

Audio Processor definition and implementation documentation

Stefan Jaritz

July 10, 2013

Contents

1	HAL	1
1.1	Variables	1
1.1.1	rational (HAL vid=1)	1
1.1.2	integer (HAL vid=2)	1
1.1.3	string (HAL vid=3)	1
1.1.4	complex (HAL vid=4)	2
1.1.5	biquad (HAL vid=10)	2
1.1.6	noisegate (HAL vid=11)	2
1.1.7	expander (HAL vid=12)	3
1.1.8	compressor (HAL vid=13)	3
1.1.9	limiter (HAL vid=14)	3
1.1.10	delay (HAL vid=20)	3
1.1.11	FFT (HAL vid=30)	3
1.1.12	sixxxx (HAL vid=66)	4
1.1.13	panel (HAL vid=100)	4
1.1.14	button (HAL vid=101)	4
1.1.15	led (HAL vid=102)	4
1.1.16	display (HAL vid=103)	4
1.2	Functions	5
1.2.1	genTestSignal (HAL fid =3)	5
1.2.2	setStringSize (HAL fid =5)	5
1.2.3	setStringValue (HAL fid =6)	5
1.2.4	concatStrings (HAL fid =7)	6
1.2.5	rationalToString (HAL fid =8)	7
1.2.6	integerToString (HAL fid =9)	7
1.2.7	assignString (HAL fid =10)	8
1.2.8	assignConstInteger (HAL fid =20)	8
1.2.9	assignInteger (HAL fid =21)	9
1.2.10	addInteger (HAL fid =22)	10
1.2.11	subInteger (HAL fid =23)	10

1.2.12	mulInteger (HAL fid =24)	10
1.2.13	divInteger (HAL fid =25)	11
1.2.14	modInteger (HAL fid =26)	11
1.2.15	assignConstRational (HAL fid =27)	12
1.2.16	assignRational (HAL fid =28)	13
1.2.17	addRational (HAL fid =29)	13
1.2.18	subRational (HAL fid =30)	13
1.2.19	mulRational (HAL fid =31)	14
1.2.20	divRational (HAL fid =32)	14
1.2.21	modRational (HAL fid =33)	15
1.2.22	compareRationalLess (HAL fid =34)	15
1.2.23	compareRationalMore (HAL fid =35)	16
1.2.24	compareRationalEqual (HAL fid =36)	16
1.2.25	compareRationalNEqual (HAL fid =37)	17
1.2.26	compareRationalLessEqual (HAL fid =38)	17
1.2.27	compareRationalMoreEqual (HAL fid =39)	18
1.2.28	compareIntegerLess (HAL fid =40)	18
1.2.29	compareIntegerMore (HAL fid =41)	19
1.2.30	compareIntegerEqual (HAL fid =42)	19
1.2.31	compareIntegerNEqual (HAL fid =43)	20
1.2.32	compareIntegerLessEqual (HAL fid =44)	20
1.2.33	compareIntegerMoreEqual (HAL fid =45)	21
1.2.34	jump (HAL fid =50)	21
1.2.35	jumpCF (HAL fid =51)	22
1.2.36	jumpNCF (HAL fid =52)	22
1.2.37	setCF (HAL fid =55)	22
1.2.38	update (HAL fid =56)	23
1.2.39	readSample (HAL fid =60)	23
1.2.40	writeSample (HAL fid =61)	23
1.2.41	readSampleFrame (HAL fid =62)	24
1.2.42	writeSampleFrame (HAL fid =63)	24
1.2.43	initBiquadAsHP (HAL fid =100)	24
1.2.44	initBiquadAsLP (HAL fid =101)	25
1.2.45	initBiquadAsPeakFilter (HAL fid =102)	25
1.2.46	initBiquadAsLowFreqShelvFilter (HAL fid =103)	26
1.2.47	initBiquadAsHighFreqShelvFilter (HAL fid =104)	27
1.2.48	convoluteBiquad (HAL fid =110)	28
1.2.49	initNoisegate (HAL fid =111)	28
1.2.50	initExpander (HAL fid =112)	29
1.2.51	initCompressor (HAL fid =113)	30
1.2.52	initLimiter (HAL fid =114)	31

1.2.53	calcNoisegate (HAL fid =115)	32
1.2.54	calcExpander (HAL fid =116)	33
1.2.55	calcCompressor (HAL fid =117)	33
1.2.56	calcLimiter (HAL fid =118)	33
1.2.57	initHannWindow (HAL fid =120)	34
1.2.58	initRFFT (HAL fid =130)	34
1.2.59	initIFFT (HAL fid =131)	35
1.2.60	processRFFT (HAL fid =132)	35
1.2.61	processIFFT (HAL fid =133)	36
1.2.62	calcDelay (HAL fid =150)	36
1.2.63	initDelay (HAL fid =151)	36
1.2.64	uiSetDim (HAL fid =200)	37
1.2.65	uiInitPanel (HAL fid =201)	38
1.2.66	uiInitButton (HAL fid =202)	38
1.2.67	uiInitDisplay (HAL fid =203)	39
1.2.68	uiInitLED (HAL fid =204)	39
1.2.69	uiCheckButtonPressed (HAL fid =210)	40
1.2.70	uiSetLED (HAL fid =211)	40
1.2.71	uiSetDisplay (HAL fid =212)	41
2	Communication Interface	42
2.1	Message Frames	42
2.1.1	ACK (msgId =1)	42
2.1.2	NACK (msgId =2)	42
2.1.3	startPrg (msgId =10)	42
2.1.4	sendVariable (msgId =11)	43
2.1.5	sendInstruction (msgId =12)	43
2.1.6	endPrg (msgId =13)	43
2.1.7	stop (msgId =20)	43
2.1.8	step (msgId =21)	43
2.1.9	run (msgId =22)	44
2.1.10	updateVariable (msgId =23)	44
2.1.11	login (msgId =30)	44
2.1.12	logout (msgId =31)	44
2.2	Message Processes	44
2.2.1	logout (msgProcId =0)	44
2.2.2	updateVariable (msgProcId =0)	45
2.2.3	login (msgProcId =10)	45
2.2.4	run (msgProcId =20)	46
2.2.5	run (msgProcId =22)	47
2.2.6	updateVariable (msgProcId =23)	47

2.2.7	login (msgProcId =30)	48
2.2.8	logout (msgProcId =31)	48
3	Implementations	50
3.1	Implementation groups	50
3.1.1	ADSP 21369 blockbased, ADC in, DAC3 & DAC4 out, UART @9600,n,8,1	50
3.1.2	ANSI C strings	75
3.1.3	AP client interface using stjSocket and APclient func- tions	80
3.1.4	MSP430-169STK	92
3.1.5	audio dynamic processing (generic)	106
3.1.6	biquad filters (generic)	122
3.1.7	boost	136
3.1.8	fftw3 & complex	136
3.1.9	generic delay	136
3.1.10	gtk+ for Windows	144
3.1.11	libsndfile frame based	164
3.1.12	libsndfile overlapped frame based	170
3.1.13	libsndfile sample based	176
3.1.14	pthreads and semaphores	181
3.1.15	rational and integer generic	182
3.1.16	no group	189
3.2	Implementation of the HAL Variables	189
3.2.1	rational (rational and integer generic)	189
3.2.2	integer (rational and integer generic)	190
3.2.3	complex (fftw3 & complex)	192
3.2.4	FFT (fftw3 & complex)	194
3.2.5	panel (MSP430-169STK)	195
3.2.6	button (MSP430-169STK)	196
3.2.7	led (MSP430-169STK)	198
3.2.8	display (MSP430-169STK)	199
3.2.9	panel (gtk+ for Windows)	200
3.2.10	button (gtk+ for Windows)	201
3.2.11	led (gtk+ for Windows)	202
3.2.12	display (gtk+ for Windows)	203
3.2.13	string (ANSI C strings)	204
3.2.14	biquad (biquad filters (generic))	207
3.2.15	noisegate (audio dynamic processing (generic))	209
3.2.16	expander (audio dynamic processing (generic))	210
3.2.17	compressor (audio dynamic processing (generic))	211

3.2.18	limiter (audio dynamic processing (generic))	212
3.2.19	delay (generic delay)	213
3.3	Implementation of the HAL functions	215
3.3.1	jump (no group)	215
3.3.2	jumpCF (no group)	216
3.3.3	jumpNCF (no group)	216
3.3.4	setCF (no group)	217
3.3.5	update (no group)	218
3.3.6	assignConstInteger (rational and integer generic)	218
3.3.7	assignInteger (rational and integer generic)	219
3.3.8	addInteger (rational and integer generic)	220
3.3.9	subInteger (rational and integer generic)	220
3.3.10	mulInteger (rational and integer generic)	221
3.3.11	divInteger (rational and integer generic)	222
3.3.12	modInteger (rational and integer generic)	223
3.3.13	assignConstRational (rational and integer generic) . . .	223
3.3.14	assignRational (rational and integer generic)	224
3.3.15	addRational (rational and integer generic)	225
3.3.16	subRational (rational and integer generic)	226
3.3.17	mulRational (rational and integer generic)	226
3.3.18	divRational (rational and integer generic)	227
3.3.19	modRational (rational and integer generic)	228
3.3.20	compareRationalLess (rational and integer generic) . .	228
3.3.21	compareRationalMore (rational and integer generic) . .	229
3.3.22	compareRationalEqual (rational and integer generic) .	230
3.3.23	compareRationalNEqual (rational and integer generic)	231
3.3.24	compareRationalLessEqual (rational and integer generic)	232
3.3.25	compareRationalMoreEqual (rational and integer generic)	233
3.3.26	compareIntegerLess (rational and integer generic) . . .	233
3.3.27	compareIntegerMore (rational and integer generic) . . .	234
3.3.28	compareIntegerEqual (rational and integer generic) . .	235
3.3.29	compareIntegerNEqual (rational and integer generic) .	236
3.3.30	compareIntegerLessEqual (rational and integer generic)	237
3.3.31	compareIntegerMoreEqual (rational and integer generic)	238
3.3.32	readSample (libsndfile sample based)	239
3.3.33	writeSample (libsndfile sample based)	239
3.3.34	initRFFT (fftw3 & complex)	240
3.3.35	initIFFT (fftw3 & complex)	241
3.3.36	processRFFT (fftw3 & complex)	242
3.3.37	processIFFT (fftw3 & complex)	242
3.3.38	readSampleFrame (libsndfile overlapped frame based) .	243

3.3.39	writeSampleFrame (libsndfile overlapped frame based)	244
3.3.40	readSampleFrame (ADSP 21369 blockbased, ADC in, DAC3 & DAC4 out, UART @9600,n,8,1)	245
3.3.41	writeSampleFrame (ADSP 21369 blockbased, ADC in, DAC3 & DAC4 out, UART @9600,n,8,1)	246
3.3.42	readSampleFrame (libsndfile frame based)	247
3.3.43	writeSampleFrame (libsndfile frame based)	248
3.3.44	uiSetDim (MSP430-169STK)	248
3.3.45	uiInitPanel (MSP430-169STK)	249
3.3.46	uiInitButton (MSP430-169STK)	249
3.3.47	uiInitDisplay (MSP430-169STK)	251
3.3.48	uiInitLED (MSP430-169STK)	252
3.3.49	uiCheckButtonPressed (MSP430-169STK)	253
3.3.50	uiSetLED (MSP430-169STK)	254
3.3.51	uiSetDisplay (MSP430-169STK)	254
3.3.52	uiSetDim (gtk+ for Windows)	255
3.3.53	uiInitPanel (gtk+ for Windows)	256
3.3.54	uiInitButton (gtk+ for Windows)	257
3.3.55	uiInitDisplay (gtk+ for Windows)	258
3.3.56	uiInitLED (gtk+ for Windows)	259
3.3.57	uiCheckButtonPressed (gtk+ for Windows)	260
3.3.58	uiSetLED (gtk+ for Windows)	261
3.3.59	uiSetDisplay (gtk+ for Windows)	261
3.3.60	setStringSize (ANSI C strings)	262
3.3.61	setStringValues (ANSI C strings)	263
3.3.62	concatStrings (ANSI C strings)	264
3.3.63	rationalToString (ANSI C strings)	265
3.3.64	integerToString (ANSI C strings)	265
3.3.65	assignString (ANSI C strings)	266
3.3.66	initBiquadAsHP (biquad filters (generic))	267
3.3.67	initBiquadAsLP (biquad filters (generic))	268
3.3.68	initBiquadAsPeakFilter (biquad filters (generic))	269
3.3.69	initBiquadAsLowFreqShelvFilter (biquad filters (generic))	270
3.3.70	initBiquadAsHighFreqShelvFilter (biquad filters (generic))	271
3.3.71	convoluteBiquad (biquad filters (generic))	272
3.3.72	initNoisegate (audio dynamic processing (generic))	273
3.3.73	initExpander (audio dynamic processing (generic))	273
3.3.74	initCompressor (audio dynamic processing (generic))	274
3.3.75	initLimiter (audio dynamic processing (generic))	275
3.3.76	calcNoisegate (audio dynamic processing (generic))	276
3.3.77	calcExpander (audio dynamic processing (generic))	277

3.3.78	calcCompressor (audio dynamic processing (generic))	. 278
3.3.79	calcLimiter (audio dynamic processing (generic))	. . . 278
3.3.80	calcDelay (generic delay) 279
3.3.81	initDelay (generic delay) 280
3.4	message system drivers 281
3.4.1	driver 1 (ADSP 21369 blockbased, ADC in, DAC3 & DAC4 out, UART @9600,n,8,1) 281
3.4.2	driver 2 (MSP430-169STK) 285
3.4.3	driver 3 (AP client interface useing stjSocket and AP-client functions) 289
3.5	audio processor blueprints 295
3.5.1	audio processor blueprint 1 (libsndfile sample based)	. 295
3.5.2	audio processor blueprint 2 (libsndfile overlapped frame based) 317
3.5.3	audio processor blueprint 3 (ADSP 21369 blockbased, ADC in, DAC3 & DAC4 out, UART @9600,n,8,1)	. . . 338
3.5.4	audio processor blueprint 4 (libsndfile frame based)	. . 361
3.5.5	audio processor blueprint 5 (MSP430-169STK) 382
3.5.6	audio processor blueprint 6 (pthreads and semaphores)	406

Chapter 1

HAL

1.1 Variables

1.1.1 rational (HAL vid=1)

Informations:

HAL variable id:	1
description:	a rational number
AL ASM syntax:	rational test [3];

sub variables:

Nr.	Name	Description	Type
1	value	the value of the number	rational

1.1.2 integer (HAL vid=2)

Informations:

HAL variable id:	2
description:	a integer number
AL ASM syntax:	integer test [3];

sub variables:

Nr.	Name	Description	Type
1	value	the value of the number	integer

1.1.3 string (HAL vid=3)

Informations:

HAL variable id:	3
description:	a string
AL ASM syntax:	string test [3];

sub variables:

Nr.	Name	Description	Type
1	length	the length of the string	integer
2	text	the text of the string	char (array)

1.1.4 complex (HAL vid=4)

Informations:

HAL variable id:	4
description:	a complex number
AL ASM syntax:	complex test [3];

sub variables:

Nr.	Name	Description	Type
1	r	real part	rational
2	i	imaginary part	integer

1.1.5 biquad (HAL vid=10)

Informations:

HAL variable id:	10
description:	a biquad filter
AL ASM syntax:	biquad test [3];

sub variables:

Nr.	Name	Description	Type
1	n0	enumerator coefficient	rational
2	n1	enumerator coefficient	rational
3	n2	enumerator coefficient	rational
4	d0	denominator coefficient	rational
5	d1	denominator coefficient	rational
6	d2	denominator coefficient	rational

1.1.6 noisegate (HAL vid=11)

Informations:

HAL variable id:	11
description:	a noisegate
AL ASM syntax:	noisegate test [3];

1.1.7 expander (HAL vid=12)

Informations:

HAL variable id:	12
description:	a expander
AL ASM syntax:	expander test [3];

1.1.8 compressor (HAL vid=13)

Informations:

HAL variable id:	13
description:	a compressor
AL ASM syntax:	compressor test [3];

1.1.9 limiter (HAL vid=14)

Informations:

HAL variable id:	14
description:	a limiter
AL ASM syntax:	limiter test [3];

1.1.10 delay (HAL vid=20)

Informations:

HAL variable id:	20
description:	a delay
AL ASM syntax:	delay test [3];

sub variables:

Nr.	Name	Description	Type
1	values	the values ordered oldest to youngest	rational (array)

1.1.11 FFT (HAL vid=30)

Informations:

HAL variable id:	30
description:	FFT or IFFT sturcture
AL ASM syntax:	FFT test [3];

sub variables:

Nr.	Name	Description	Type
1	N	number of samples	integer

1.1.12 sixxx (HAL vid=66)

Informations:

HAL variable id:	66
description:	six six six
AL ASM syntax:	sixxx test [3];

sub variables:

Nr.	Name	Description	Type
1	dataL	the data list	rational (array)
2	keffi	koefficient	rational
3	indx	index	integer
4	array2	array numero 2	integer (array)

1.1.13 panel (HAL vid=100)

Informations:

HAL variable id:	100
description:	panel UI
AL ASM syntax:	panel test [3];

1.1.14 button (HAL vid=101)

Informations:

HAL variable id:	101
description:	button UI
AL ASM syntax:	button test [3];

1.1.15 led (HAL vid=102)

Informations:

HAL variable id:	102
description:	led UI
AL ASM syntax:	led test [3];

1.1.16 display (HAL vid=103)

Informations:

HAL variable id:	103
description:	display UI
AL ASM syntax:	display test [3];

1.2 Functions

1.2.1 genTestSignal (HAL fid =3)

Informations:

HAL function id:	3
description:	generate a test signal, witch can be used to meassure performance
AL ASM syntax:	genTestSignal value;

Parameters:

Nr.	Name	Description	Class	Type
1	value	a value	basic io type	basic type integer

1.2.2 setStringSize (HAL fid =5)

Informations:

HAL function id:	5
description:	resets the size of a string
AL ASM syntax:	setStringSize s,i,length;

Parameters:

Nr.	Name	Description	Class	Type
1	s	the string	reference to the data of a variable	HAL variable type string
2	i	index of the string at the array	basic io type	basic type integer
3	length	length of the string	basic io type	basic type integer

1.2.3 setStringValues (HAL fid =6)

Informations:

HAL function id:	6
description:	set the string
AL ASM syntax:	setStringValues s,i,p,v1,v2,v3,v4;

Parameters:

Nr.	Name	Description	Class	Type
1	s	the string	reference to the data of a variable	HAL variable type string
2	i	index of the string at the array	basic io type	basic type integer
3	p	position at the string where to start from	basic io type	basic type integer
4	v1	chars coded as 4 byte integer	basic io type	basic type raw
5	v2	chars coded as 4 byte integer	basic io type	basic type raw
6	v3	chars coded as 4 byte integer	basic io type	basic type raw
7	v4	chars coded as 4 byte integer	basic io type	basic type raw

1.2.4 concatStrings (HAL fid =7)

Informations:

HAL function id:	7
description:	concat two strings
AL ASM syntax:	concatStrings s1,i1,s2,i2;

Parameters:

Nr.	Name	Description	Class	Type
1	s1	the string at its end the other string is concated	reference to the data of a variable	HAL variable type string
2	i1	index of the string 1	basic io type	basic type integer
3	s2	the concat string	reference to the data of a variable	HAL variable type string
4	i2	index of the string 2	basic io type	basic type integer

1.2.5 rationalToString (HAL fid =8)

Informations:

HAL function id:	8
description:	converts a rational to a string
AL ASM syntax:	rationalToString s,sIndex,r,rIndx;

Parameters:

Nr.	Name	Description	Class	Type
1	s	the string	reference to the data of a variable	HAL variable type string
2	sIndex	index of the string at the array	basic io type	basic type integer
3	r	rational vector	reference to the data of a variable	HAL variable type rational
4	rIndx	rational vector index	basic io type	basic type integer

1.2.6 integerToString (HAL fid =9)

Informations:

HAL function id:	9
description:	converts an integer to a string
AL ASM syntax:	integerToString s,sIndex,i,iIndex;

Parameters:

Nr.	Name	Description	Class	Type
1	s	the string	reference to the data of a variable	HAL variable type string
2	sIndex	index of the string at the array	basic io type	basic type integer
3	i	integer vector	reference to the data of a variable	HAL variable type integer
4	iIndex	integer vector indx	basic io type	basic type integer

1.2.7 assignString (HAL fid =10)

Informations:

HAL function id:	10
description:	assigns a string to an other
AL ASM syntax:	assignString s1,i1,s2,i2;

Parameters:

Nr.	Name	Description	Class	Type
1	s1	the string at its end the other string is concated	reference to the data of a variable	HAL variable type string
2	i1	index of the string 1	basic io type	basic type integer
3	s2	the concat string	reference to the data of a variable	HAL variable type string
4	i2	index of the string 2	basic io type	basic type integer

1.2.8 assignConstInteger (HAL fid =20)

Informations:

HAL function id:	20
description:	a = values
AL ASM syntax:	assignConstInteger iv,iStart,num,v1,v2,v3,v4;

