jump if flag was set
jtnf	flagname,label			Toggle flag. Jump if flag was clear
timerf	name,frequency,{abs|rel}	[*name*_freq]	*name* (flag) [*name*_acc]	Software periodic timer defined by a frequency. Adds freq to acc at each sample. Sets the flag on acc overflow
timerp	name,period	[*name*_per]	*name* (flag) [cntr_*name*]	Software periodic timer. Decrements cntr at each invocation. When cntr reaches 0, sets flag *name* and reloads cntr.
timers	name[,duration]	[*name*_dur]	*name* (flag) [*name*_cntr]	Software single shot timer. Starts with *timers name, duration* Dec cntr with *timers name*. Sets flag and stops when cntr=0
waicf	flag_name			Wait until flag set, clear flag

mu.psi
Data Management Macros

Macro	Operands	Description
bufm	name,field,size	Reserve a non connectable Modulo buffer. Field may be x, y, or l.
bufr	name,field,size	Reserve a reverse carry buffer. Field may be x, y, or l.
cn	source,destination	Connection: define destination as identical to source. If source isn't defined, reserve variable
cnb	source,destination	Connect boolean data
coef	name,value	Create a coefficient in memory
coefc	name,ro,alpha,p name,re,im,c	Create a complex coefficient in memory Arguments may be in polar or cartesian form
coefd	name,value	Create a double (48b) coefficient in memory
flagr	name	Reserve a flag in memory
ptr	name,n,Rn,Mn,Nn	Reserve and initialize a memory pointer
ptru	name	Update pointer
regin	_	Activate register input mode in following block
saverest	name	Reserve room for interrupt context save and restore, implement save and restore routines
twiddle	name,points	Generate complex exponential table for FFT
u	register	Declare which registers are used (thus modified) in current macro
var	name[,value]	Reserve a variable in memory (with optional initial value)
varc	name[,ro,alpha,p] name[,re,im,c]	Reserve a complex variable in memory (with optional initial value)
varcd	name[,ro,alpha,p] name[,re,im,c]	Reserve a complex double variable in memory (with optional initial value)
vard	name[,value]	Reserve a double (48b) variable in memory (with optional initial value)
varin	_	Inhibit register input mode in following block (default)

mu.psi
Debugging macros

Macro	Operands	Description
cc_ac	n	Accumulate n times current cycle count to timing result (n is <> 1 for loops only)
cc_add	m	Add value m to timing result
cc_disa		Disable cycle counting
cc_ena		Enable cycle counting
cc_off		Disable cycle counting
cc_rst		Reset current cycle count to 0 (inside a block)
cc_start		Reset timing result to 0 and enable time counting
cc_view	string	Display the timing result
dbg	*name*	View the current fractional value of a variable (debug mode)
dbgf	*flag_name*	View the current state of a flag (debug mode)
dbgi	*name*	View the current integer value of a variable (debug mode)
dbgw	string	Stop execution and display "string",LF,CR (debug mode)
logg	*name*,size,field	Copy a variable to (memory field) circular buffer (for debug purpose)