Parameters:

Nr.	Name	Description	Class	Type
1	iv	integer vector	reference to the data of a variable	HAL variable type integer
2	iStart	start index	basic io type	basic type integer
3	num	amount of values used	basic io type	basic type integer
4	v1	value 1	basic io type	basic type integer
5	v2	value 2	basic io type	basic type integer
6	v3	value 3	basic io type	basic type integer
7	v4	value 4	basic io type	basic type integer

1.2.9 assignInteger (HAL fid =21)

Informations:

HAL function id:	21
description:	a = b
AL ASM syntax:	assignInteger a,b;

Parameters:

Nr.	Name	Description	Class	Type
1	a	vector a	reference to the data of a variable	HAL variable type integer
2	b	vector b	reference to the data of a variable	HAL variable type integer

1.2.10 addInteger (HAL fid =22)

Informations:

HAL function id:	22
description:	$c = a + b$
AL ASM syntax:	addInteger a,b,c;

Parameters:

Nr.	Name	Description	Class	Type
1	a	vector a	reference to the data of a variable	HAL variable type integer
2	b	vector b	reference to the data of a variable	HAL variable type integer
3	c	vector c	reference to the data of a variable	HAL variable type integer

1.2.11 subInteger (HAL fid =23)

Informations:

HAL function id:	23
description:	$c = a - b$
AL ASM syntax:	subInteger a,b,c;

Parameters:

Nr.	Name	Description	Class	Type
1	a	vector a	reference to the data of a variable	HAL variable type integer
2	b	vector b	reference to the data of a variable	HAL variable type integer
3	c	vector c	reference to the data of a variable	HAL variable type integer

1.2.12 mulInteger (HAL fid =24)

Informations:

HAL function id:	24
description:	$c = a * b$
AL ASM syntax:	mulInteger a,b,c;

Parameters:

Nr.	Name	Description	Class	Type
1	a	vector a	reference to the data of a variable	HAL variable type integer
2	b	vector b	reference to the data of a variable	HAL variable type integer
3	c	vector c	reference to the data of a variable	HAL variable type integer

1.2.13 divInteger (HAL fid =25)

Informations:

HAL function id:	25
description:	$c = a / b$
AL ASM syntax:	divInteger a,b,c;

Parameters:

Nr.	Name	Description	Class	Type
1	a	vector a	reference to the data of a variable	HAL variable type integer
2	b	vector b	reference to the data of a variable	HAL variable type integer
3	c	vector c	reference to the data of a variable	HAL variable type integer

1.2.14 modInteger (HAL fid =26)

Informations:

HAL function id:	26
description:	$c = \text{mod}(a,b)$
AL ASM syntax:	modInteger a,b,c;

Parameters:

Nr.	Name	Description	Class	Type
1	a	vector a	reference to the data of a variable	HAL variable type integer
2	b	vector b	reference to the data of a variable	HAL variable type integer
3	c	vector c	reference to the data of a variable	HAL variable type integer

1.2.15 assignConstRational (HAL fid =27)

Informations:

HAL function id:	27
description:	a = values
AL ASM syntax:	assignConstRational iv,iStart,num,v1,v2,v3,v4;

Parameters:

Nr.	Name	Description	Class	Type
1	iv	rational vector	reference to the data of a variable	HAL variable type rational
2	iStart	start index	basic io type	basic type integer
3	num	amount of values used	basic io type	basic type integer
4	v1	value 1	basic io type	basic type rational
5	v2	value 2	basic io type	basic type rational
6	v3	value 3	basic io type	basic type rational
7	v4	value 4	basic io type	basic type rational

1.2.16 assignRational (HAL fid =28)

Informations:

HAL function id:	28
description:	$a = b$
AL ASM syntax:	assignRational a,b;

Parameters:

Nr.	Name	Description	Class	Type
1	a	vector a	reference to the data of a variable	HAL variable type rational
2	b	vector b	reference to the data of a variable	HAL variable type rational

1.2.17 addRational (HAL fid =29)

Informations:

HAL function id:	29
description:	$c = a + b$
AL ASM syntax:	addRational a,b,c;

Parameters:

Nr.	Name	Description	Class	Type
1	a	vector a	reference to the data of a variable	HAL variable type rational
2	b	vector b	reference to the data of a variable	HAL variable type rational
3	c	vector c	reference to the data of a variable	HAL variable type rational

1.2.18 subRational (HAL fid =30)

Informations:

HAL function id:	30
description:	$c = a - b$
AL ASM syntax:	subRational a,b,c;

Parameters:

Nr.	Name	Description	Class	Type
1	a	vector a	reference to the data of a variable	HAL variable type rational
2	b	vector b	reference to the data of a variable	HAL variable type rational
3	c	vector c	reference to the data of a variable	HAL variable type rational

1.2.19 mulRational (HAL fid =31)

Informations:

HAL function id:	31
description:	$c = a * b$
AL ASM syntax:	mulRational a,b,c;

Parameters:

Nr.	Name	Description	Class	Type
1	a	vector a	reference to the data of a variable	HAL variable type rational
2	b	vector b	reference to the data of a variable	HAL variable type rational
3	c	vector c	reference to the data of a variable	HAL variable type rational

1.2.20 divRational (HAL fid =32)

Informations:

HAL function id:	32
description:	$c = a / b$
AL ASM syntax:	divRational a,b,c;

Parameters:

Nr.	Name	Description	Class	Type
1	a	vector a	reference to the data of a variable	HAL variable type rational
2	b	vector b	reference to the data of a variable	HAL variable type rational
3	c	vector c	reference to the data of a variable	HAL variable type rational

1.2.21 modRational (HAL fid =33)

Informations:

HAL function id:	33
description:	$c = \text{mod}(a,b)$
AL ASM syntax:	modRational a,b,c;

Parameters:

Nr.	Name	Description	Class	Type
1	a	vector a	reference to the data of a variable	HAL variable type rational
2	b	vector b	reference to the data of a variable	HAL variable type rational
3	c	vector c	reference to the data of a variable	HAL variable type rational

1.2.22 compareRationalLess (HAL fid =34)

Informations:

HAL function id:	34
description:	$a < b ? CF = 1 : CF = 0$
AL ASM syntax:	compareRationalLess a,ia,b,ib;

Parameters:

Nr.	Name	Description	Class	Type
1	a	left side number	reference to the data of a variable	HAL variable type rational
2	ia	index at a vector	basic io type	basic type integer
3	b	right side number	reference to the data of a variable	HAL variable type rational
4	ib	index at b vector	basic io type	basic type integer

1.2.23 compareRationalMore (HAL fid =35)

Informations:

HAL function id:	35
description:	$a > b ? CF = 1 : CF = 0$
AL ASM syntax:	compareRationalMore a,ia,b,ib;

Parameters:

Nr.	Name	Description	Class	Type
1	a	left side number	reference to the data of a variable	HAL variable type rational
2	ia	index at a vector	basic io type	basic type integer
3	b	right side number	reference to the data of a variable	HAL variable type rational
4	ib	index at b vector	basic io type	basic type integer

1.2.24 compareRationalEqual (HAL fid =36)

Informations:

HAL function id:	36
description:	$a == b ? CF = 1 : CF = 0$
AL ASM syntax:	compareRationalEqual a,ia,b,ib;

Parameters:

Nr.	Name	Description	Class	Type
1	a	left side number	reference to the data of a variable	HAL variable type rational
2	ia	index at a vector	basic io type	basic type integer
3	b	right side number	reference to the data of a variable	HAL variable type rational
4	ib	index at b vector	basic io type	basic type integer

1.2.25 compareRationalNEqual (HAL fid =37)

Informations:

HAL function id:	37
description:	$a \neq b ? CF = 1 : CF = 0$
AL ASM syntax:	compareRationalNEqual a,ia,b,ib;

Parameters:

Nr.	Name	Description	Class	Type
1	a	left side number	reference to the data of a variable	HAL variable type rational
2	ia	index at a vector	basic io type	basic type integer
3	b	right side number	reference to the data of a variable	HAL variable type rational
4	ib	index at b vector	basic io type	basic type integer

1.2.26 compareRationalLessEqual (HAL fid =38)

Informations:

HAL function id:	38
description:	$a \leq b ? CF = 1 : CF = 0$
AL ASM syntax:	compareRationalLessEqual a,ia,b,ib;

Parameters:

Nr.	Name	Description	Class	Type
1	a	left side number	reference to the data of a variable	HAL variable type rational
2	ia	index at a vector	basic io type	basic type integer
3	b	right side number	reference to the data of a variable	HAL variable type rational
4	ib	index at b vector	basic io type	basic type integer

1.2.27 compareRationalMoreEqual (HAL fid =39)

Informations:

HAL function id:	39
description:	$a \geq b ? CF = 1 : CF = 0$
AL ASM syntax:	compareRationalMoreEqual a,ia,b,ib;

Parameters:

Nr.	Name	Description	Class	Type
1	a	left side number	reference to the data of a variable	HAL variable type rational
2	ia	index at a vector	basic io type	basic type integer
3	b	right side number	reference to the data of a variable	HAL variable type rational
4	ib	index at b vector	basic io type	basic type integer

1.2.28 compareIntegerLess (HAL fid =40)

Informations:

HAL function id:	40
description:	$a < b ? CF = 1 : CF = 0$
AL ASM syntax:	compareIntegerLess a,ia,b,ib;

Parameters:

Nr.	Name	Description	Class	Type
1	a	left side number	reference to the data of a variable	HAL variable type integer
2	ia	index at a vector	basic io type	basic type integer
3	b	right side number	reference to the data of a variable	HAL variable type integer
4	ib	index at b vector	basic io type	basic type integer

1.2.29 compareIntegerMore (HAL fid =41)

Informations:

HAL function id:	41
description:	$a > b ? CF = 1 : CF = 0$
AL ASM syntax:	compareIntegerMore a,ia,b,ib;

Parameters:

Nr.	Name	Description	Class	Type
1	a	left side number	reference to the data of a variable	HAL variable type integer
2	ia	index at a vector	basic io type	basic type integer
3	b	right side number	reference to the data of a variable	HAL variable type integer
4	ib	index at b vector	basic io type	basic type integer

1.2.30 compareIntegerEqual (HAL fid =42)

Informations:

HAL function id:	42
description:	$a == b ? CF = 1 : CF = 0$
AL ASM syntax:	compareIntegerEqual a,ia,b,ib;

Parameters:

Nr.	Name	Description	Class	Type
1	a	left side number	reference to the data of a variable	HAL variable type integer
2	ia	index at a vector	basic io type	basic type integer
3	b	right side number	reference to the data of a variable	HAL variable type integer
4	ib	index at b vector	basic io type	basic type integer

1.2.31 compareIntegerNEqual (HAL fid =43)

Informations:

HAL function id:	43
description:	a != b ? CF = 1 : CF = 0
AL ASM syntax:	compareIntegerNEqual a,ia,b,ib;

Parameters:

Nr.	Name	Description	Class	Type
1	a	left side number	reference to the data of a variable	HAL variable type integer
2	ia	index at a vector	basic io type	basic type integer
3	b	right side number	reference to the data of a variable	HAL variable type integer
4	ib	index at b vector	basic io type	basic type integer

1.2.32 compareIntegerLessEqual (HAL fid =44)

Informations:

HAL function id:	44
description:	a <= b ? CF = 1 : CF = 0
AL ASM syntax:	compareIntegerLessEqual a,ia,b,ib;

Parameters:

Nr.	Name	Description	Class	Type
1	a	left side number	reference to the data of a variable	HAL variable type integer
2	ia	index at a vector	basic io type	basic type integer
3	b	right side number	reference to the data of a variable	HAL variable type integer
4	ib	index at b vector	basic io type	basic type integer

1.2.33 compareIntegerMoreEqual (HAL fid =45)

Informations:

HAL function id:	45
description:	a >= b ? CF = 1 : CF = 0
AL ASM syntax:	compareIntegerMoreEqual a,ia,b,ib;

Parameters:

Nr.	Name	Description	Class	Type
1	a	left side number	reference to the data of a variable	HAL variable type integer
2	ia	index at a vector	basic io type	basic type integer
3	b	right side number	reference to the data of a variable	HAL variable type integer
4	ib	index at b vector	basic io type	basic type integer

1.2.34 jump (HAL fid =50)

Informations:

HAL function id:	50
description:	jumps n instructions
AL ASM syntax:	jump number;

Parameters:

Nr.	Name	Description	Class	Type
1	number	amount of instructions to jump	basic io type	basic type integer

1.2.35 jumpCF (HAL fid =51)

Informations:

HAL function id:	51
description:	jumps if the carry flag is set n instructions
AL ASM syntax:	jumpCF number;

Parameters:

Nr.	Name	Description	Class	Type
1	number	amount of instructions to jump	basic io type	basic type integer

1.2.36 jumpNCF (HAL fid =52)

Informations:

HAL function id:	52
description:	jumps if the carry flag is not set n instructions
AL ASM syntax:	jumpNCF number;

Parameters:

Nr.	Name	Description	Class	Type
1	number	amount of instructions to jump	basic io type	basic type integer

1.2.37 setCF (HAL fid =55)

Informations:

HAL function id:	55
description:	sets the carry flag
AL ASM syntax:	setCF value;

Parameters:

Nr.	Name	Description	Class	Type
1	value	the value of the CF	basic io type	basic type integer

1.2.38 update (HAL fid =56)

Informations:

HAL function id:	56
description:	updates a global variable
AL ASM syntax:	update var;

Parameters:

Nr.	Name	Description	Class	Type
1	var	the variable (internally the index of the variable)	variable index	unknown(error)

1.2.39 readSample (HAL fid =60)

Informations:

HAL function id:	60
description:	reading a sample from an input
AL ASM syntax:	readSample channel,resValue;

Parameters:

Nr.	Name	Description	Class	Type
1	channel	the channel	basic io type	basic type integer
2	resValue	the result of the action	reference to the data of a variable	HAL variable type rational

1.2.40 writeSample (HAL fid =61)

Informations:

HAL function id:	61
description:	writes a sample to a output
AL ASM syntax:	writeSample channel,value;

Parameters:

Nr.	Name	Description	Class	Type
1	channel	the channel	basic io type	basic type integer
2	value	the value to be written to the output	reference to the data of a variable	HAL variable type rational

1.2.41 readSampleFrame (HAL fid =62)

Informations:

HAL function id:	62
description:	reading a frames of sample from an input
AL ASM syntax:	readSampleFrame channel,frameBuffer,waitForNewFrame;

Parameters:

Nr.	Name	Description	Class	Type
1	channel	the channel	basic io type	basic type integer
2	frameBuffer	the buffer witch receives the samples	reference to the data of a variable	HAL variable type rational
3	waitForNewFrame	if not zero the function waits for a new sample frame	basic io type	basic type integer

1.2.42 writeSampleFrame (HAL fid =63)

Informations:

HAL function id:	63
description:	writes a frame of samples to a output
AL ASM syntax:	writeSampleFrame channel,frameBuffer;

Parameters:

Nr.	Name	Description	Class	Type
1	channel	the channel	basic io type	basic type integer
2	frameBuffer	the buffer which is written to the channel	reference to the data of a variable	HAL variable type rational

1.2.43 initBiquadAsHP (HAL fid =100)

Informations:

HAL function id:	100
description:	inits a biquad filter as an high pass filter
AL ASM syntax:	initBiquadAsHP bq,index,fs,fc;

Parameters:

Nr.	Name	Description	Class	Type
1	bq	biquad cascade	reference to variable	HAL variable type biquad
2	index	index at the cascade	basic io type	basic type integer
3	fs	sample frequency	reference to the data of a variable	HAL variable type rational
4	fc	cut off frequency	reference to the data of a variable	HAL variable type rational

1.2.44 initBiquadAsLP (HAL fid =101)

Informations:

HAL function id:	101
description:	inits a biquad filter as a low pass filter
AL ASM syntax:	initBiquadAsLP bq,index,fs,fc;

Parameters:

Nr.	Name	Description	Class	Type
1	bq	biquad cascade	reference to variable	HAL variable type biquad
2	index	index at the cascade	basic io type	basic type integer
3	fs	sample frequency	reference to the data of a variable	HAL variable type rational
4	fc	cut off frequency	reference to the data of a variable	HAL variable type rational

1.2.45 initBiquadAsPeakFilter (HAL fid =102)

Informations:

HAL function id:	102
description:	inits a biquad filter as peak filter
AL ASM syntax:	initBiquadAsPeakFilter bq,index,fs,fc,q,g;

Parameters:

Nr.	Name	Description	Class	Type
1	bq	biquad cascade	reference to variable	HAL variable type biquad
2	index	index at the cascade	basic io type	basic type integer
3	fs	sample frequency	reference to the data of a variable	HAL variable type rational
4	fc	center frequency	reference to the data of a variable	HAL variable type rational
5	q	quality	reference to the data of a variable	HAL variable type rational
6	g	gain (not in dB)	reference to the data of a variable	HAL variable type rational

1.2.46 initBiquadAsLowFreqShelvFilter (HAL fid =103)

Informations:

HAL function id:	103
description:	inits a biquad filter as low frequency shelving filter
AL ASM syntax:	initBiquadAsLowFreqShelvFilter bq,index,fs,f,q,g;

Parameters:

Nr.	Name	Description	Class	Type
1	bq	biquad cascade	reference to variable	HAL variable type biquad
2	index	index at the cascade	basic io type	basic type integer
3	fs	sample frequency	reference to the data of a variable	HAL variable type rational
4	f	cut/boost frequency	reference to the data of a variable	HAL variable type rational
5	q	quality	reference to the data of a variable	HAL variable type rational
6	g	gain (not in dB)	reference to the data of a variable	HAL variable type rational

1.2.47 initBiquadAsHighFreqShelvFilter (HAL fid =104)

Informations:

HAL function id:	104
description:	inits a biquad filter as high frequency shelving filter
AL ASM syntax:	initBiquadAsHighFreqShelvFilter bq,index,fs,f,q,g;

Parameters:

Nr.	Name	Description	Class	Type
1	bq	biquad cascade	reference to variable	HAL variable type biquad
2	index	index at the cascade	basic io type	basic type integer
3	fs	sample frequency	reference to the data of a variable	HAL variable type rational
4	f	cut/boost frequency	reference to the data of a variable	HAL variable type rational
5	q	quality	reference to the data of a variable	HAL variable type rational
6	g	gain (not in dB)	reference to the data of a variable	HAL variable type rational

1.2.48 convoluteBiquad (HAL fid =110)

Informations:

HAL function id:	110
description:	convolute biquad with an input and generate an output
AL ASM syntax:	convoluteBiquad x,bqa,y;

Parameters:

Nr.	Name	Description	Class	Type
1	x	input	reference to the data of a variable	HAL variable type rational
2	bqa	biquad cascade	reference to the data of a variable	HAL variable type biquad
3	y	output	reference to the data of a variable	HAL variable type rational

1.2.49 initNoisegate (HAL fid =111)

Informations:

HAL function id:	111
description:	initialize a noisegate
AL ASM syntax:	initNoisegate ng,rmsTAV,AT,RT,NT,NS;

Parameters:

Nr.	Name	Description	Class	Type
1	ng	noisegate	reference to the data of a variable	HAL variable type noisegate
2	rmsTAV	time average value for the rms	basic io type	basic type
3	AT	attack value for the smoothing	basic io type	rational basic type
4	RT	release value for the smoothing	basic io type	rational basic type
5	NT	noise cut off threshold	basic io type	rational basic type
6	NS	slope	basic io type	rational basic type

1.2.50 initExpander (HAL fid =112)

Informations:

HAL function id:	112
description:	initialize a expander
AL ASM syntax:	initExpander exp,rmsTAV,AT,RT,ET,ES;

Parameters:

Nr.	Name	Description	Class	Type
1	exp	expander	reference to the data of a variable	HAL variable type expander
2	rmsTAV	time average value for the rms	basic io type	basic type
3	AT	attack value for the smoothing	basic io type	rational basic type
4	RT	release value for the smoothing	basic io type	rational basic type
5	ET	expander threshold	basic io type	rational basic type
6	ES	slope	basic io type	rational basic type

1.2.51 initCompressor (HAL fid =113)

Informations:

HAL function id:	113
description:	initialize a compressor
AL ASM syntax:	initCompressor comp,rmsTAV,AT,RT,CT,CS;

Parameters:

Nr.	Name	Description	Class	Type
1	comp	compressor	reference to the data of a variable	HAL variable type compressor
2	rmsTAV	time average value for the rms	basic io type	basic type
3	AT	attack value for the smoothing	basic io type	rational basic type
4	RT	release value for the smoothing	basic io type	rational basic type
5	CT	compressor threshold	basic io type	rational basic type
6	CS	slope	basic io type	rational basic type

1.2.52 initLimiter (HAL fid =114)

Informations:

HAL function id:	114
description:	initialize a limiter
AL ASM syntax:	initLimiter lim,ATpeak,RTpeak,ATsmooth,RTsmooth,LT,LS;

Parameters:

Nr.	Name	Description	Class	Type
1	lim	limiter	reference to the data of a variable	HAL variable type limiter
2	ATpeak	attack value for the peak detection	basic io type	basic type rational
3	RTpeak	release value for the peak detection	basic io type	basic type rational
4	ATsmooth	attack value for the smoothing	basic io type	basic type rational
5	RTsmooth	release value for the smoothing	basic io type	basic type rational
6	LT	limiter threshold	basic io type	basic type rational
7	LS	slope	basic io type	basic type rational

1.2.53 calcNoisegate (HAL fid =115)

Informations:

HAL function id:	115
description:	sends a stream of samples through a noisegate
AL ASM syntax:	calcNoisegate x,ng,y;

Parameters:

Nr.	Name	Description	Class	Type
1	x	input	reference to the data of a variable	HAL variable type rational
2	ng	noisegate	reference to the data of a variable	HAL variable type noisegate
3	y	output	reference to the data of a variable	HAL variable type rational

1.2.54 calcExpander (HAL fid =116)

Informations:

HAL function id:	116
description:	sends a stream of samples through a expander
AL ASM syntax:	calcExpander x,exp,y;

Parameters:

Nr.	Name	Description	Class	Type
1	x	input	reference to the data of a variable	HAL variable type rational
2	exp	expander	reference to the data of a variable	HAL variable type expander
3	y	output	reference to the data of a variable	HAL variable type rational

1.2.55 calcCompressor (HAL fid =117)

Informations:

HAL function id:	117
description:	sends a stream of samples through a compressor
AL ASM syntax:	calcCompressor x,comp,y;

Parameters:

Nr.	Name	Description	Class	Type
1	x	input	reference to the data of a variable	HAL variable type rational
2	comp	compressor	reference to the data of a variable	HAL variable type compressor
3	y	output	reference to the data of a variable	HAL variable type rational

1.2.56 calcLimiter (HAL fid =118)

Informations:

HAL function id:	118
description:	sends a stream of samples through a limiter
AL ASM syntax:	calcLimiter x,lim,y;

Parameters:

Nr.	Name	Description	Class	Type
1	x	input	reference to the data of a variable	HAL variable type rational
2	lim	limiter	reference to the data of a variable	HAL variable type limiter
3	y	output	reference to the data of a variable	HAL variable type rational

1.2.57 initHannWindow (HAL fid =120)

Informations:

HAL function id:	120
description:	inits an array as an "von Hann" window
AL ASM syntax:	initHannWindow wl,flag,wnd;

Parameters:

Nr.	Name	Description	Class	Type
1	wl	window length	basic io type	basic type integer
2	flag	set to 1 if used for periodic constructs like DFT/FFT	basic io type	basic type integer
3	wnd	window coefficients (array of rational values)	reference to variable	HAL variable type rational

1.2.58 initRFFT (HAL fid =130)

Informations:

HAL function id:	130
description:	init FFT structure as real input FFT
AL ASM syntax:	initRFFT N,x,y,fftStruct;

Parameters:

Nr.	Name	Description	Class	Type
1	N	FFT length	basic io type	basic type integer
2	x	input	reference to the data of a variable	HAL variable type rational
3	y	output	reference to the data of a variable	HAL variable type complex
4	fftStruct	FFT structure	reference to variable	HAL variable type FFT

1.2.59 initIFFT (HAL fid =131)

Informations:

HAL function id:	131
description:	init inverse FFT structure
AL ASM syntax:	initIFFT N,x,y,ifftStruct;

Parameters:

Nr.	Name	Description	Class	Type
1	N	IFFT length	basic io type	basic type integer
2	x	input	reference to the data of a variable	HAL variable type complex
3	y	output	reference to the data of a variable	HAL variable type rational
4	ifftStruct	IFFT structure	reference to variable	HAL variable type FFT

1.2.60 processRFFT (HAL fid =132)

Informations:

HAL function id:	132
description:	processes the real input FFT
AL ASM syntax:	processRFFT fftStruct;

Parameters:

Nr.	Name	Description	Class	Type
1	fftStruct	the fft structure	reference to variable	HAL variable type FFT

1.2.61 processIFFT (HAL fid =133)

Informations:

HAL function id:	133
description:	processes the IFFT
AL ASM syntax:	processIFFT ifftStruct;

Parameters:

Nr.	Name	Description	Class	Type
1	ifftStruct	the ifft info structure	reference to variable	HAL variable type FFT

1.2.62 calcDelay (HAL fid =150)

Informations:

HAL function id:	150
description:	stream a vector of samples in and a vector of exactly the same size out
AL ASM syntax:	calcDelay x,delay,y;

Parameters:

Nr.	Name	Description	Class	Type
1	x	input	reference to the data of a variable	HAL variable type rational
2	delay	the delay	reference to the data of a variable	HAL variable type delay
3	y	output	reference to the data of a variable	HAL variable type rational

1.2.63 initDelay (HAL fid =151)

Informations:

HAL function id:	151
description:	inits the delay
AL ASM syntax:	initDelay d,N,Nindex,readToWriteOffset;

Parameters:

Nr.	Name	Description	Class	Type
1	d	the delay	reference to the data of a variable	HAL variable type delay
2	N	number of samples which correspond to the delay time ($T_{delay} = N * T_a$)	reference to the data of a variable	HAL variable type integer
3	Nindex	the index at the vector of N	basic io type	basic type integer
4	readToWriteOffset	the distance in Samples between the read and write position	basic io type	basic type integer

1.2.64 uiSetDim (HAL fid =200)

Informations:

HAL function id:	200
description:	sets the dimension of a UI
AL ASM syntax:	uiSetDim x,y,xle,y,ui,uiIndex;

Parameters:

Nr.	Name	Description	Class	Type
1	x	x position	basic io type	basic type integer
2	y	y position	basic io type	basic type integer
3	xle	x length	basic io type	basic type integer
4	yle	y length	basic io type	basic type integer
5	ui	ui	reference to the data of a variable	unknown(error)
6	uiIndex	ui index	basic io type	basic type integer

1.2.65 uiInitPanel (HAL fid =201)

Informations:

HAL function id:	201
description:	inits a panel
AL ASM syntax:	uiInitPanel uiUUID,p,pIndex;

Parameters:

Nr.	Name	Description	Class	Type
1	uiUUID	uuid of the pannel	basic io type	basic type integer
2	p	panel	reference to the data of a variable	HAL variable type panel
3	pIndex	panel index	basic io type	basic type integer

1.2.66 uiInitButton (HAL fid =202)

Informations:

HAL function id:	202
description:	inits a button
AL ASM syntax:	uiInitButton uiUUID,b,bIndex,p,pIndex;

Parameters:

Nr.	Name	Description	Class	Type
1	uiUUID	uuid of the button	basic io type	basic type integer
2	b	button	reference to the data of a variable	HAL variable type button
3	bIndex	button index	basic io type	basic type integer
4	p	panel	reference to the data of a variable	HAL variable type panel
5	pIndex	panel index	basic io type	basic type integer

1.2.67 uiInitDisplay (HAL fid =203)

Informations:

HAL function id:	203
description:	inits a display
AL ASM syntax:	uiInitDisplay uiUUID,d,dIndex,p,pIndex;

Parameters:

Nr.	Name	Description	Class	Type
1	uiUUID	uuid of the display	basic io type	basic type integer
2	d	display	reference to the data of a variable	HAL variable type display
3	dIndex	display index	basic io type	basic type integer
4	p	panel	reference to the data of a variable	HAL variable type panel
5	pIndex	panel index	basic io type	basic type integer

1.2.68 uiInitLED (HAL fid =204)

Informations:

HAL function id:	204
description:	inits a LED
AL ASM syntax:	uiInitLED uiUUID,l,lIndex,p,pIndex;

Parameters:

Nr.	Name	Description	Class	Type
1	uiUUID	uuid of the LED	basic io type	basic type integer
2	l	LED	reference to the data of a variable	HAL variable type led
3	lIndex	LED index	basic io type	basic type integer
4	p	panel	reference to the data of a variable	HAL variable type panel
5	pIndex	panel index	basic io type	basic type integer

1.2.69 uiCheckButtonPressed (HAL fid =210)

Informations:

HAL function id:	210
description:	if the button was pressed the CF is set
AL ASM syntax:	uiCheckButtonPressed b,bIndex;

Parameters:

Nr.	Name	Description	Class	Type
1	b	button	reference to the data of a variable	HAL variable type button
2	bIndex	button index	basic io type	basic type integer

1.2.70 uiSetLED (HAL fid =211)

Informations:

HAL function id:	211
description:	set the LED state
AL ASM syntax:	uiSetLED l,lIndex,onFlag;

Parameters:

Nr.	Name	Description	Class	Type
1	l	LED	reference to the data of a variable	HAL variable type led
2	lIndex	LED index	basic io type	basic type integer
3	onFlag	if the flag is not zero the LED is turned on	basic io type	basic type integer

1.2.71 uiSetDisplay (HAL fid =212)

Informations:

HAL function id:	212
description:	set the text of a display
AL ASM syntax:	uiSetDisplay d,dIndex,s,iString;

Parameters:

Nr.	Name	Description	Class	Type
1	d	display	reference to the data of a variable	HAL variable type display
2	dIndex	display index	basic io type	basic type integer
3	s	the string	reference to the data of a variable	HAL variable type string
4	iString	index of the string at the array	basic io type	basic type integer

Chapter 2

Communication Interface

2.1 Message Frames

2.1.1 ACK (msgId =1)

Informations:

message type id:	1
description:	acknowledge

2.1.2 NACK (msgId =2)

Informations:

message type id:	2
description:	not acknowledge

2.1.3 startPrg (msgId =10)

Informations:

message type id:	10
description:	start sending a program

Elements:

Pos.	Name	Description	Type
0	globalVariableNumber	number of the global variables	integer
1	localVariableNumber	number of the local variables	integer
2	instructionNumber	number of the instructions	integer

2.1.4 sendVariable (msgId =11)

Informations:

message type id:	11
description:	sends a variable

Elements:

Pos.	Name	Description	Type
0	index	the index of the variable	integer
1	varTypeID	type id of the variable	integer
2	num	number of variables	integer

2.1.5 sendInstruction (msgId =12)

Informations:

message type id:	12
description:	sends an instruction

Elements:

Pos.	Name	Description	Type
0	index	index of the function call	integer
1	fbcode	function byte code	raw (array)

2.1.6 endPrg (msgId =13)

Informations:

message type id:	13
description:	sings that the program transmission has completed

2.1.7 stop (msgId =20)

Informations:

message type id:	20
description:	stops the AP

2.1.8 step (msgId =21)

Informations:

message type id:	21
description:	the AP executes one instruction

2.1.9 run (msgId =22)

Informations:

message type id:	22
description:	the AP runs the program

2.1.10 updateVariable (msgId =23)

Informations:

message type id:	23
description:	a variable going to be updated

Elements:

Pos.	Name	Description	Type
0	gIndex	global variable index	integer
1	dataElements	amount of data elements	integer

2.1.11 login (msgId =30)

Informations:

message type id:	30
description:	a AP is going to be logged in to the system

2.1.12 logout (msgId =31)

Informations:

message type id:	31
description:	a AP is going to be logged out of the system

2.2 Message Processes

2.2.1 logout (msgProcId =0)

Informations:

message process id:	0
description:	log the AP out of the system
handler:	TX

code:

```
1 TX_logout [handle=none]() {  
2     declare recv    Node;  
3     declare mNum    mNum;  
4 }
```

```

5         recv = ALL;
6         loopAll drivers[driver] {
7             mNum = getNewMsgNum();
8             driver>>send(logout(recv,mNum));
9         }
10    return 0;
11 }

```

2.2.2 updateVariable (msgProcId =0)

Informations:

message process id:	0
description:	updates a global variable at different systems
handler:	TX

code:

```

1 TX_updateVariable [handle=none](VarIndex vi) {
2     declare mNum      mNum;
3     declare apV        Variable;
4
5     apV = getVariableByIndex(vi);
6
7     loopAll nodes[node] {
8         if (getNodeIDfromNode(node) !=
9             getSelfSenderID()) {
10             mNum = getNewMsgNum();
11             apV>>call(sendUpdate(
12                 getVariableData(apV),
13                 getDriverFromNode(node),ALL,
14                 mNum,vi));
15             if (waitACK(mNum)) {
16                 return -1;
17             }
18         }
19     }
20     return 0;
21 }

```

2.2.3 login (msgProcId =10)

Informations:

message process id:	10
description:	log in the AP to the system
handler:	TX

code:

```

1 TX_login [handle=none]() {
2     declare recv    Node;
3     declare mNum    mNum;
4
5     recv = ALL;
6     loopAll drivers[driver] {
7         mNum = getNewMsgNum();
8         driver>>send(login(recv,mNum));
9     }
10    return 0;
11 }

```

2.2.4 run (msgProcId =20)

Informations:

message process id:	20
description:	runs the audio-processor
handler:	TX

code:

```

1 TX_run [handle=none](Node dest) {
2     declare driver  Driver;
3     declare sender  Node;
4     declare mNum    mNum;
5
6     sender = getSelfSenderID();
7     if (dest != ALL) {
8         mNum = getNewMsgNum();
9         driver = getDriver(dest);
10        driver>>send(run(sender,mNum));
11        return waitACK(mNum);
12    }
13
14    loopAll drivers[drv] {
15        mNum = getNewMsgNum();
16        drv>>send(run(sender,mNum));
17        if (waitACK(mNum)) {
18            return -1;
19        }
20    }
21    return 0;
22 }

```

2.2.5 run (msgProcId =22)

Informations:

message process id:	22
description:	handles the run command
handler:	RX

code:

```
1 RX_run [handle=run]() {
2     declare sender                Node;
3     declare driver                Driver;
4     declare mNum                  mNum;
5
6     sender = getSender();
7     driver = getDriver();
8     mNum = getMsgNum();
9
10    if (!runAP()) {
11        return driver>>send(ACK(sender,mNum));
12    } else {
13        return driver>>send(NACK(sender,mNum));
14    }
15 }
```

2.2.6 updateVariable (msgProcId =23)

Informations:

message process id:	23
description:	updates a global variable at different systems
handler:	RX

code:

```
1 RX_updateVariable [handle=updateVariable]() {
2     declare sender                Node;
3     declare driver                Driver;
4     declare mNum                  mNum;
5     declare apV                    Variable
6     ;
7     declare gIndex                int;
8
9     sender = getSender();
10    driver = getDriver();
11    mNum = getMsgNum();
12    gIndex = getMsgDataToInt(0);
13
14    apV = getVariableByIndex(gIndex);
15    if (!apV) {
```

```

15         return -1;
16     }
17
18     apV>>call(recvUpdate(getVariableData(apV),
19                     getMsgDataRef()));
20
21     return driver>>send(ACK(sender,mNum));
22 }

```

2.2.7 login (msgProcId =30)

Informations:

message process id:	30
description:	handle login calls
handler:	RX

code:

```

1 RX_login [handle=login]() {
2     declare sender  Node;
3     declare driver  Driver;
4     declare mNum    mNum;
5
6     sender = getSender();
7     driver = getDriver();
8     mNum = getMsgNum();
9
10    if (!addNode(sender, driver)) {
11        return driver>>send(login(sender,mNum));
12    }
13    return 0;
14 }

```

2.2.8 logout (msgProcId =31)

Informations:

message process id:	31
description:	handle login calls
handler:	RX

code:

```

1 RX_logout [handle=logout]() {
2     declare sender  Node;
3
4     sender = getSender();
5

```



```
6     removeNode(sender);  
7     return 0;  
8 }
```

Chapter 3

Implementations

3.1 Implementation groups

3.1.1 ADSP 21369 blockbased, ADC in, DAC3 & DAC4 out, UART @9600,n,8,1

includes:

c-Include	c-Library	system lib
sru21369.h		yes
cdef21369.h		yes
def21369.h		yes
signal.h		yes
stdio.h		yes

code:

```
1 // The following definition allows the SRU macro to
  // check for errors. Once the routings have
2 // been verified, this definition can be removed to save
  // some program memory space.
3 // The preprocessor will issue a warning stating this
  // when using the SRU macro without this
4 // definition
5 #define SRUDEBUG // Check SRU Routings for errors.
6 #include <SRU.h>
7
8
9 #define NUM_SAMPLES (1024)
10 #define DAC4
11
12 //
  =====
```

```

13 // AD 1835 defines
14 //
=====
15
16 //
17 // AD1835.h
18 //
19 // Configuration values for the AD1835A codec
20 //
21
22 #define DACCTRL1 (0x0000) // DAC control
    register 1 (R/W)
23 #define DACCTRL2 (0x1000) // DAC control
    register 2 (R/W)
24 #define DACVOL_L1 (0x2000) // DAC volume - left 1
    (R/W)
25 #define DACVOL_R1 (0x3000) // DAC volume - right 1
    (R/W)
26 #define DACVOL_L2 (0x4000) // DAC volume - left 2
    (R/W)
27 #define DACVOL_R2 (0x5000) // DAC volume - right 2
    (R/W)
28 #define DACVOL_L3 (0x6000) // DAC volume - left 3
    (R/W)
29 #define DACVOL_R3 (0x7000) // DAC volume - right 3
    (R/W)
30 #define DACVOL_L4 (0x8000) // DAC volume - left 4
    (R/W)
31 #define DACVOL_R4 (0x9000) // DAC volume - right 4
    (R/W)
32 #define ADCPEAKL (0xA000) // ADC left peak
    (R)
33 #define ADCPEAKR (0xB000) // ADC right peak
    (R)
34 #define ADCCTRL1 (0xC000) // ADC control 1
    (R/W)
35 #define ADCCTRL2 (0xD000) // ADC control 2
    (R/W)
36 #define ADCCTRL3 (0xE000) // ADC control 3
    (R/W)
37
38 #define RD (0x0800)
39 #define WR (0x0000) // Write to register
40
41
42 // DAC control register 1
43 #define DEEMPH44_1 (0x0100) // Deemphasis filter
    for 44.1 KHz
44 #define DEEMPH32 (0x0200) // Deemphasis filter
    for 32.0 KHz
45 #define DEEMPH48 (0x0300) // Deemphasis filter
    for 48.0 KHz
46
47 #define DACI2S (0x0000) // DAC receives I2S

```

```

format
48 #define DACRJ          (0x0020)  // DAC receives I2S
format
49 #define DACDSP         (0x0040)  // DAC receives I2S
format
50 #define DACLJ          (0x0060)  // DAC receives I2S
format
51 #define DACPACK256     (0x0080)  // DAC receives I2S
format
52
53 #define DAC24BIT        (0x0000)  // 24-bit output word
length
54 #define DAC20BIT        (0x0008)  // 20-bit output word
length
55 #define DAC16BIT        (0x0010)  // 16-bit output word
length
56
57 #define DACPOWERDN      (0x0004)  // DAC into power-down
mode
58
59 #define DACFS48          (0x0000)  // Sample rate = 48
kHz (x8)
60 #define DACFS96          (0x0001)  // Sample rate = 96
kHz (x4)
61 #define DACFS192         (0x0002)  // Sample rate = 192
kHz (x2)
62
63
64 // DAC control register 2
65
66 #define DACREPLICATE     (0x0100)  // Replicate output of
DAC 1/2 on 3/4, 5/6 & 7/8
67 #define DACMUTE_R4       (0x0080)  // Mute DAC output
channel (clear to un-mute)
68 #define DACMUTE_L4       (0x0040)  // Mute DAC output
channel (clear to un-mute)
69 #define DACMUTE_R3       (0x0020)  // Mute DAC output
channel (clear to un-mute)
70 #define DACMUTE_L3       (0x0010)  // Mute DAC output
channel (clear to un-mute)
71 #define DACMUTE_R2       (0x0008)  // Mute DAC output
channel (clear to un-mute)
72 #define DACMUTE_L2       (0x0004)  // Mute DAC output
channel (clear to un-mute)
73 #define DACMUTE_R1       (0x0002)  // Mute DAC output
channel (clear to un-mute)
74 #define DACMUTE_L1       (0x0001)  // Mute DAC output
channel (clear to un-mute)
75
76
77 //
-----
78 //DAC Volume Control - 10-bit granularity (1024 levels)
79 #define DACVOL_MIN       (0x000)

```

```

80 #define DACVOL_LOW      (0X100)
81 #define DACVOL_MED      (0X200)
82 #define DACVOL_HI       (0X300)
83 #define DACVOL_MAX      (0x3FF)
84 #define DACVOL_MASK     (0x3FF) // Volume in dB is in
    10 LSBs
85
    // 3FF = 0 dBFS =
    1023/1023
86    // 3FE = -0.01 dBFS =
    1022/1023
87    // ...
88    // 002 = -50.7 dBFS =
    3/1023
89    // 001 = -54.2 dBFS =
    2/1023
90
91 //
-----

92 // ADC Control 1
93
94 #define ADCHPF          (0x0100) // High pass filter (AC-
    coupled)
95 #define ADCPOWERDN     (0x0080) // DAC into power-down mode
96 #define ADCFS48        (0x0000) // Sample rate = 48 KHz
97 #define ADCFS96        (0x0040) // Sample rate = 96 KHz
98
99 //
-----

100 // ADC Control 2
101
102 #define AUXSLAVE        (0x0000) // Aux input is in slave
    mode
103 #define AUXMASTER      (0x0200) // Aux input is in master
    mode
104
105 #define ADCI2S          (0x0000) // ADC transmits in I2S
    format
106 #define ADCRJ           (0x0040) // ADC transmits in right-
    justified format
107 #define ADCDSP          (0x0080) // ADC transmits in DSP (
    TDM) format
108 #define ADCLJ           (0x00C0) // ADC transmits in left-
    justified format
109 #define ADCPACK256      (0x0100) // ADC transmits in packed
    256 format
110 #define ADCAUX256       (0x0180) // ADC transmits in packed
    128 format
111
112 #define ADC24BIT         (0x0000) // 24-bit output word
    length
113 #define ADC20BIT        (0x0010) // 20-bit output word
    length
114 #define ADC16BIT        (0x0020) // 16-bit output word

```

```

length
115
116 #define ADCMUTER      (0x0002) // Mute right channel from
    ADC
117 #define ADCMUTEL      (0x0001) // Mute right channel from
    ADC
118
119 //
    -----

120 //  ADC Control 3
121
122 #define IMCLKx2        (0x0000) // Internal MCLK = external
    MCLK x 2
123 #define IMCLKx1        (0x0040) // Internal MCLK = external
    MCLK
124 #define IMCLKx23       (0x0080) // Internal MCLK = external
    MCLK x 2/3
125
126 #define PEAKRDEN       (0x0020) // Enable reads of peak ADC
    levels
127 #define PEAKLEVELMASK (0x003F) // Six significant bit
    of level
128

129
130
131 //
    =====

132 // talk through interface
133 //
    =====

134
135 // Function prototypes for this talkthrough code
136
137 extern void InitPLL_SDRAM(void);
138 extern void processBlock(unsigned int *);
139
140 extern void InitSRU(void);
141 extern void Init1835viaSPI(void);

```

```

142
143 extern void InitSPORT(void);
144 extern void TalkThroughISR(int);
145 extern void ClearSPORT(void);
146
147 extern void SetupSPI1835 (void) ;
148 extern void DisableSPI1835 (void) ;
149 extern void Configure1835Register (int i) ;
150 extern unsigned int Get1835Register (int i) ;
151
152 extern void SetupIRQ01 (void) ;
153 extern void Irq0ISR (int i) ;
154 extern void Irq1ISR (int i) ;
155
156 typedef void (* TFkt_ADSPuartCB) (unsigned int value);
157
158 extern void initUART(TFkt_ADSPuartCB cbRXFunction);
159 extern void UARTISR(int i);
160 extern void sendUARTuint32Values(unsigned int *pD, int
    amount);
161
162 extern void Delay (int i) ;
163
164 //
    =====
165 // init AD1835
166 //
    =====

167 /* Setup the SPI pramaters here in a buffer first */
168 unsigned int Config1835Param [] = {
169     WR | DACCTRL1 | DACI2S | DAC24BIT | DACFS48,
170     WR | DACCTRL2 ,//| DACMUTE_R4 | DACMUTE_L4,
171     WR | DACVOL_L1 | DACVOL_MAX,
172     WR | DACVOL_R1 | DACVOL_MAX,
173     WR | DACVOL_L2 | DACVOL_MAX,
174     WR | DACVOL_R2 | DACVOL_MAX,
175     WR | DACVOL_L3 | DACVOL_MAX,
176     WR | DACVOL_R3 | DACVOL_MAX,
177     WR | DACVOL_L4 | DACVOL_MAX,
178     WR | DACVOL_R4 | DACVOL_MAX,
179     WR | ADCCTRL1 | ADCFS48,
180     WR | ADCCTRL2 | ADCI2S | ADC24BIT,
181     WR | ADCCTRL3 | IMCLKx2
182     } ;
183
184 volatile int spiFlag ;
185
186 //Set up the SPI port to access the AD1835
187 void SetupSPI1835 ()
188 {
189     /* First configure the SPI Control registers */
190     /* First clear a few registers */
191     *pSPICTL = (TXFLSH | RXFLSH) ;

```

```

192     *pSPIFLG = 0;
193     *pSPICTL = 0;
194
195     /* Setup the baud rate to 500 KHz */
196     *pSPIBAUD = 100;
197
198     /* Setup the SPI Flag register to FLAG3 : 0xF708*/
199     *pSPIFLG = 0xF708;
200
201     /* Now setup the SPI Control register : 0x5281*/
202     *pSPICTL = (SPIEN | SPIMS | MSBF | WL16 | TIMOD1) ;
203
204 }
205
206 //Disable the SPI Port
207 void DisableSPI1835 ()
208 {
209     *pSPICTL = (TXFLSH | RXFLSH);
210 }
211
212 //Send a word to the AD1835 via SPI
213 void Configure1835Register (int val)
214 {
215     *pTXSPI = val ;
216     Delay(100);
217
218     //Wait for the SPI to indicate that it has finished.
219     while (1)
220     {
221         if (*pSPISTAT & SPIF)
222             break ;
223     }
224     Delay (100) ;
225 }
226
227 //Receive a register setting from the AD1835
228 unsigned int Get1835Register (int val)
229 {
230     *pTXSPI = val ;
231     Delay(100);
232
233     //Wait for the SPI port to indicate that it has
234     //finished
235     while (1)
236     {
237         if (SPIF & *pSPISTAT)
238             break ;
239     }
240     Delay (100) ;
241     return *pRXSPI ;
242 }
243
244 //Set up all AD1835 registers via SPI
245 void Init1835viaSPI()

```



```

246 {
247     int configSize = sizeof (Config1835Param) / sizeof (
248         int) ;
249     int i ;
250     SetupSPI1835 () ;
251
252     for (i = 0; i < configSize; ++i)
253     {
254         Configure1835Register (Config1835Param[i]) ;
255     }
256
257     DisableSPI1835 () ;
258 }
259
260 //Delay loop
261 void Delay (int i)
262 {
263     for (;i>0;--i)
264         asm ("nop;") ;
265 }
266
267 //
268 //=====
269 // PLL for SDRAM init
270 //
271 //=====
272
273 void InitPLL_SDRAM(){
274     int i, pmctlsetting;
275
276     //Change this value to optimize the performance for
277     //quazi-sequential accesses (step > 1)
278     #define SDMODIFY 1
279
280     pmctlsetting= *pPMCTL;
281     pmctlsetting &= ~(0xFF); //Clear
282
283     // CLKIN= 24.576 MHz, Multiplier= 27, Divisor= 1,
284     // INDIV=1, CCLK_SDCLK_RATIO= 2.
285     // Core clock = (24.576 MHz * 27) /2 = 331.776 MHz
286     pmctlsetting= SDCKR2|PLLM27|INDIV|DIVEN;
287     *pPMCTL= pmctlsetting;
288     pmctlsetting|= PLLBP;
289     *pPMCTL= pmctlsetting;
290
291     //Wait for around 4096 cycles for the pll to lock.
292     for (i=0; i<4096; i++)
293         asm("nop;");
294
295     *pPMCTL ^= PLLBP; //Clear Bypass Mode
296     *pPMCTL |= (CLKOUTEN); //and start clkout

```

```

294
295
296 // Programming SDRAM control registers and enabling
    SDRAM read optimization
297 // CCLK_SDCLK_RATIO= 2.5
298 // RDIV = ((f SDCLK X t REF )/NRA) - (tRAS + tRP )
299 // (166*(10^6)*64*(10^-3)/4096) - (7+3) = 2583
300
301 *pSDRRC= (0xA17)|(SDMODIFY<<17)|SDROPT;
302
303 //
    =====
304
305 // Configure SDRAM Control Register (SDCTL) for PART
    MT48LC4M32B2
306
307 // SDCL3 : SDRAM CAS Latency= 3 cycles
308 // DSDCLK1: Disable SDRAM Clock 1
309 // SDPSS : Start SDRAM Power up Sequence
310 // SDCAW8 : SDRAM Bank Column Address Width= 8 bits
311 // SDRAW12: SDRAM Row Address Width= 12 bits
312 // SDTRAS7: SDRAM tRAS Specification. Active
    Command delay = 7 cycles
313 // SDTRP3 : SDRAM tRP Specification. Precharge
    delay = 3 cycles.
314 // SDTWR2 : SDRAM tWR Specification. tWR = 2 cycles
    .
315 // SDTRCD3: SDRAM tRCD Specification. tRCD = 3
    cycles.
316 //
317 //
    -----
318
319 *pSDCTL= SDCL3|DSDCLK1|SDPSS|SDCAW8|SDRAW12|SDTRAS7|
    SDTRP3|SDTWR2|SDTRCD3;
320
321 // Note that MS2 & MS3 pin multiplexed with flag2 &
    flag3.
322 // MSEN bit must be enabled to access SDRAM, but
    LED7 cannot be driven with sdram
323 *pSYSCTL |=MSEN;
324
325 // Mapping Bank 2 to SDRAM
326 // Make sure that jumper is set appropriately so
    that MS2 is connected to
327 // chip select of 16-bit SDRAM device
328 *pEPCTL |=B2SD;
329 *pEPCTL &= ~(B0SD|B1SD|B3SD);
330
331 //
    =====
332
    //

```

```

333 // Configure AMI Control Register (AMICTL0) Bank 0
    for the ISSI IS61LV5128
334 //
335 // WS2 : Wait States = 2 cycles
336 // HC1 : Bus Hold Cycle (at end of write access)=
    1 cycle.
337 // AMIEN: Enable AMI
338 // BW8 : External Data Bus Width= 8 bits.
339 //
340 //
    -----

341
342 //SRAM Settings
343 *pAMICTL0 = WS2|HC1|AMIEN|BW8;
344
345 //
    =====

346 //
347 // Configure AMI Control Register (AMICTL) Bank 1
    for the AMD AM29LV08
348 //
349 // WS23 : Wait States= 23 cycles
350 // AMIEN: Enable AMI
351 // BW8 : External Data Bus Width= 8 bits.
352 //
353 //
    -----

354
355 //Flash Settings
356 *pAMICTL1 = WS23|AMIEN|BW8;
357 }
358
359 //
    =====

360 // serial Port
361 //
    =====

362 /*
363 Here is the mapping between the SPORTS and the DACS
364 ADC -> DSP : SPORT0A : I2S
365 DSP -> DAC1 : SPORT1A : I2S
366 DSP -> DAC2 : SPORT1B : I2S
367 DSP -> DAC3 : SPORT2A : I2S
368 DSP -> DAC4 : SPORT2B : I2S
369 */
370
371 unsigned int PCI = 0x00080000 ;
372 unsigned int OFFSET = 0x00080000 ;
373
374 // TCB blocks for Chaining

```

```

375 //Each block will be used for:
376 //      Filling from the ADC
377 //      Processing filled data
378 //      Sending to DAC
379 //
380 //Each one is doing only one of these steps for each
      SPORT interrupt.
381
382 //For this example the startup state is
383 // Start to 1st interrupt: gBlock_A is filled first,
      gBlock_C is sent
384 // 1st int to 2nd int: gBlock_C filled, gBlock_A
      processed, gBlock_B sent
385 // 2nd int to 3rd int: gBlock_B filled, gBlock_C
      processed, gBlock_A sent
386 // 3rd int to 4th int: gBlock_A filled, gBlock_B
      processed, gBlock_C sent
387 unsigned int gBlock_A[NUM_SAMPLES] ;
388 unsigned int gBlock_B[NUM_SAMPLES] ;
389 unsigned int gBlock_C[NUM_SAMPLES] ;
390
391 //Set up the TCBs to rotate automatically
392 int TCB_gBlock_A[4] = { 0, sizeof(gBlock_A), 1, 0};;
393 int TCB_gBlock_B[4] = { 0, sizeof(gBlock_B), 1, 0};
394 int TCB_gBlock_C[4] = { 0, sizeof(gBlock_C), 1, 0};
395
396 void InitSPORT()
397 {
398     //Proceed from Block A to Block C
399     TCB_gBlock_A[0] = (int) TCB_gBlock_C + 3 - OFFSET +
        PCI ;
400     TCB_gBlock_A[3] = (unsigned int) gBlock_A - OFFSET ;
401
402     //Proceed from Block B to Block A
403     TCB_gBlock_B[0] = (int) TCB_gBlock_A + 3 - OFFSET +
        PCI ;
404     TCB_gBlock_B[3] = (unsigned int) gBlock_B - OFFSET ;
405
406     //Proceed from Block C to Block B
407     TCB_gBlock_C[0] = (int) TCB_gBlock_B + 3 - OFFSET +
        PCI ;
408     TCB_gBlock_C[3] = (unsigned int) gBlock_C - OFFSET ;
409
410     //Clear the Mutlichannel control registers
411     *pSPMCTL0 = 0;
412     *pSPMCTL1 = 0;
413     *pSPMCTL2 = 0;
414     *pSPCTL0 = 0 ;
415     *pSPCTL1 = 0 ;
416     *pSPCTL2 = 0 ;
417
418     //
      =====
419     //

```

```

420 // Configure SPORT 0 for input from ADC
421 //
422 //
-----

423
424
425 *pSPCTL0 = (OPMODE | SLEN24 | SPEN_A | SCHEN_A |
SDEN_A);
426
427 // Enabling Chaining
428 // Block A will be filled first
429 *pCPSP0A = (unsigned int) TCB_gBlock_A - OFFSET + 3
;
430
431 //
=====

432 //
433 // Configure SPORTs 1 & 2 for output to DACs 1-4
434 //
435 //
-----

436
437 #ifdef DAC1
438 *pSPCTL1 = (SPTRAN | OPMODE | SLEN24 | SPEN_A |
SCHEN_A | SDEN_A) ;
439 // write to DAC1
440 *pCPSP1A = (unsigned int) TCB_gBlock_C - OFFSET + 3
;
441 #endif
442
443 #ifdef DAC2
444 *pSPCTL1 |= (SPTRAN | OPMODE | SLEN24 | SPEN_B |
SCHEN_B | SDEN_B) ;
445 // write to DAC2
446 *pCPSP1B = (unsigned int) TCB_gBlock_C - OFFSET + 3
;
447 #endif
448
449 #ifdef DAC3
450 *pSPCTL2 = (SPTRAN | OPMODE | SLEN24 | SPEN_A |
SCHEN_A | SDEN_A) ;
451 // write to DAC3
452 *pCPSP2A = (unsigned int) TCB_gBlock_C - OFFSET + 3
;
453 #endif
454
455 #ifdef DAC4
456 *pSPCTL2 |= (SPTRAN | OPMODE | SLEN24 | SPEN_B |
SCHEN_B | SDEN_B) ;
457 // write to DAC4
458 *pCPSP2B = (unsigned int) TCB_gBlock_C - OFFSET + 3
;

```

```

459     #endif
460 }
461
462 //
463 // =====
464 // init SRU
465 // =====
466
467 void InitSRU(){
468 //
469 // -----
470 //
471 // MCLK: The output of the 12.288 MHz xtal is either
472 // directly connected to the
473 // codec, but also connected to DAI_P06, or just
474 // to DAI_P17. This is
475 // determined by switch SW3 For this example we
476 // route the MCLK into
477 // DAI_P17 and supply the clock to the ADC via
478 // DAI_P06 by routing the
479 // signal through the SRU.
480 //
481 // Tie the pin buffer input LOW.
482 // SRU(LOW,DAI_PB17_I);
483 //
484 // Tie the pin buffer enable input LOW
485 // SRU(LOW,PBEN17_I);
486 //
487 // -----
488 //
489 // Connect the ADC: The codec drives a BCLK output to
490 // DAI pin 7, a LRCLK
491 // (a.k.a. frame sync) to DAI pin 8 and data to
492 // DAI pin 5.
493 //
494 // Connect the ADC to SPORT0, using data input
495 // A
496 //
497 // All three lines are always inputs to the
498 // SHARC so tie the pin
499 // buffer inputs and pin buffer enable inputs
500 // all low.
501 //
502 // -----

```

```

495 // Connect the ADC to SPORT0, using data input A
496
497 // Clock in on pin 7
498 SRU(DAI_PB07_0,SPORT0_CLK_I);
499
500 // Frame sync in on pin 8
501 SRU(DAI_PB08_0,SPORT0_FS_I);
502
503 // Data in on pin 5
504 SRU(DAI_PB05_0,SPORT0_DA_I);
505
506 //
-----

507 // Tie the pin buffer inputs LOW for DAI pins 5, 7
508 // and 8. Even though
509 // these pins are inputs to the SHARC, tying unused
510 // pin buffer inputs
511 // LOW is "good coding style" to eliminate the
512 // possibility of
513 // termination artifacts internal to the IC. Note
514 // that signal
515 // integrity is degraded only with a few specific SRU
516 // combinations.
517 // In practice, this occurs VERY rarely, and these
518 // connections are
519 // typically unnecessary.
520
521 SRU(LOW,DAI_PB05_I);
522 SRU(LOW,DAI_PB07_I);
523 SRU(LOW,DAI_PB08_I);
524
525 //
-----

526 // Tie the pin buffer enable inputs LOW for DAI pins 5,
527 // 6, 7 and 8 so
528 // that they are always input pins.
529
530 SRU(LOW,PBEN05_I);
531 SRU(LOW,PBEN07_I);
532 SRU(LOW,PBEN08_I);
533
534 //
-----

535 //
536 // Connect the DACs: The codec accepts a BCLK input
537 // from DAI pin 13 and
538 // a LRCLK (a.k.a. frame sync) from DAI pin 14
539 // and has four
540 // serial data outputs to DAI pins 12, 11, 10
541 // and 9
542 //

```

```

534 //          Connect DAC1 to SPORT1, using data output A
535 //          Connect DAC2 to SPORT1, using data output B
536 //          Connect DAC3 to SPORT2, using data output A
537 //          Connect DAC4 to SPORT2, using data output B
538 //
539 //          Connect the clock and frame sync inputs to
SPORT1 and SPORT2
540 //          should come from the ADC on DAI pins 7 and
8, respectively
541 //
542 //          Connect the ADC BCLK and LRCLK back out to
the DAC on DAI
543 //          pins 13 and 14, respectively.
544 //
545 //          All six DAC connections are always outputs
from the SHARC
546 //          so tie the pin buffer enable inputs all high
.
547 //
548 //
549 //
-----

550 //  Connect the pin buffers to the SPORT data lines and
ADC BCLK & LRCLK
551
552     SRU(SPORT2_DB_0,DAI_PB09_I);
553     SRU(SPORT2_DA_0,DAI_PB10_I);
554     SRU(SPORT1_DB_0,DAI_PB11_I);
555     SRU(SPORT1_DA_0,DAI_PB12_I);
556
557 //
-----

558 //  Connect the clock and frame sync input from the ADC
directly
559 //  to the output pins driving the DACs.
560
561     SRU(DAI_PB07_0,DAI_PB13_I);
562     SRU(DAI_PB08_0,DAI_PB14_I);
563     SRU(DAI_PB17_0,DAI_PB06_I);
564
565 //
-----

566 //  Connect the SPORT clocks and frame syncs to the
clock and
567 //  frame sync from the SPDIF receiver
568
569     SRU(DAI_PB07_0,SPORT1_CLK_I);
570     SRU(DAI_PB07_0,SPORT2_CLK_I);
571     SRU(DAI_PB08_0,SPORT1_FS_I);
572     SRU(DAI_PB08_0,SPORT2_FS_I);
573
574 //

```



```

575 // Tie the pin buffer enable inputs HIGH to make DAI
    pins 9-14 outputs.
576     SRU(HIGH,PBEN06_I);
577     SRU(HIGH,PBEN09_I);
578     SRU(HIGH,PBEN10_I);
579     SRU(HIGH,PBEN11_I);
580     SRU(HIGH,PBEN12_I);
581     SRU(HIGH,PBEN13_I);
582     SRU(HIGH,PBEN14_I);
583 //
-----

584 // Route SPI signals to AD1835.
585
586     SRU(SPI_MOSI_0,DPI_PB01_I)      //Connect MOSI to
        DPI PB1.
587     SRU(DPI_PB02_0 , SPI_MISO_I)    //Connect DPI PB2 to
        MISO.
588     SRU(SPI_CLK_0 , DPI_PB03_I)     //Connect SPI CLK to
        DPI PB3.
589     SRU(SPI_FLG3_0 , DPI_PB04_I)     //Connect SPI FLAG3
        to DPI PB4.
590 //
-----

591 // Tie pin buffer enable from SPI peipherals to
    determine whether they are
592 // inputs or outputs
593
594     SRU(SPI_MOSI_PBEN_0 , DPI_PBEN01_I);
595     SRU(SPI_MISO_PBEN_0 , DPI_PBEN02_I);
596     SRU(SPI_CLK_PBEN_0 , DPI_PBEN03_I);
597     SRU(SPI_FLG3_PBEN_0 , DPI_PBEN04_I);
598 //
599 //
-----

600 // UART config
601     SRU2(UART0_TX_0,DPI_PB09_I); // UART transmit signal
        is connected to DPI pin 9
602     SRU2(HIGH,DPI_PBEN09_I);
603     SRU2(DPI_PB10_0,UART0_RX_I); // connect the pin
        buffer output signal to the UART0 receive
604     SRU2(LOW,DPI_PB10_I);
605     SRU2(LOW,DPI_PBEN10_I);      // disables DPI pin10
        as input
606 }
607 //
608 //
=====

609 // IRQ's
610 //

```

```

=====
611 void SetupIRQ01 ()
612 {
613     //Enable the pins as IRQ0 and IRQ1
614     *pSYSCTL|= IRQ0EN | IRQ1EN;
615     asm ("#include <def21369.h>") ;
616     //Set the IRQ pins to be edge sensitive
617     asm ("bit set mode2 IRQ0E;") ;
618     asm ("bit set mode2 IRQ1E;") ;
619 }
620
621 void Irq0ISR (int i)
622 {
623     int leftDAC4Vol, rightDAC4Vol ;
624
625     // IRQ0 is used to decrease volume
626     SetupSPI1835 () ;
627     leftDAC4Vol = Get1835Register (RD | DACVOL_L4) ;
628     rightDAC4Vol = Get1835Register (RD | DACVOL_R4) ;
629
630     // Now decrease by a step size of 0x3F
631     leftDAC4Vol -= 0x3F ;
632     rightDAC4Vol -= 0x3F ;
633
634     if (leftDAC4Vol > 0)
635         Configure1835Register (WR | DACVOL_L4 |
636                                 leftDAC4Vol) ;
637
638     if (rightDAC4Vol > 0)
639         Configure1835Register (WR | DACVOL_R4 |
640                                 rightDAC4Vol) ;
641
642     DisableSPI1835 () ;
643 }
644
645 void Irq1ISR (int i)
646 {
647     int leftDAC4Vol, rightDAC4Vol ;
648
649     // IRQ1 is used to decrease volume
650     SetupSPI1835 () ;
651     leftDAC4Vol = Get1835Register (RD | DACVOL_L4) ;
652     rightDAC4Vol = Get1835Register (RD | DACVOL_R4) ;
653
654     // Now decrease by a step size of 0x3F
655     leftDAC4Vol += 0x3F ;
656     rightDAC4Vol += 0x3F ;
657
658     if (leftDAC4Vol < 0x3FF)
659         Configure1835Register (WR | DACVOL_L4 |
660                                 leftDAC4Vol) ;
661
662     if (rightDAC4Vol < 0x3FF)
663         Configure1835Register (WR | DACVOL_R4 |

```

```

        rightDAC4Vol) ;
661
662     DisableSPI1835 () ;
663 }
664
665 //
666 // =====
667 // UART
668 // =====
669
670 // type for the low level receive
671 typedef struct SADSPuartRecv {
672     unsigned int    val;    //!< the value
673     int             i;
674     //!< the index
675     TFkt_ADSPuartCB cb;    //!< the
676                             callback function when we received 4 bytes of
677                             data
678 } TADSPuartRecv;
679
680 TADSPuartRecv gUARTrx;
681
682 // init ADSP uart
683 //
684 //     Bits per Second  -> 19200
685 //     Data Bits        -> 8
686 //     Parity           -> odd
687 //     Stop Bits        -> 2
688 //     Flow Control     -> None
689 void initUART(TFkt_ADSPuartCB cbRXFunction) {
690     // Sets the Baud rate for UART0
691     *pUARTOLCR = UARTDLAB; //enables access to
692                             Divisor register to set baud rate
693 //
694 //     *pUARTODLL = 0x1c;    //0x21c = 540 for
695 //     divisor value and gives a baud rate of 19200 for core
696 //     clock 331.776MHz
697 //
698 //     *pUARTODLH = 0x02;
699
700     *pUARTODLL = 0x38;    //1080 = 0x438 for
701                             divisor value and gives a baud rate of 9600
702                             for core clock 331.776MHz
703     *pUARTODLH = 0x04;
704
705 //
706 //     Configures UART0 LCR
707 //     *pUARTOLCR = UARTWLS8|
708 //     word length 8
709 //     UARTPEN|
710 //     parity enable ODD parity
711 //     UARTSTB ;
712 //
713 //     Two stop bits

```

```

700         *pUARTOLCR = UARTWLS8; // 8Bit 1StopBit
701         NoParity
702
703         //enables UART0 in receive mode
704         *pUARTORXCTL = UARTEN;
705         //enables UART0 in core driven mode
706         *pUARTOTXCTL = UARTEN;
707
708         // set rx callback function and the state machine
709         gUARTrx.val = 0;
710         gUARTrx.i = 3;
711         gUARTrx.cb = cbRXFunction;
712     }
713
714     // UART isr
715     void UARTisr(int i) {
716         unsigned int v;
717         v = *pUARTORBR;
718         // shift register
719         gUARTrx.val <<= 8;
720         gUARTrx.val |= v;
721         if (gUARTrx.i) {
722             gUARTrx.i--;
723         } else {
724             gUARTrx.cb(gUARTrx.val);
725             gUARTrx.i = 3;
726             gUARTrx.val = 0;
727         }
728     }
729
730     // UART send
731     // the system encoding of int values is big endian
732     // the uart send stream put this out as little endian
733     // value: 32 - 0 Bit 44 33 22 11
734     // @ADSP: 11 22 33 44
735     // send: 44 33 22 11 = little endian
736     void sendUARTuint32Value(unsigned int v) {
737         // 44
738         // wait till the transmitter is ready
739         while ((*pUARTOLSR & UARTTHRE) == 0);
740         // mask all other bytes out and send the lowest
741         // byte
742         *pUARTOTHR = v & 0xFF;
743         v >>= 8;
744
745         // 33
746         while ((*pUARTOLSR & UARTTHRE) == 0);
747         *pUARTOTHR = v & 0xFF;
748         v >>= 8;
749
750         // 22
751         while ((*pUARTOLSR & UARTTHRE) == 0);
752         *pUARTOTHR = v & 0xFF;
753         v >>= 8;

```

```

753         // 11
754         while ((*pUARTOLSR & UARTTHRE) == 0);
755         *pUARTOTHR = v & 0xFF;
756
757         // wait till the transmitter is ready
758         while ((*pUARTOLSR & UARTTHRE) == 0);
759     }
760
761     void sendUARTFloatVector (float * pD, int amount) {
762         uint32_t * pS = (uint32_t *) pD;
763         while (amount) {
764             sendUARTuint32Value(*pS);
765             pS++;
766             amount--;
767         }
768     }
769
770     void sendUARTraw4ByteVector (uint32_t * pD, int amount)
771     {
772         while (amount) {
773             sendUARTuint32Value(*pD);
774             pD++;
775             amount--;
776         }
777     }
778
779     void sendUARTintVectorBigEndian (int32_t * pD, int
780     amount) {
781         uint32_t * pS = (uint32_t *) pD;
782         unsigned int v;
783         while (amount) {
784             // rotate bytes
785             v = (*pS & 0x000000FF) << 24;
786             v |= (*pS & 0x0000FF00) << 8;
787             v |= (*pS & 0x00FF0000) >> 8;
788             v |= (*pS & 0xFF000000) >> 24;
789             // send value
790             sendUARTuint32Value(v);
791             pS++;
792             amount--;
793         }
794     }
795
796     void sendUARTuintVectorBigEndian (uint32_t * pD, int
797     amount) {
798         uint32_t * pS = (uint32_t *) pD;
799         unsigned int v;
800         while (amount) {
801             // rotate bytes
802             v = (*pS & 0x000000FF) << 24;
803             v |= (*pS & 0x0000FF00) << 8;
804             v |= (*pS & 0x00FF0000) >> 8;
805             v |= (*pS & 0xFF000000) >> 24;

```

```

805         // send value
806         sendUARTuint32Value(v);
807         pS++;
808         amount--;
809     }
810 }
811
812 //
=====
813 // SPORT IRQs
814 //
=====
815
816     //Pointer to the blocks
817
818 unsigned int *gpProcessBuffer[3] = {gBlock_A,gBlock_C,
    gBlock_B};
819
820 // Counter to choose which buffer to process
821 volatile int gProcessBufferCounter=2;
822 // Semaphore to indicate to main that a block is ready
    for processing
823 volatile int gProcessBufferReady=0;
824
825 void TalkThroughISR(int sig_int)
826 {
827     //Increment the block pointer
828     gProcessBufferCounter++;
829     gProcessBufferCounter %= 3;
830
831     gProcessBufferReady = 1;
832 }
833
834 //
=====
835 // LED func
836 //
=====
837
838 void LEDSRUinit () {
839     // Init LED Ports
840     SRU(LOW,DPI_PB06_I);
841     // Connect GND to DPI_PB06 input (LED1)
842     SRU(LOW,DPI_PB07_I);
843     // Connect GND to DPI_PB07 input (LED2)
844     SRU(LOW,DPI_PB08_I);
845     // Connect GND to DPI_PB08 input (LED3)
846     SRU(LOW,DPI_PB13_I);
847     // Connect GND to DPI_PB13 input (LED4)
848     SRU(LOW,DPI_PB14_I);
849     // Connect GND to DPI_PB14 input (LED5)

```

```

845     SRU(LOW,DAI_PB15_I);
      // Connect GND to DAI_PB15 input (LED6)
846     SRU(LOW,DAI_PB16_I);
      // Connect GND to DAI_PB16 input (LED7)
847
848     //Enabling the Buffer using the following
      sequence: High -> Output, Low -> Input
849
850     SRU(HIGH,DPI_PBEN06_I);
      // LED 1
851     SRU(HIGH,DPI_PBEN07_I);
      // LED 2
852     SRU(HIGH,DPI_PBEN08_I);
      // LED 3
853     SRU(HIGH,DPI_PBEN13_I);
      // LED 4
854     SRU(HIGH,DPI_PBEN14_I);
      // LED 5
855     SRU(HIGH,PBEN15_I);
      // LED 6
856     SRU(HIGH,PBEN16_I);
      // LED 7
857 }
858
859 #define set_LED_1 SRU(HIGH,DPI_PB06_I)
860 #define set_LED_2 SRU(HIGH,DPI_PB07_I)
861 #define set_LED_3 SRU(HIGH,DPI_PB08_I)
862 #define set_LED_4 SRU(HIGH,DPI_PB13_I)
863 #define set_LED_5 SRU(HIGH,DPI_PB14_I)
864 #define set_LED_6 SRU(HIGH,DPI_PB15_I)
865 #define set_LED_7 SRU(HIGH,DPI_PB16_I)
866
867 #define clear_LED_1 SRU(LOW,DPI_PB06_I)
868 #define clear_LED_2 SRU(LOW,DPI_PB07_I)
869 #define clear_LED_3 SRU(LOW,DPI_PB08_I)
870 #define clear_LED_4 SRU(LOW,DPI_PB13_I)
871 #define clear_LED_5 SRU(LOW,DPI_PB14_I)
872 #define clear_LED_6 SRU(LOW,DPI_PB15_I)
873 #define clear_LED_7 SRU(LOW,DPI_PB16_I)
874
875 //
      =====
876 // init HW
877 //
      =====
878
879 void initHW(TFkt_ADSPuartCB cbRXFunction) {
880     // uart stuff
881     *pPICR2 &= ~(0x3E0); //Sets the UART0 receive
882         interrupt to P13
883
884     *pPICR2 |= (0x13<<5);

```

```

885
886
887 //Initialize PLL to run at CCLK= 331.776 MHz & SDCLK
      = 165.888 MHz.
888 //SDRAM is setup for use, but cannot be accessed
      until MSEN bit is enabled
889 InitPLL_SDRAM();
890
891 // Setting up IRQ0 and IRQ1
892 SetupIRQ01() ;
893
894 // Need to initialize DAI because the sport signals
      need to be routed
895 InitSRU();
896
897 // This function will configure the codec on the kit
898 Init1835viaSPI();
899
900 interrupt (SIG_SP0,TalkThroughISR);
901 interrupt (SIG_IRQ0, Irq0ISR) ;
902 interrupt (SIG_IRQ1, Irq1ISR) ;
903
904 *pUARTOLCR=0;
905 *pUARTOIER = UARTRBFIE; // enables UART0
      receive interrupt
906 interrupt(SIG_P13,UARTIsr);
907
908
909 // init LEDs
910 LEDSRUinit();
911
912 // init UART
913 initUART (cbRXFunction);
914 }
915
916 void startHW() {
917 // Finally setup the sport to receive / transmit the
      data
918 InitSPORT();
919 }
920
921
922 //
      =====
923 // processing
924 //
      =====
925
926
927 #ifndef INT24_MAX
928 #define INT24_MIN (-16777215-1)
929 #define INT24_MAX (16777215)
930 #endif

```



```

931
932 #define dAD1835_ChannelAmount (2)
933 #define dAD1835_leftChannelOffset (1)
934 #define dAD1835_rightChannelOffset (0)
935
936 #define dAD1835_ChannelFlag_left ('l')
937 #define dAD1835_ChannelFlag_right ('r')
938
939 // the adsp codec channel type
940 typedef struct SCodecChannel {
941     unsigned int    size;           //!<
942     char            left r=right channel channelFlag;    //!< l=
943 } TCodecChannel;
944
945 // the codec channel list
946 typedef struct SCodecChannelList {
947     TCodecChannel *    pCC;
948     int                number;
949 } TCodecChannelList;
950
951 // find the channel struct by a given channel
952 inline TCodecChannel * ADSP_getChannel (int channel) {
953     extern TCodecChannelList gADSPcodecChannels;
954     if ((channel < 0) || (channel >=
955         gADSPcodecChannels.number)) return NULL;
956     return &gADSPcodecChannels.pCC[channel];
957 }
958
959 // wait for the sample frame
960 inline void ADSP_waitForSamples () {
961     if (!gProcessBufferReady) {
962         set_LED_1;
963         while (!gProcessBufferReady) {
964             };
965             gProcessBufferReady = 0;
966             clear_LED_1;
967         }
968     }
969
970 // ADSP ADC 24Bit value format
971 // 011...11          +FS
972 // 0.....0          0
973 // 111...11          -FS
974 //
975 // reads some samples from the input channel
976 void ADSP_readSamplesFromChannel (TCodecChannel * pIC,
977     float * pBuffer, unsigned int amount, int
978     waitForNewFrame) {
979     // wait for new samples
980     if (waitForNewFrame) ADSP_waitForSamples ();
981     // after that get the pointer to the buffer

```

```

981     int * pBinSRC = (int *) gpProcessBuffer[
        gProcessBufferCounter];
982     unsigned int i;
983     // point to the first sample
984     if (dAD1835_ChannelFlag_right == pIC->
        channelFlag) {
985         pBinSRC += dAD1835_rightChannelOffset;
986     } else {
987         pBinSRC += dAD1835_leftChannelOffset;
988     }
989
990     // #pragma SIMD_for
991     for (i = 0; i < amount; i++) {
992         *pBuffer = ((float) ((int)(*pBinSRC)<<8)
            ) * (1.0/2147483648.0);
993         pBuffer++;
994         pBinSRC += dAD1835_ChannelAmount;
995     }
996 }
997
998 // writes some samples to the output channel
999 void ADSP_writesSamplesToChannel (TCodecChannel * pOC,
1000     float * pBuffer, unsigned int amount) {
1001     int * pBinSRC = (int *) gpProcessBuffer[
        gProcessBufferCounter];
1002     unsigned int i;
1003     float tv;
1004     // point to the first sample
1005     if (dAD1835_ChannelFlag_right == pOC->
        channelFlag) {
1006         pBinSRC += dAD1835_rightChannelOffset;
1007     } else {
1008         pBinSRC += dAD1835_leftChannelOffset;
1009     }
1010     // #pragma SIMD_for
1011     for (i = 0; i < amount; i++) {
1012         *pBinSRC = ((int)(2147483648.0 * pBuffer
            [i]))>>8;
1013         pBinSRC += dAD1835_ChannelAmount;
1014     }
1015 }
1016
1017 TCodecChannel gADSPcodecChannel []= {
1018     {NUM_SAMPLES/2,dAD1835_ChannelFlag_left
        }, // ADC left input
1019     {NUM_SAMPLES/2,dAD1835_ChannelFlag_left}
        // DAC4 & DAC3 left output
1020 };
1021
1022 TCodecChannelList gADSPcodecChannels = {
1023     gADSPcodecChannel,
1024     sizeof(gADSPcodecChannel) / sizeof(
        TCodecChannel)
1025 };

```

3.1.2 ANSI C strings

includes:

c-Include	c-Library	system lib
stdlib.h		yes
string.h		yes

code:

```
1 // -----
2 // TAPstringVector
3 // -----
4
5 // ap string
6 typedef struct SAPstring {
7     char * szTxt; //!< pointer to char buffer
8     size_t maxLen; //!< length of the char buffer(
9                     without the zero at the end)
10 } TAPstring;
11
12 // vector of strings type
13 typedef struct SAPstringVector {
14     TAPstring * sv; //!<
15     string vector
16     unsigned int num; //!< amount of strings
17     at the vector
18 } TAPstringVector;
19
20 // creates a AP string
21 TAPstringVector * APstringVector_create (
22     int num //!<(in)
23     number of strings at the vector
24 ) {
25     TAPstringVector * pR;
26     int i;
27
28     pR = malloc(sizeof(TAPstringVector));
29     if (!pR) return NULL;
30     pR->sv = malloc(sizeof(TAPstring)*num);
31     if (!pR->sv){
32         free (pR);
33         return NULL;
34     }
35     // init str
36     TAPstring * ps = pR->sv;
37     for (i = 0; i < num; i++) {
38         ps->szTxt = NULL;
39         ps->maxLen = 0;
40         ps++;
41     }
```

```

37     }
38     pR->num = num;
39     return pR;
40 }
41
42 // frees a AP string
43 void APstringVector_free (
44     TAPstringVector *      pSV      //!<(in)
45     pointer to the string vector
46 ) {
47     unsigned int i;
48     if (pSV->sv) {
49         TAPstring * ps = pSV->sv;
50         for (i = 0; i < pSV->num; i++) {
51             if (ps->szTxt){
52                 free(ps->szTxt);
53             }
54         }
55         free(pSV->sv);
56     }
57     free(pSV);
58 }
59
60 // resizes the amount of strings at the string vector
61 int APstringVector_resizeVector (
62     TAPstringVector *      pSV,      //!<(in)
63     pointer to the string vector
64     int
65     num                      //!<(in) number of
66     strings at the vector
67 ) {
68     if (pSV->num == num) return 0;
69     unsigned int i;
70     TAPstring * ps;
71
72     if (pSV->sv) {
73         ps = pSV->sv;
74         for (i = 0; i < pSV->num; i++) {
75             if (ps->szTxt){
76                 free(ps->szTxt);
77             }
78         }
79         free(pSV->sv);
80     }
81     pSV->sv = malloc(sizeof(TAPstring)*num);
82     if (!pSV->sv){
83         return -1;
84     }
85     // init str
86     ps = pSV->sv;
87     for (i = 0; i < num; i++) {
88         ps->szTxt = NULL;
89         ps->maxLen = 0;
90         ps++;
91     }

```

```

88         pSV->num = num;
89         return 0;
90     }
91
92     // resizes a string at the vector
93     int APstringVector_resize (
94         TAPstringVector *      pSV,      //!<(in)
95         int                    index,    //!<(in) index of the string
96         size_t                 newLen    //!<(in) new length of the string
97     ) {
98         TAPstring * ps = pSV->sv + index;
99         if (ps->maxLen < newLen) {
100             // allocate new string
101             char * nsz = malloc(sizeof(char) * (
102                 newLen + 1));
103             if (!nsz) return -1;
104             if (ps->szTxt) {
105                 // copy old string
106                 strcpy(nsz, ps->szTxt);
107                 // release old string
108                 free(ps->szTxt);
109                 ps->szTxt = NULL;
110             } else {
111                 *nsz = 0;
112             }
113             // set the new string
114             ps->szTxt = nsz;
115             // remember the length
116             ps->maxLen = newLen;
117         } else {
118             if (ps->szTxt) {
119                 // trail old string
120                 ps->szTxt[newLen] = 0;
121             }
122         }
123         return 0;
124     }
125
126     // fills the string from an extern source
127     void APstringVector_fill(
128         TAPstringVector *      pSV,      //!<(in)
129         int                    index,    //!<(in) index of the string
130         int                    iStart,   //!<(in) start index at the
131         int                    iEnd,     //!<(in) end index of the
132                                     string

```

```

131         char * pSource
           //!<(in) source from where the chars
           are copied
132     ) {
133         int i, imax;
134         char * pDest = pSV->sv[index].szTxt;
135         pDest += iStart;
136         // clip str
137         imax = pSV->sv[index].maxLen;
138         if (iEnd > imax) {
139             iEnd = imax;
140         }
141
142         for (i = iStart; i < iEnd; i++) {
143             *pDest = *pSource;
144             pDest++;
145             pSource++;
146         }
147         *pDest = 0;
148     }
149
150 // concat string 2 with string 1
151 int APstringVector_concat(
152     TAPstringVector * pSV1,
153     //!<(in) pointer to the first string
154     vector
155     int indexSV1, //!<(in) index of the
156     string at the stringarray
157     TAPstringVector * pSV2,
158     //!<(in) pointer to the second string
159     vector
160     int indexSV2 //!<(in) index of the
161     string at the stringarray
162 ) {
163     TAPstring * ps1 = pSV1->sv + indexSV1;
164     TAPstring * ps2 = pSV2->sv + indexSV2;
165     // get new size
166     size_t sl1;
167     size_t sl2;
168     sl1 = ps1->szTxt ? strlen(ps1->szTxt) : 0;
169     sl2 = ps2->szTxt ? strlen(ps2->szTxt) : 0;
170     size_t newL = sl1 + sl2;
171     // reallocate?
172     if (ps1->maxLen < newL) {
173         if (APstringVector_resize(pSV1, indexSV1
174             , newL)) return -1;
175     }
176     // copy string 2 at the end of string 1
177     strcpy(ps1->szTxt + sl1, ps2->szTxt);
178     return 0;
179 }

```

```

175
176
177 // assigns string 2 to string 1
178 int APstringVector_assign(
179     TAPstringVector *      pSV1,
180     //!(in) pointer to the first string
181     //vector
182     int
183     indexSV1,              //!(in) index of the
184     //string at the stringarray
185     TAPstringVector *      pSV2,
186     //!(in) pointer to the second string
187     //vector
188     int
189     indexSV2              //!(in) index of the
190     //string at the stringarray
191 ) {
192     TAPstring * ps1 = pSV1->sv + indexSV1;
193     TAPstring * ps2 = pSV2->sv + indexSV2;
194     // get new size
195     size_t sl2 = strlen(ps2->szTxt);
196     // reallocate?
197     if (ps1->maxLen < sl2) {
198         if (APstringVector_resize(pSV1, indexSV1
199             , sl2)) return -1;
200     }
201     // copy string 2 at the end of string 1
202     strcpy(ps1->szTxt, ps2->szTxt);
203     return 0;
204 }
205
206 // print a floating point number into the string
207 void APstringVector_printFloat(
208     TAPstringVector *      pSV,
209     //!(in) pointer to the first
210     //string vector
211     int
212     indexSV,              //!(in)
213     //index of the string at the
214     //stringarray
215     float
216     number                //!(in) number to be
217     //printed
218 ) {
219     TAPstring * ps = pSV->sv + indexSV;
220     snprintf(ps->szTxt, ps->maxLen+1, "%f", (double)
221         number);
222 }
223
224 // print a integer number into the string
225 void APstringVector_printInt(
226     TAPstringVector *      pSV,
227     //!(in) pointer to the first

```

```

212         string vector
           int
           indexSV,          //!<(in)
           index of the string at the
           stringarray
213         int
           number            //!<(in)
           number to be printed
214     ) {
215     TAPstring * ps = pSV->sv + indexSV;
216     snprintf(ps->szTxt, ps->maxLen+1, "%i", number);
217 }

```

3.1.3 AP client interface using stjSocket and APclient functions

includes:

c-Include	c-Library	system lib
pthread.h	pthread	yes
winsock2.h	ws2_32	yes
stdint.h		yes
ws2tcpip.h		yes
string.h		yes
stdio.h		yes

code:

```

1  //
   =====
2  // socket functions to communicate via UDP
3  //
   =====
4  // written by Stefan Jaritz -> prefix stj
5  // part I: defines
6
7  enum eAdminMsgTypes {
8      eAdminMsgType_login      = 'i',
9      eAdminMsgType_logout    = 'o',
10     eAdminMsgType_exit        = 'e',
11     eAdminMsgType_ack         = 'a',
12     eAdminMsgType_nack        = 'n'
13 };
14
15
16 typedef struct SstjSocket_addr {
17     struct sockaddr_in      Addr;
18     int                      len;

```



```

19 } TstjSocket_addr;
20
21
22 typedef struct SstjSocket_loginMsg {
23     uint8_t      msgID;
24     uint16_t     uuid;
25     uint16_t     dataOutPort;
26     uint16_t     dataInPort;
27 } TstjSocket_loginMsg;
28
29 typedef uint8_t TstjSocket_loginRAWmsg [7];
30
31 // creates a addinfo struct with the address of the
   local
32 struct addinfo * stjSocket_getLocalSocketAddress (
   uint16_t port);
33
34 // creates a UDP server
35 int stjSocket_createServer (
36     uint16_t                                port,
37     SOCKET *                               pS,
38     TstjSocket_addr *                       pAI
39     );
40
41 // creates a UDP client witch connects to a local server
42 int stjSocket_createClient (
43     uint16_t                                port,
44     SOCKET *                               pS,
45     TstjSocket_addr *                       pAI
46     );
47
48 // closes socket & address
49 void stjSocket_close (
50     SOCKET                                S,
51     TstjSocket_addr *                       pAI
52     );
53
54 // sends some data to an address
55 int stjSocket_send (
56     SOCKET                                S,
57     void *                                pData,
58     int                                    dataSize,
59     TstjSocket_addr *                       pAddr
60 
```

```

61         );
62
63 // receives some data and filles the address with the
64 // sender
65 int stjSocket_recv (
66     SOCKET S,
67     void *    //!< socket
68     ,        //!< data to send
69     int      //!< amount of bytes
70     ,        //!< address
71     );
72
73 // =====
74 // socket functions to communicate via UDP
75 // =====
76
77 // written by Stefan Jaritz -> prefix stj
78 // part II: implementation
79
80 // defines
81 #define dMaxHostNameChars (254)
82
83 /*
84 // creates a addinfo struct with the address of the
85 // local
86 struct addrinfo * stjSocket_getLocalSocketAddress (
87     uint16_t port) {
88     char szPath[
89         dMaxHostNameChars] = "";
90     char szDummyStr[255]
91         = "";
92     struct hostent * pHostInfo;
93     struct addrinfo hints;
94     struct addrinfo * pAI;
95
96     memset ( &hints, 0, sizeof(hints));
97     hints.ai_family = AF_UNSPEC; // IPv4 and IPv6
98     hints.ai_socktype = SOCK_DGRAM;
99     hints.ai_protocol = 0; // any protocol
100     hints.ai_flags = 0;
101
102     // resolve hostname
103     if (gethostname(szPath, dMaxHostNameChars)) goto
104     mainErrorWithMsg;
105
106     pHostInfo = gethostbyname(szPath);
107     if (!pHostInfo) goto mainErrorWithMsg;

```

```

101     sprintf(szDummyStr,"%u",(unsigned int)port);
102
103     // build address info
104     if (getaddrinfo (pHostInfo->h_name,szDummyStr,&
105         hints,&pAI)) {
106         fprintf (stderr,"getting address-
107             information of the TCP port failed (
108                 error code %s)!\n",gai_strerror(
109                     WSAGetLastError()));
110         goto mainError;
111     }
112     return pAI;
113
114 mainErrorWithMsg:
115     fprintf (stderr, "error: %s\n",gai_strerror(
116         WSAGetLastError()));
117 mainError:
118     return NULL;
119 }
120 */
121
122 // creates a UPD server
123 int stjSocket_createServer (
124     uint16_t                                port,
125     SOCKET *                                pS,
126     TstjSocket_addr *                        pAI
127     /*!(out) socket
128     /*!(out) address info
129 ) {
130     // create socket
131     *pS=socket(AF_INET,SOCK_DGRAM,0);
132     if (!*pS) goto mainErrorWithMsg;
133
134     // create address
135     pAI->Addr.sin_family=AF_INET; // AF_UNSPEC
136     pAI->Addr.sin_port=htons(port);
137     pAI->Addr.sin_addr.s_addr=ADDR_ANY;
138     pAI->len = sizeof(SOCKADDR_IN);
139
140     // bind socket to address
141     if (SOCKET_ERROR == bind(*pS,(SOCKADDR*)&(pAI->
142         Addr),pAI->len)) goto mainErrorWithMsg;
143     return 0;
144 mainErrorWithMsg:
145     printf ("error: %s\n",gai_strerror(
146         WSAGetLastError()));
147     return -1;
148 }
149
150 // creates a UDP client witch connects to a local server
151 int stjSocket_createClient (
152     uint16_t                                port,

```

```

146         //!(in) port
        SOCKET * pS,
147         //!(out) socket
        TstjSocket_addr * pAI
        //!(out) address info
148     ) {
149         char szPath[
150             dMaxHostNameChars] = "";
151         struct hostent * pHostInfo;
152         // resolve hostname
153         if (gethostname(szPath,dMaxHostNameChars)) goto
            mainErrorWithMsg;
154
155         pHostInfo = gethostbyname(szPath);
156         if (!pHostInfo) goto mainErrorWithMsg;
157
158         // create socket
159         *pS=socket(AF_INET,SOCK_DGRAM,0);
160         if (!*pS) goto mainErrorWithMsg;
161
162         // get the local ip from the host
163         char * szLocalIP;
164         szLocalIP = inet_ntoa (*(struct in_addr *)*
            pHostInfo->h_addr_list);
165
166         unsigned long addr = inet_addr(szLocalIP);
167         if ((INADDR_NONE == addr) || (INADDR_ANY == addr)
            )) {
168             closesocket(*pS);
169             printf ("unknown inet address\nerror: %s
                \n",gai_strerror(WSAGetLastError()));
170             return -2;
171         }
172
173         // create address
174         pAI->Addr.sin_family=AF_INET; // AF_UNSPEC
175         pAI->Addr.sin_port=htons(port);
176         pAI->Addr.sin_addr.s_addr= addr;
177         pAI->len = sizeof(SOCKADDR_IN);
178
179         return 0;
180 mainErrorWithMsg:
181         printf ("error: %s\n",gai_strerror(
            WSAGetLastError()));
182         return -1;
183     }
184
185     // closes socket & address
186     void stjSocket_close (
187         SOCKET S,
188         TstjSocket_addr * pAI
189         //!(in) socket
        //!(in) address info
    ) {

```

```

190         // pAI is self build so we don't need a free
191         call
192         //freeaddrinfo(gMsgServer.aiAdmin);
193         closesocket(S);
194     }
195     // sends some data to an address
196     int stjSocket_send (
197         SOCKET                                S,
198         void *                                //!< socket      pData,
199         int                                    //!< data to send
200         dataSize,                            //!< amount of bytes
201         TstjSocket_addr *                    pAddr
202         //!< address
203     ) {
204         int n;
205         n = sendto (S, (const char *) pData, dataSize,
206                     0, (struct sockaddr *)&(pAddr->Addr), pAddr->
207                     len);
208         if (n != dataSize) {
209             fprintf (stderr, "sending data failed (%i
210                       bytes send)!(error: %s)!\n", n,
211                       gai_strerror(WSAGetLastError()));
212             return -1;
213         }
214         return 0;
215     }
216     // receives some data and filles the address with the
217     sender
218     int stjSocket_recv (
219         SOCKET                                S,
220         void *                                //!< socket      pData,
221         int                                    //!< data to send
222         dataSize,                            //!< amount of bytes
223         TstjSocket_addr *                    pAddr
224         //!< address
225     ) {
226         int n;
227         n = recvfrom(S, (char *) pData, dataSize, 0, (
228             struct sockaddr *)&(pAddr->Addr), &pAddr->len
229             );
230         if (n != dataSize) {
231             fprintf (stderr, "receiving data failed
232                       (%i bytes received)!(error: %s)!\n", n,
233                       gai_strerror(WSAGetLastError()));
234             return -1;
235         }
236         return 0;
237     }

```

```

227 // =====
228 // a TCP/IP client running the RX at an
229 // own thread and use a callback-function
230 // to sign that some data are received
231 // =====
232 // part I: header defines
233
234 typedef int (* TpfktAPClientRecvCallback) (void *
235     pUserData, uint16_t number, uint8_t * pData);
236
237 typedef struct SAPClient {
238     uint16_t
239         uuid;                //!< uuid of the
240         client
241
242     SOCKET
243         sAdmin;              //!< admin socket
244         uint16_t
245             adminPort;        //!< admin port
246         TstjSocket_addr      aAdmin;
247         admin                //!< socket address of the
248
249     SOCKET
250         sDataIn;             //!< data in socket
251         uint16_t
252             dataInPort;        //!< port of the data
253             in socket
254         TstjSocket_addr      aDataIn;
255         data port            //!< socket address of the
256
257     SOCKET
258         sDataOut;            //!< data out socket
259         uint16_t
260             dataOutPort;        //!< port of the data out
261             socket
262         TstjSocket_addr      aDataOut
263         ;                    //!< socket address of the
264         data port
265
266     pthread_t
267         recvThread;          //!< receive thread
268         uint8_t *
269             recvBuffer;        //!< temporary
270             receive buffer
271         uint16_t
272             recvBufferSize;    //!< size in bytes of the
273             receive buffer
274         void *
275             pUserData;          //!< user data (can
276             be NULL)
277         TpfktAPClientRecvCallback fktRecvCB;
278             //!< receive call back funtion
279 } TAPClient;

```

```

257
258 // creates an admin client
259 int APclient_create (
260     TAPClient *
                pC,
                //!<(in/out) pointer
                to a client structure to be filled
261     uint16_t
                adminPort,
                //!<(in) the port of the
                admin server port
262     uint16_t
                recvBufferSize, //!<<
                in) size in bytes of the receive
                buffer
263     void *
                pUserData,
                //!<(in) user data
264     TpfktAPClientRecvCallback fktCB
                //!<(in)
                callback function when receiving data
265 );
266
267 //!< free's the client
268 int APclient_close (
269     TAPClient *
                pC
                //!<(in) pointer to a client
                structure
270 );
271
272
273 int APclient_send (
274     TAPClient *
                pC,
                //!<(in) pointer to a client
                structure
275     uint16_t
                num,
                //!<(in) amount of bytes to send
276     uint8_t *
                pData
                //!<(in) pointer to the data
277 );
278
279 // =====
280 // a TCP/IP client running the RX at an
281 // own thread and use a callback-function
282 // to sign that some data are received
283 // =====
284 // part II: implementation
285
286
287 // =====
288 // pre defs
289 // =====
290
291 // the thread function for receiving data
292 void * APclient_RecvThread (void *);

```

```

293
294
295 // =====
296 // functions
297 // =====
298
299
300 // creates an admin client
301 int APclient_create (
302     TAPClient *
303         pC,
304         //!(in/out) pointer
305         // to a client structure to be filled
306         uint16_t
307             adminPort,
308             //!(in) the port of the
309             // admin server port
310             uint16_t
311                 recvBufferSize,
312                 //!(in)
313                 // size in bytes of the receive
314                 // buffer
315                 void *
316                     pUserData,
317                     //!(in) user data
318                     TpfktAPClientRecvCallback
319                     fktCB
320                     //!(in)
321                     // callback function when receiving data
322 ) {
323     uint8_t
324         msgID;
325
326     // 0. save the user data
327     pC->pUserData = pUserData;
328
329     // 1. create a socket to communicate with the
330     // admin port
331     if (stjSocket_createClient(
332         adminPort,
333         &(pC->sAdmin),
334         &(pC->aAdmin)
335     )) {
336         return -1;
337     }
338
339     pC->adminPort = adminPort;
340     // 2. setup
341     // request login
342     msgID = eAdminMsgType_login;
343     if (stjSocket_send(pC->sAdmin, &msgID, 1, &(pC->
344         aAdmin))) goto sendFailed;
345     // get the ports
346     if (stjSocket_recv(pC->sAdmin, (char *)&msgID
347         , 1, &(pC->aAdmin))) goto receivedFailed;
348     if (msgID != eAdminMsgType_login) {
349         fprintf(stderr, "admin send wrong message
350             back\n");

```



```

331         goto abortAndError;
332     }
333
334     if (stjSocket_recv(pC->sAdmin,(char *)&(pC->uuid)
335         ),2,&(pC->aAdmin))) goto receivedFailed;
336     if (stjSocket_recv(pC->sAdmin,(char *)&(pC->
337         dataInPort),2,&(pC->aAdmin))) goto
338         receivedFailed;
339     if (stjSocket_recv(pC->sAdmin,(char *)&(pC->
340         dataOutPort),2,&(pC->aAdmin))) goto
341         receivedFailed;
342
343     printf("data in: %u\ndata out: %u\n", (unsigned
344         int)pC->dataInPort, (unsigned int)pC->
345         dataOutPort);
346
347     // create data sockets
348     if (stjSocket_createServer(
349         pC->dataInPort,
350         &(pC->sDataIn),
351         &(pC->aDataIn)
352     )) goto DataSocketCreationError;
353
354     // create recv buffer & threads
355     pC->recvBuffer = malloc((size_t)recvBufferSize);
356     if (!pC) goto abortAndError;
357     pC->recvBufferSize = recvBufferSize;
358     pC->fktRecvCB = fktCB;
359
360     // now we can start the handling thread
361     pthread_create(&pC->recvThread, NULL,
362         APclient_RecvThread, pC);
363
364     // send ack to server
365     msgID = eAdminMsgType_ack;
366     if (stjSocket_send(pC->sAdmin, &msgID, 1, &(pC->
367         aAdmin))) goto sendFailed;
368
369     // wait till server is ready
370     if (stjSocket_recv(pC->sAdmin, &msgID, 1, &(pC->
371         aAdmin))) goto receivedFailed;
372     if (eAdminMsgType_ack != msgID) {
373         fprintf(stderr, "admin error\n");
374         return -10;
375     }
376
377     // now open the client data port
378     if (stjSocket_createClient(
379         pC->dataOutPort,
380         &(pC->sDataOut),
381         &(pC->aDataOut)
382     )) goto DataSocketCreationError;
383
384     // some info

```

```

376     printf ("client setup with uuid=%u data-in port:%u
        data-out port:%u\n", (unsigned int)pC->uuid, (
        unsigned int)pC->dataInPort, (unsigned int)pC->
        dataOutPort);
377     fflush(stdout);
378     return 0;
379
380 abortAndError:
381     msgID = eAdminMsgType_nack;
382     stjSocket_send(pC->sAdmin, &msgID, 1, &(pC->aAdmin)
        );
383     return -2;
384
385 sendFailed:
386     fprintf(stderr, "sending to admin failed\n");
387     fprintf (stderr, "error: %s\n", gai_strerror(
        WSAGetLastError()));
388     return -3;
389
390 receivedFailed:
391     fprintf(stderr, "receiving from admin failed\n");
392     fprintf (stderr, "error: %s\n", gai_strerror(
        WSAGetLastError()));
393     return -4;
394
395 DataSocketCreationError:
396     fprintf(stderr, "creating data sockets failed\n")
        ;
397     fprintf (stderr, "error: %s\n", gai_strerror(
        WSAGetLastError()));
398     return -4;
399 }
400
401 //! free's the client
402 int APclient_close (
403     TAPClient *      pC
404     //!<(in/out) pointer to a
405     client structure to be filled
406 ) {
407     // vars
408     uint8_t          msgID;
409
410     // request logout
411     msgID = eAdminMsgType_logout;
412     if (stjSocket_send(pC->sAdmin, (char *)&msgID
413         , 1, &(pC->aAdmin))) goto sendFailed;
414     if (stjSocket_send(pC->sAdmin, (char *)&(pC->uuid)
415         , 2, &(pC->aAdmin))) goto sendFailed;
416     // wait till server finished it
417     if (stjSocket_recv(pC->sAdmin, (char *)&msgID
418         , 1, &(pC->aAdmin))) goto receivedFailed;
419     if (msgID != eAdminMsgType_ack) {
420         fprintf(stderr, "logout failed\n");
421     }

```

```

418     stjSocket_close(pC->sAdmin,&(pC->aAdmin));
419     stjSocket_close(pC->sDataIn,&(pC->aDataIn));
420     stjSocket_close(pC->sDataOut,&(pC->aDataOut));
421
422     // wait till thread is gone
423     pthread_join(pC->recvThread,NULL);
424
425
426     return 0;
427 sendFailed:
428     fprintf(stderr,"sending to admin failed\n");
429     return -3;
430
431 receivedFailed:
432     fprintf(stderr,"receiving from admin failed\n");
433     return -4;
434 }
435
436 // the thread function for receiving data
437 void * APclient_RecvThread (
438     void * pArg
439 ) {
440     uint16_t                msgLen;
441     uint8_t *               pNB;
442
443     TAPClient *             pC = pArg;
444
445     // get data
446     for (;;) {
447         // read amount of data to be received
448         if (stjSocket_recv(pC->sDataIn,&msgLen
449             ,2,&(pC->aDataIn))) goto recvError;
450         // check if we have enough memory
451         // allocated at the buffer
452         if (msgLen > pC->recvBufferSize) {
453             pNB = realloc(pC->recvBuffer,
454                 msgLen);
455             if (pNB) {
456                 pC->recvBuffer = pNB;
457                 pC->recvBufferSize =
458                     msgLen;
459             } else {
460                 fprintf (stderr,"realloc
461                     memory failed");
462                 goto recvError;
463             }
464         }
465         // transmit data
466         if (stjSocket_recv(pC->sDataIn,pC->
467             recvBuffer,msgLen,&(pC->aDataIn)))
468             goto recvError;
469         // and handle the data
470         pC->fktRecvCB(pC->pUserData, msgLen, pC
471             ->recvBuffer);

```

```

465     }
466     pthread_exit((void *)0);
467     return NULL;
468
469 recvError:
470     // close data connections
471     fprintf (stderr,"receiving data failed!");
472     pthread_exit((void *)-3);
473     return NULL;
474 }
475
476
477 int APclient_send (
478     TAPClient *          pC,
479     //!(in) pointer to a client
480     //structure
481     uint16_t             num,
482     //!(in) amount of bytes to send
483     uint8_t *            pData
484     //!(in) pointer to the data
485 ) {
486     if (stjSocket_send(pC->sDataOut,&pC->uuid,2,&(pC->aDataOut))) goto sendFailed;
487     if (stjSocket_send(pC->sDataOut,&num,2,&(pC->aDataOut))) goto sendFailed;
488     if (stjSocket_send(pC->sDataOut,pData,(int)num,&(pC->aDataOut))) goto sendFailed;
489
490     return 0;
491 sendFailed:
492     fprintf(stderr,"sending data to server failed\n");
493     return -1;
494 }

```

3.1.4 MSP430-169STK

includes:

c-Include	c-Library	system lib
stdlib.h		yes
string.h		yes
msp430x16x.h		yes

code:

```

1  //
   =====
2  // MSP 430 API

```

```

3 //
=====

4
5 #define dMSP430_LED1_ON          P3OUT &= ~BIT6
6 #define dMSP430_LED1_OFF        P3OUT |= BIT6
7 #define dMSP430_LED2_ON          P3OUT &= ~BIT7
8 #define dMSP430_LED2_OFF        P3OUT |= BIT7
9 #define dMSP430_B1               BIT5&P1IN          //B1
10    - P1.5
11 #define dMSP430_B2               BIT6&P1IN          //B2
12    - P1.6
13 #define dMSP430_B3               BIT7&P1IN          //B3
14    - P1.7
15 #define dMSP430_E_HIGH           P4OUT |= BIT1
16 #define dMSP430_E_LOW           P4OUT &= ~BIT1
17 #define dMSP430_RS_HIGH          P4OUT |= BIT3
18 #define dMSP430_RS_LOW           P4OUT &= ~BIT3
19 #define dMSP430_LCD_Data         P4OUT
20 #define dMSP430_LCD_LIGHT_ON     P4OUT |= BIT0
21 #define dMSP430_LCD_LIGHT_OFF    P4OUT &= ~BIT0
22
23 #define dMSP430_INPUT             0
24 #define dMSP430_OUTPUT            0xff
25 #define dMSP430_ON                1
26 #define dMSP430_OFF              0
27 #define dMSP430_BUF_SIZE          25
28
29 #define dMSP430__100us             7                //7
30    cycles *12 + 20 = 104 / 104*1us = 104us
31
32 //NAND FLASH
33 #define dMSP430_MAX_BLOCK_NUMB    1024
34
35 #define dMSP430_TRANS_LDY          50
36 #define dMSP430_WRITE_DLY         400
37 #define dMSP430_ERASE_DLY         4000
38
39 #define dMSP430_OUT_PORT           P5OUT
40 #define dMSP430_IN_PORT            P5IN
41 #define dMSP430_IO_DIR             P5DIR
42
43 #define dMSP430__CE_ON             P2OUT &=
44    ~BIT0
45 #define dMSP430__CE_OFF            P2OUT |=
46    BIT0
47 #define dMSP430__RE_ON             P2OUT &=
48    ~BIT1
49 #define dMSP430__RE_OFF            P2OUT |=
50    BIT1
51 #define dMSP430__WE_ON             P2OUT &=
52    ~BIT2
53 #define dMSP430__WE_OFF            P2OUT |=
54    BIT2
55

```

```

46 #define dMSP430_ALE_ON P2OUT |=
    BIT3
47 #define dMSP430_ALE_OFF P2OUT &=
    ~BIT3
48 #define dMSP430_CLE_ON P2OUT |=
    BIT4
49 #define dMSP430_CLE_OFF P2OUT &=
    ~BIT4
50
51 #define dMSP430_R_B P2IN &
    BIT7
52 #define dMSP430_DALLAS P2IN &
    BIT5
53
54 #define dMSP430_READ_SPARE 0x50
55 #define dMSP430_READ_0
    0x00
56 #define dMSP430_READ_1
    0x01
57 #define dMSP430_READ_STATUS 0x70
58
59 #define dMSP430_WRITE_PAGE 0x80
60 #define dMSP430_WRITE_AKN 0x10
61
62 #define dMSP430_ERASE_BLOCK 0x60
63 #define dMSP430_ERASE_AKN 0xD0
64
65 #define dMSP430_DEV_ID
    0x90
66
67 #define dMSP430_SAMSUNG_ID 0xECE6
68
69
70 // =====
71 // helper
72 // =====
73
74 // delay cpu cycles
75 void msp430_Delay (unsigned int cycles)
76 {
77     unsigned char k;
78     for (k=0 ; k != cycles; k++); //20+a*12 cycles
79     (for 1MHz)
80 }
81
82 // delay a given time
83 void msp430_DelayN100us(unsigned char n)
84 {
85     unsigned char j;
86     for (j=0; j!=n; ++j) msp430_Delay (
87         dMSP430__100us);
88 }
89
90 // =====
91 // LCD

```

```

90 // =====
91
92 enum eMSP430_LCDcommands {
93     eMSP430_LCDcom_clear           = 0x01,
94     eMSP430_LCDcom_returnHome     = 0x02,
95     eMSP430_LCDcom_entryMode      = 0x04,
96     eMSP430_LCDcom_display        = 0x08,
97     eMSP430_LCDcom_cursorDisplay= 0x10,
98     eMSP430_LCDcom_function       = 0x20,
99     eMSP430_LCDcom_setCGram       = 0x40,
100    eMSP430_LCDcom_setDDram       = 0x80
101 };
102
103
104 void msp430_LCD_E()
105 {
106     dMSP430_E_HIGH;           //toggle E for LCD
107     _NOP();
108     _NOP();
109     dMSP430_E_LOW;
110 }
111
112 // sends a char to the display
113 void msp430_LCD_sendChar (unsigned char d)
114 {
115     unsigned char temp;
116
117     msp430_DelayN100us(5);           //0.5ms
118     temp = d & 0xf0;                 //get upper
119                                     nibble
120     dMSP430_LCD_Data &= 0x0f;
121     dMSP430_LCD_Data |= temp;
122     dMSP430_RS_HIGH;                 //set
123                                     LCD to data mode
124     msp430_LCD_E();                 //
125                                     toggle E for LCD
126     temp = d & 0x0f;
127     temp = temp << 4;                 //get down
128                                     nibble
129     dMSP430_LCD_Data &= 0x0f;
130     dMSP430_LCD_Data |= temp;
131     dMSP430_RS_HIGH;                 //set
132                                     LCD to data mode
133     msp430_LCD_E();                 //
134                                     toggle E for LCD
135 }
136
137 // sends a command to the LCD controller
138 void msp430_LCD_sendCmd (unsigned char e)
139 {
140     unsigned char temp;
141
142     msp430_DelayN100us(10);           //10ms
143     temp = e & 0xf0;                 //get upper
144                                     nibble

```

```

138     dMSP430_LCD_Data &= 0x0f;
139     dMSP430_LCD_Data |= temp;                                //send
140     CMD to LCD
141     dMSP430_RS_LOW;                                          //set
142     LCD to CMD mode
143     msp430_LCD_E();                                          //
144     toggle E for LCD
145     temp = e & 0x0f;
146     temp = temp << 4;                                        //get down
147     nibble
148     dMSP430_LCD_Data &= 0x0f;
149     dMSP430_LCD_Data |= temp;
150     dMSP430_RS_LOW;                                          //set
151     LCD to CMD mode
152     msp430_LCD_E();                                          //
153     toggle E for LCD
154 }
155 // cmd clear
156 inline void msp430_LCD_cmdClear () {
157     msp430_LCD_sendCmd(eMSP430_LCDcom_clear);
158 }
159 // cmd cur. home
160 inline void msp430_LCD_cmdCurHome () {
161     msp430_LCD_sendCmd(eMSP430_LCDcom_returnHome);
162 }
163 // cmd entry mode (if cursor is shifted)
164 inline void msp430_LCD_cmdEntry (
165     unsigned char incrFlag,    //!< if
166     1 increment cursor, else decrement
167     unsigned char enable      //!< if
168     set incr/decr is enabled
169 ) {
170     unsigned char cmd;
171
172     cmd = eMSP430_LCDcom_entryMode;
173     if (incrFlag) cmd |= 0x2;
174     if (enable) cmd |= 0x1;
175     msp430_LCD_sendCmd(cmd);
176 }
177 // cmd set display on, show cursor, flash cursor
178 inline void msp430_LCD_cmdDisplay (
179     unsigned char displayOn,    //!< if
180     1 display is turned on
181     unsigned char cursorOn,    //!< if
182     1 the cursor is set on
183     unsigned char cursorFlashOn    //!< if
184     1 the cursor flashes
185 ) {
186     unsigned char cmd;
187
188     cmd = eMSP430_LCDcom_display;

```



```

182         if (displayOn) cmd |= 0x4;
183         if (cursorOn) cmd |= 0x2;
184         if (cursorFlashOn) cmd |= 0x1;
185         msp430_LCD_sendCmd(cmd);
186     }
187
188     // cmd shift cursor
189     inline void msp430_LCD_cmdShiftCursor (
190         unsigned char leftFlag, //!< if 1 cursor
191         unsigned char num          //!<
192         number of shifts
193     ) {
194         unsigned char cmd,n;
195
196         cmd = eMSP430_LCDcom_cursorDisplay;
197         if (!leftFlag) cmd |= 0x4;
198         for (n = 0; n < num; n++) {
199             msp430_LCD_sendCmd(cmd);
200         }
201
202     // cmd shift display
203     inline void msp430_LCD_cmdShiftDisplay (
204         unsigned char leftFlag, //!< if 1 cursor
205         unsigned char num          //!<
206         number of shifts
207     ) {
208         unsigned char cmd,n;
209
210         cmd = eMSP430_LCDcom_cursorDisplay | 0x8;
211         if (!leftFlag) cmd |= 0x4;
212         for (n = 0; n < num; n++) {
213             msp430_LCD_sendCmd(cmd);
214         }
215
216     // cmd shift display
217     inline void msp430_LCD_cmdFunction (
218         unsigned char datamode,          //!< 0=4
219         bit 1=8bit mode
220         unsigned char displayLines      //!< 0=1
221         line, 1=2 lines
222     ) {
223         unsigned char cmd;
224
225         cmd = eMSP430_LCDcom_function;
226         if (datamode) cmd |= 0x10;
227         if (displayLines) cmd |= 0x08;
228         msp430_LCD_sendCmd(cmd);
229     }
230
231     // init the LCD display

```

```

231 void msp430_LCD_init()
232 {
233     dMSP430_RS_LOW;
234
235     //Delay 100ms
236     msp430_DelayN100us(250);
237     msp430_DelayN100us(250);
238     msp430_DelayN100us(250);
239     msp430_DelayN100us(250);
240
241     // setup
242     dMSP430_LCD_Data |= BIT4 | BIT5;           //D7
243     -D4 = 0011
244     dMSP430_LCD_Data &= ~BIT6 & ~BIT7;
245
246     msp430_LCD_E();                             //
247     toggle E for LCD
248     msp430_DelayN100us(100);                     //10
249     ms
250     msp430_LCD_E();                             //
251     toggle E for LCD
252     msp430_DelayN100us(100);                     //10
253     ms
254     msp430_LCD_E();                             //
255     toggle E for LCD
256     msp430_DelayN100us(100);                     //10
257     ms
258     dMSP430_LCD_Data &= ~BIT4;                   //D7
259     -D4 = 0010
260     msp430_LCD_E();                             //
261     toggle E for LCD
262
263     msp430_LCD_cmdFunction(0,1);
264     msp430_LCD_cmdDisplay(1,1,1);
265     msp430_LCD_cmdClear();
266 }
267
268 // writes a string at the display
269 void msp430_LCD_print (
270     unsigned char    x,
271     unsigned char    y,
272     char *           szStr
273 ) {
274     msp430_LCD_cmdClear();
275     msp430_LCD_cmdCurHome();
276
277     while (*szStr) {
278         msp430_LCD_sendChar(*szStr);
279         szStr++;
280     }
281 }
282
283 // =====
284 // UART

```

```

277 // =====
278
279 // type for the fifo structure
280 typedef struct Sstj32BitFIFO {
281     uint32_t *      buffer;
282     uint32_t *      pBufferEnd; // pointer
283         with the end value of the buffer
284     uint32_t *      pW;          // write
285         pointer
286     uint32_t *      pR;          // read
287         pointer
288     // write element
289     unsigned int    eC;          // element
290         counter
291     // status
292     unsigned int    num;         // amount of elements at
293         the buffer
294     unsigned int    numMax;     // maximum of elements
295 } Sstj32BitFIFO;
296
297 // inits the fifo
298 inline int stjFIFO_init (
299     Sstj32BitFIFO * pFIFO,
300     int             elements
301 ) {
302     pFIFO->buffer = malloc(elements*sizeof(uint32_t)
303 );
304     if (!pFIFO->buffer) return -1;
305
306     pFIFO->numMax = elements;
307
308     // setup the fifo
309     pFIFO->pBufferEnd = pFIFO->buffer + elements;
310     pFIFO->pW = pFIFO->buffer;
311     pFIFO->pR = pFIFO->buffer;
312
313     pFIFO->eC = 3;
314
315     pFIFO->num = 0;
316     while (elements) {
317         elements--;
318         *pFIFO->pW = 0;
319         pFIFO->pW++;
320     }
321     pFIFO->pW = pFIFO->buffer;
322     return 0;
323 }
324
325 // frees the fifo
326 inline int stjFIFO_free (
327     Sstj32BitFIFO * pFIFO
328 ) {
329     free (pFIFO->buffer);
330     pFIFO->buffer = NULL;
331 }

```

```

326         pFIFO->numMax = 0;
327         return 0;
328     }
329
330
331     // writes a char to the buffer
332     inline void stjFIFO_writeChar(Sstj32BitFIFO * pFIFO,
333         uint8_t d) {
334         *pFIFO->pW |= d;
335         // some checks
336
337         // 1. have we written 4 bytes?
338         if (pFIFO->eC) {
339             // no - do some shift stuff
340             pFIFO->eC--;
341             *(pFIFO->pW) <= 8;
342         } else {
343             pFIFO->eC = 3;
344             // yes - set that we got a new element
345             pFIFO->num++;
346             pFIFO->pW++;
347             if (pFIFO->pW == pFIFO->pBufferEnd) {
348                 pFIFO->pW = pFIFO->buffer;
349             }
350         }
351     }
352
353     // writes a char to the buffer
354     inline void stjFIFO_writeCharWithRotation(Sstj32BitFIFO
355         * pFIFO, uint8_t d) {
356         *pFIFO->pW |= (((uint32_t)d) << 24);
357         // some checks
358
359         // 1. have we written 4 bytes?
360         if (pFIFO->eC) {
361             // no - do some shift stuff
362             pFIFO->eC--;
363             *(pFIFO->pW) >= 8;
364         } else {
365             pFIFO->eC = 3;
366             // yes - set that we got a new element
367             pFIFO->num++;
368             pFIFO->pW++;
369             if (pFIFO->pW == pFIFO->pBufferEnd) {
370                 pFIFO->pW = pFIFO->buffer;
371             }
372         }
373     }
374
375     // read a element(uint32_t) from the buffer
376     inline int stjFIFO_readElement (Sstj32BitFIFO * pFIFO,
377         uint32_t * pRes) {
378         // if there are no entries report error
379         if (!pFIFO->num) return -1;

```

```

378         // ok let's read a value
379         *pRes = *pFIFO->pR;
380         *pFIFO->pR = 0;
381         pFIFO->pR++;
382         pFIFO->num--;
383         if (pFIFO->pR == pFIFO->pBufferEnd) {
384             pFIFO->pR = pFIFO->buffer;
385         }
386
387         return 0;
388     }
389
390     Sstj32BitFIFO gMsp430_uartFIFO;
391
392     // init UART0 port
393     void msp430_UART_init(int fifoElements)
394     {
395
396         P3SEL |= 0x30;                                // P3.4 =
397         USART0_TXD, P3.5 = USART0_RXD                // P3.4
398         P3DIR |= 0x10;                                // P3.4
399         output direction
400
401         // setup UART0
402         UCTL0 = CHAR;                                // 8-bit
403         character
404
405         UTCTL0 = SSEL1;                                // UCLK =
406         XT2
407         UBR00 = 0x41;                                // 8 000
408         000/9600
409         UBR10 = 0x03;                                //
410         UMCTL0 = 0x0;
411
412         ME1 |= UTXE0 + URXE0;                        // Enabled
413         USART0_TXD/RXD
414         IE1 |= URXIE0;                                // Enabled
415         USART0_RX interrupt
416
417         // setup the fifo
418         stjFIFO_init(&gMsp430_uartFIFO, fifoElements);
419     }
420
421     // UART rx irq
422     #pragma vector=UART0RX_VECTOR
423     __interrupt void msp430_UART_RXISR (void)
424     {
425         unsigned char v;
426         _NOP();
427         // save value
428         v = RXBUF0;
429         stjFIFO_writeChar(&gMsp430_uartFIFO, v);
430     }
431
432     // send a bytes via uart

```

```

426 void msp430_UART_send (unsigned char * pD, unsigned int
    amount)
427 {
428     unsigned int c;
429
430     for (c = 0; c < amount; c++) {
431         // wait till tx buffer is ready
432         while ((IFG1 & UTXIFG0) == 0);
433         // copy data
434         TXBUF0 = *pD;
435         pD++;
436     }
437     // wait till transfer has finished
438     while ((IFG1 & UTXIFG0) == 0);
439 }
440
441 // =====
442 // AD/DA converter
443 // =====
444
445 // init DA converter
446 void msp430_DAC_init ()
447 {
448     DAC12_OCTL = DAC12SREF1 + /*DAC12RES + */
449                     DAC12IR + DAC12AMP_7;
450                                     //Ve REF+, 8-bit
451                                     resolution
452     DAC12_1CTL = DAC12SREF1 + /*DAC12RES + */
453                     DAC12IR + DAC12AMP_7;
454                                     //Ve REF+, 8-bit
455                                     resolution
456 // DAC12_OCTL = DAC12SREF1 + DAC12RES + DAC12IR +
457 // DAC12AMP_7; //Ve REF+, 8-bit
458 // DAC12_1CTL = DAC12SREF1 + DAC12RES + DAC12IR +
459 // DAC12AMP_7;
460 }
461
462 // init AD converter
463 void msp430_ADC_init ()
464 {
465     ADC12CTL0 = SHT0_0 + ADC12ON; // Set
466                                     sampling time, turn on ADC12
467     ADC12CTL1 = SHP; // Use
468                                     sampling timer
469     //ADC12IE = 0x01; //
470                                     Enable interrupt
471     ADC12MCTL0 = SREF_7; //VR+ =
472                                     VeREF+ and VR = VREF/VeREF
473     ADC12CTL0 |= ENC; //
474                                     Conversion enabled
475     P6SEL = BIT5 + BIT4 + BIT3 + BIT2 + BIT1 + BIT0;
476                                     // P6.4 ADC option

```

```

465         select
466     }
467     // set AD channel
468 void msp430_setADChannel (unsigned int channel)
469 {
470     ADC12CTL0 &= ~ENC;
471                                     //disable
472     conversion
473     ADC12MCTL0 &= 0xfff8;
474                                     //clear
475     select channel bits
476     ADC12MCTL0 |= channel;
477                                     //select
478     channel
479     ADC12CTL0 |= ENC;
480                                     //enable
481     conversion
482     ADC12CTL0 |= ADC12SC;
483                                     //Sampling
484     open
485     while ((ADC12CTL1 & ADC12BUSY) != 0);
486 }
487
488 // =====
489 // NAND FLASH
490 // =====
491
492 //pull flash pins to inactive condition
493 void msp430_Flash_inactive() {
494     dMSP430_IO_DIR=dMSP430_INPUT;           //IO is
495     inputs
496     dMSP430__CE_OFF;           // =1
497     dMSP430__RE_OFF;           // =1
498     dMSP430__WE_OFF;           // =1
499     dMSP430__ALE_OFF;           // =0
500     dMSP430__CLE_OFF;           // =0
501 }
502
503 // write a data byte to the flash
504 void msp430_Flash_writeByte (unsigned char d) {
505     dMSP430_IO_DIR=dMSP430_OUTPUT;           //IO is
506     outputs
507     dMSP430__WE_ON;
508     dMSP430_OUT_PORT=d;
509     dMSP430__WE_OFF;           //latch data
510 }
511
512 // reads a byte from the flash
513 unsigned char msp430_Flash_readByte()
514 {
515     unsigned char f;
516
517     dMSP430_IO_DIR=dMSP430_INPUT;           //IO is
518     inputs

```

```

506     dMSP430__RE_ON;
507     f=dMSP430_IN_PORT;
508     dMSP430__RE_OFF;           //read data
509     return f;
510 }
511
512 // write a block to the flash
513 unsigned char msp430_Flash_write(
514     unsigned char page,
515     unsigned char colAddr,
516     unsigned char rowAddLow,
517     unsigned char rowAddHigh,
518     unsigned char num,
519     unsigned char * pD
520 ) {
521     unsigned char k, l;
522
523     msp430_Flash_inactive();
524     dMSP430_CLE_ON;
525     dMSP430__CE_ON;
526     msp430_Flash_writeByte(page);
527     dMSP430_CLE_OFF;
528     dMSP430_ALE_ON;
529     msp430_Flash_writeByte(colAddr);
530     msp430_Flash_writeByte(rowAddLow);
531     msp430_Flash_writeByte(rowAddHigh);
532     dMSP430_ALE_OFF;
533     for (k=0; k != num; k++) {
534         msp430_Flash_writeByte(*pD);
535         pD++;
536     }
537     dMSP430_CLE_ON;
538     msp430_Flash_writeByte(dMSP430_WRITE_AKN);
539     while ((dMSP430_R_B) == 0);
540     msp430_Flash_writeByte(dMSP430_READ_STATUS);
541     dMSP430_CLE_OFF;
542     l = msp430_Flash_readByte();
543     msp430_Flash_inactive();
544     return l;
545 }
546
547 // read a block of bytes from the flash
548 void msp430_Flash_read (
549     unsigned char colAddr,
550     unsigned char rowAddLow,
551     unsigned char rowAddHigh,
552     unsigned char num,
553     unsigned char * pD
554 )
555 {
556     unsigned char n;
557
558     msp430_Flash_inactive();
559     dMSP430_CLE_ON;
560     dMSP430__CE_ON;

```



```

561     msp430_Flash_writeByte(dMSP430_READ_0);
562     dMSP430_CLE_OFF;
563     dMSP430_ALE_ON;
564     msp430_Flash_writeByte(colAddr);
565     msp430_Flash_writeByte(rowAddLow);
566     msp430_Flash_writeByte(rowAddHigh);
567     dMSP430_ALE_OFF;
568     while ((dMSP430_R_B) == 0);
569     for (n=0; n != num; n++) {
570         *pD = msp430_Flash_readByte();
571         pD++;
572     }
573     msp430_Flash_inactive();
574 }
575
576 unsigned char msp430_Flash_erase(
577     unsigned char blockAddLow,
578     unsigned char blockAddHigh
579 ) {
580     unsigned char m;
581
582     msp430_Flash_inactive();
583     dMSP430_CLE_ON;
584     dMSP430_CE_ON;
585     msp430_Flash_writeByte(dMSP430_ERASE_BLOCK);
586     dMSP430_CLE_OFF;
587     dMSP430_ALE_ON;
588     msp430_Flash_writeByte(blockAddLow);
589     msp430_Flash_writeByte(blockAddHigh);
590     dMSP430_ALE_OFF;
591     dMSP430_CLE_ON;
592     msp430_Flash_writeByte(dMSP430_ERASE_AKN);
593     while ((dMSP430_R_B) == 0);
594     msp430_Flash_writeByte(dMSP430_READ_STATUS);
595     dMSP430_CLE_OFF;
596     m = msp430_Flash_readByte();
597     msp430_Flash_inactive();
598     return m;
599 }
600
601 // =====
602 // HW init
603 // =====
604
605 void msp430_start() {
606     // Stop watchdog timer
607     WDTCTL = WDTPW + WDT HOLD;
608
609     //XT2-ON
610     BCSCTL1 &= ~BIT7;
611     //XT2 is SMCLK
612     BCSCTL2 |= BIT3;
613
614     //hardware init
615     // 1. configure I/O Pins

```

```

616         P1DIR=BIT0;
617
618         //NAND FLASH init
619         P2OUT=0x07;
620         P2DIR=0x1F;
621
622         //LED1
623         P3OUT = BIT6 | BIT7;
624         //LED2
625         P3DIR = BIT6 | BIT7;
626
627         //LCD init
628         P4OUT = 0;
629         P4DIR = 0xff;
630     }
631
632 void msp430_initHW (int fifoElements) {
633     //configure modules
634
635     //1 first the UART!
636     msp430_UART_init(fifoElements);
637     //2nd the LCD, if switched with UART the port
638     //    sends a ghost sign because Port 3 is used by
639     //    LCD and UART
640     msp430_LCD_init();
641     //3th DAC
642     msp430_DAC_init();
643     // 4th ADC
644     msp430_ADC_init();
645
646     // Enable interrupts
647     _EINT();
648 }

```

3.1.5 audio dynamic processing (generic)

includes:

c-Include	c-Library	system lib
math.h	m	yes

code:

```

1 // =====
2 // dynamics processing (DynProc)
3 // =====
4 // Literature:
5 // Udo Zlzer, Digitale Audiosignal Verarbeitung, 3.
6 // Auflage
7 // Udo Zlzer, DAFX - Digital Audio Effects

```

```

8
9 // short:
10 //      AT - attack time
11 //      RT - release time
12 //      TAV - average time
13 //      LT- limiter threshold
14 //      LS - limiter slope
15
16
17 // -----
18 // helper
19 // -----
20
21 float DynProc_calcSlope (
22     float x1db, float y1db,
23     float x2db, float y2db
24 );
25
26 float DynProc_calcThreshold (
27     float xDB,
28     float xMax
29 );
30
31 float DynProc_calcDB (float v);
32
33 float DynProc_calcValueFromDB (float dbVlaue, float xMax
34 );
35
36 //! calc time parameter (AT, RT, TAV)
37 float DynProc_calcTimeParameter (
38     float Ta,      //!< (in) sampling
39     float period,
40     float t,      //!< (in) time
41     float parameter
42 );
43
44 float DynProc_calcTimeFromTimeparameter (
45     float Ta,
46     float Tval
47 );
48
49 // -----
50 // RMS
51 // -----
52 // RMS calc after Zlzer @p. 238
53
54 // structure for RMS calculation
55 typedef struct SDynProc_RMS {
56     float TAV;      //!< time
57     float average;
58     float oneMinusTAV; //!< 1-TAV
59     float x20ld;      //!< x(n-1)
60 } TDynProc_RMS;

```

```

59  ///! init rms
60  void DynProc_InitRMS (
61      TDynProc_RMS * pRMS,    ///!< (in/out) structure
        to be filled
62      float          TAV          ///!< (in
        ) averaging time coefficient
63  );
64
65  ///! calc rms for one step (!rms = x!)
66  float DynProc_calcRMS (
67      TDynProc_RMS * pRMS,    ///!< (in)
        structure to be filled
68      float          x
        ///!< (in) input signal witch should
        be rms
69  );
70
71  // -----
72  // Peak
73  // -----
74  // Peak calc after Zlzer
75
76  // structure for Peak calculation
77  typedef struct SDynProc_Peak {
78      float    AT;                ///!< attack time
79      float    RT;                ///!< release
        time
80      float    oneMinusAT;        ///!< 1 - AT
81      float    oneMinusRT;        ///!< 1 - RT
82      float    peak;              ///!< peak
83  } TDynProc_Peak;
84
85
86  ///! init peak
87  void DynProc_InitPeak (
88      TDynProc_Peak * pPeak,    ///!< (in/out) structure
        to be filled
89      float          AT,          ///!< (in
        ) attack time
90      float          RT          ///!< (in
        ) release time
91  );
92
93  ///! calc peak for one step
94  float DynProc_calcPeak (
95      TDynProc_Peak * pPeak,    ///!< (in)
        structure to be filled
96      float          x
        ///!< (in) input signal witch should
        be rms
97  );
98
99
100 // -----
101 // smooth gain

```

```

102 // -----
103 // smooth the gain value
104 // idea: use a hysteresis curve
105 //
106 // formula:  $g(n) = (1-k) * g(n - 1) + k * f(n)$ 
107 //
108 //  $k = AT$  or  $k = RT$ 
109 //  $k = (f(n) > f(n-1)) ? AT : RT$ 
110 // if new value is over the old value -> attack
111 // else -> release
112
113 // structure for attack and release time adjustment
    calculation
114 typedef struct SDynProc_SmoothG {
115     float          AT;                //!<
        attack time
116     float          RT;                //!<
        release time
117     float          fOld;             //!<  $f(n-1)$ 
118     float          gOld;             //!<  $g(n-1)$ 
119 } TDynProc_SmoothG;
120
121 //! init AT/RT
122 void DynProc_InitSmoothG (
123     TDynProc_SmoothG *    pSG,        //!< (in/out)
        structure to be filled
124     float                AT,
        //!< (in) attack time
125     float                RT
        //!< (in) release time
126 );
127
128 //! init AT/RT
129 float DynProc_calcSmoothG (
130     float                fn,
        //!< (in) input
131     TDynProc_SmoothG *    pSG        //!< (in)
        structure to be filled
132 );
133
134 // -----
135 // limiter
136 // -----
137 // LT - limiter treshold (where the limiter starts
    working)
138 // LS - limiter slope (how fast the limiter works)
139
140 // structure for dynamic range controler calculation
141 typedef struct SDynProc_Limiter {
142     float          LTlog;
        //!< limiter threshold LT
143     float          LS;
        //!< limiter slope LS
144     TDynProc_Peak    peakDetector;    //!< the
        peak detector

```

```

145     TDynProc_SmoothG      SmoothG;
146     //!< AT/RT block
147     float                delay;
148     //!< delay at the input - output lane
149 } TDynProc_Limiter;
150
151 //! init AT/RT
152 void DynProc_InitLimiter (
153     TDynProc_Limiter *      pLim,
154     //!< (in/out) structure to be filled
155     float                peakAT,
156     //!< (in) peak attack time
157     float                peakRT,
158     //!< (in) peak release time
159     float                smoothAT
160     , //!< (in) smoothing attack
161     time
162     float                smoothRT
163     , //!< (in) smoothing release
164     time
165     float                LT,
166     //!< (in) limiter
167     threshold
168     float                LS
169     //!< (in) limiter
170     slope
171 );
172
173 //! init AT/RT
174 float DynProc_calcLimiter (
175     TDynProc_Limiter *      pLim, //!< (in
176     ) the limiter
177     float                x
178     //!< (in) input signal
179 );
180
181 // -----
182 // compressor
183 // -----
184 // CT - compressor treshold (where the compressor starts
185 //      working)
186 // CS - compressor slope (how fast the compressor works)
187
188 // structure for dynamic range controller calculation
189 typedef struct SDynProc_Compressor {
190     float                CTlog;
191     //!< compressor threshold
192     float                CSlog;
193     //!< compressor slope
194     TDynProc_RMS
195     //!< RMS unit
196     TDynProc_SmoothG      SmoothG;
197     //!< AT/RT block
198     float                delay[2];
199     //!< delay at the input - output lane

```

```

179 } TDynProc_Compressor;
180
181
182
183 //! init compressor
184 void DynProc_InitCompressor (
185     TDynProc_Compressor *    pComp,          //!< (in
186     /out) structure to be filled
187     float                    rmsTAV,
188     //!< (in) rms time average
189     coefficient
190     float                    smoothAT
191     , //!< (in) smoothing attack time
192     float                    smoothRT
193     , //!< (in) smoothing release time
194     float                    CT,
195     //!< (in) compressor threshold
196     float                    CS
197     //!< (in) compressor slope
198 );
199
200 //! calc compressor
201 float DynProc_calcCompressor (
202     TDynProc_Compressor *    pComp, //!< (in
203     ) the compressor
204     float
205     x //!< (in) input
206     signal
207 );
208
209 // -----
210 // expander
211 // -----
212 // ET - expander threshold (where the expander starts
213 // working)
214 // ES - expander slope (how fast the expander works)
215
216 // structure for dynamic range controller calculation
217 typedef struct SDynProc_Expander {
218     float                    ETlog;
219     //!< expander threshold log10 (ET)
220     float                    ESlog;
221     //!< expander slope log10 (ES)
222     TDynProc_RMS             RMS;
223     //!< RMS unit
224     TDynProc_SmoothG         SmoothG;
225     //!< AT/RT block
226     float                    delay[2];
227     //!< delay at the input - output lane
228 } TDynProc_Expander;
229
230
231 //! init expander
232 void DynProc_InitExpander (
233     TDynProc_Expander *    pExp,          //!< (in
234     /out) structure to be filled

```

```

217         float                rmsTAV,
218         //!< (in) rms time average coefficient
219         float                smoothAT,
220         //!< (in) smoothing attack time
221         float                smoothRT,
222         //!< (in) smoothing release time
223         float                ET,
224         //!< (in) expander treshold in dB
225         float                ES
226         //!< (in) expander slope in dB
227 );
228
229 //! calc expander
230 float DynProc_calcExpander (
231     TDynProc_Expander *    pExp,    //!< (in
232     ) the expander
233     float
234     x                      //!< (in) input
235     signal
236 );
237
238 // -----
239 // noisegate
240 // -----
241 // NT - noisegate threshold (till the noisegate works)
242
243 // structure for dynamic range controller calculation
244 typedef struct SDynProc_Noisegate {
245     float                NTlog;
246     //!< noisegate threshold log10 (NT)
247     float                NSlog;
248     //!< noisegate slope log10 (NS)
249     TDynProc_RMS         RMS;
250     //!< RMS unit
251     TDynProc_SmoothG     SmoothG;
252     //!< AT/RT block
253     float                delay[2];
254     //!< delay at the input - output lane
255 } TDynProc_Noisegate;
256
257 //! init noisegate
258 void DynProc_InitNoisegate (
259     TDynProc_Noisegate *    pNG,    //!< (in
260     /out) structure to be filled
261     float                rmsTAV,
262     //!< (in) rms time average
263     coefficient
264     float                smoothAT
265     ,    //!< (in) smoothing attack time
266     float                smoothRT
267     ,    //!< (in) smoothing release time
268     float                NT,
269     //!< (in) noisegate threshold
270     float                NS
271     ,    //!< (in) noisegate slope

```



```

252 );
253
254 //! calc noisegate
255 float DynProc_calcNoisegate (
256     TDynProc_Noisegate *    pNG,    //!< (in
        ) the noisegate
257     float
        x                //!< (in) input
        signal
258     );
259
260 // =====
261 // dynamics processing (DynProc)
262 // =====
263 // Literature:
264 // Udo Zlzer, Digitale Audiosignal Verarbeitung, 3.
    Auflage
265 // Udo Zlzer, DAFX - Digital Audio Effects
266
267
268 // short:
269 //                  AT - attack time
270 //                  RT - release time
271 //                  TAV - average time
272 //                  LT- limiter threshold
273 //                  LS - limiter slope
274
275 // it works in that way
276 // Y(n) = G(n) * x(n -D)
277 // g = {noisegate, expander, compressor, limiter}
278 // D - delay m - samples
279
280
281 // helper
282
283 float DynProc_calcSlope (
284     float x1db, float y1db,
285     float x2db, float y2db
286 ) {
287     // y = m * x + n; P1 & P2
288     // m = (P2y - P1y) / (P2x - P1x)
289     return (y2db - y1db) / (x2db - x1db);
290 }
291
292 float DynProc_calcThreshold (
293     float xDB,
294     float xMax
295 ) {
296     return xMax * powf (10.0f, xDB / 20.0f);
297 }
298
299 float DynProc_calcDB (float x) {
300     return 10.0f * logf(x*x); // = 10.0 * log(x) =
        20.0f * log(x)
301 }

```

```

302
303 float DynProc_calcValueFromDB (float dbVlaue, float xMax
    ) {
304     return powf(10.0f, dbVlaue / 20.0) * xMax;
305 }
306
307
308 float DynProc_calcCompressionFactor (float slope) {
309     return 1.f / (1.f- slope);
310 }
311
312 ///! calc time parameter (AT, RT, TAV)
313 float DynProc_calcTimeParameter (
314     float Ta, ///!< (in) sampling
315     float t ///!< (in) time
316     parameter
317     ) {
318     // formular and 2.2 explained at "Digitale Audio
319     Signalverarbeitung" @p.237
320     return (1.0f - expf((-2.2f * Ta) / t));
321 }
322
323 float DynProc_calcTimeFromTimeparameter (
324     float Ta,
325     float Tval
326     ) {
327     return (-2.2f * Ta) / (logf(1-Tval));
328 }
329
330 // -----
331 // RMS
332 // -----
333 // RMS calc after Zlzer @p. 238
334
335 ///! init rms
336 void DynProc_InitRMS (
337     TDynProc_RMS * pRMS, ///!< (in/out) structure
338     to be filled
339     float TAV ///!< (in)
340     ) averaging time coefficient
341     ) {
342     pRMS->TAV = TAV;
343     pRMS->oneMinusTAV = 1.0f - TAV;
344     pRMS->x20ld = 0.0f;
345 }
346
347 ///! calc rms for one step
348 float DynProc_calcRMS (
349     TDynProc_RMS * pRMS, ///!< (in)
350     structure to be filled
351     float x
352     ///!< (in) input signal witch should
353     be rms
354     ) {

```

```

348     float x2RMS;
349     // calc forward
350     x2RMS = pRMS->oneMinusTAV * pRMS->x2Old + pRMS->
        TAV * x * x;
351     // calc backward
352     pRMS->x2Old = x2RMS;
353     return x2RMS;
354 }
355
356 // -----
357 // Peak
358 // -----
359 // Peak calc after Zlzer
360
361 //! init peak
362 void DynProc_InitPeak (
363     TDynProc_Peak * pPeak,    //!< (in/out) structure
        to be filled
364     float          AT,        //!< (in
        ) attack time
365     float          RT        //!< (in
        ) release time
366 ) {
367     pPeak->AT = AT;
368     pPeak->RT = RT;
369     pPeak->oneMinusAT = 1.0f - AT;
370     pPeak->oneMinusRT = 1.0f - RT;
371     pPeak->peak = 0.0f;
372 }
373
374 //! calc peak for one step
375 float DynProc_calcPeak (
376     TDynProc_Peak * pPeak,    //!< (in)
        structure to be filled
377     float          x          //!< (in) input signal witch should
        be rms
378 ) {
379
380     // formula from Zlzer @p. 235 wrong!
381     // |x(n)| > xPeak(n-1) -> attack
382     // |x(n)| <= xPeak(n-1) -> release
383     // @attack: xpeak(n) = (1-AT) * xpeak(n-1) + AT
        * |x(n)|
384     // @release: xpeak(n) = (1-RT) * peak(n-1)
385     float a;
386     a = fabsf(x);
387
388     if (a > pPeak->peak) {
389         pPeak->peak = pPeak->oneMinusAT * (pPeak
            ->peak) + pPeak->AT * a;
390     } else {
391         pPeak->peak = pPeak->oneMinusRT * (pPeak
            ->peak) + pPeak->RT * a;
392     }

```

```

393         return pPeak->peak;
394     }
395
396     // -----
397     // smooth gain
398     // -----
399     // smooth the gain value
400     // idea: use a hysteresis curve
401     //
402     // formula:  $g(n) = (1-k) * g(n-1) + k * f(n)$ 
403     //
404     // k = AT or k = RT
405     // k = (f(n) > f(n-1)) ? AT : RT
406     // if new value is over the old value -> attack
407     // else -> release
408
409     //! init AT/RT
410     void DynProc_InitSmoothG (
411         TDynProc_SmoothG *    pSG,        //!< (in/out)
412         structure to be filled
413         float                  AT,
414         //!< (in) attack time
415         float                  RT
416         //!< (in) release time
417     ) {
418         pSG->AT = AT;
419         pSG->RT = RT;
420         pSG->fOld = 0.0f;
421         pSG->gOld = 0.0f;
422     }
423
424     //! init AT/RT
425     float DynProc_calcSmoothG (
426         float                  fn,
427         //!< (in) input
428         TDynProc_SmoothG *    pSG        //!< (in)
429         structure to be filled
430     ) {
431         float k;
432         float gn;
433
434         // attack or release
435         k = (fn > pSG->fOld) ? pSG->AT : pSG->RT;
436
437         // calc gain
438         gn = (1.0f-k) * pSG->gOld + k * fn;
439         pSG->gOld = gn;
440         pSG->fOld = fn;
441         return gn;
442     }
443
444     // -----
445     // limiter
446     // -----
447     // LT - limiter treshold (where the limiter starts

```

```

    working)
443 // LS - limiter slope (how fast the limiter works)
444
445 //! init AT/RT
446 void DynProc_InitLimiter (
447     TDynProc_Limiter *    pLim,          //!< (in
        /out) structure to be filled
448     float                peakAT,
        //!< (in) peak attack time
449     float                peakRT,
        //!< (in) peak release time
450     float                smoothAT,
        //!< (in) smoothing attack time
451     float                smoothRT,
        //!< (in) smoothing release time
452     float                LT,
        //!< (in) limiter threshold
453     float                LS
        //!< (in) limiter slope
454 ) {
455     DynProc_InitPeak(&(pLim->peakDetector),peakAT,
        peakRT);
456     DynProc_InitSmoothG(&(pLim->SmoothG),smoothAT,
        smoothRT);
457     pLim->LTlog = log10f(LT);
458     pLim->LS = LS;
459     pLim->delay = 0.0f;
460 }
461
462 //! init AT/RT
463 float DynProc_calcLimiter (
464     TDynProc_Limiter *    pLim,          //!< (in
        ) the limiter
465     float                x
        //!< (in) input signal
466 ) {
467     float y;
468     float g;
469
470     // 1.) calc direct feedthrough lane
471     y = pLim->delay;
472     // 1.1.) recalc delay
473     pLim->delay = x;
474
475     // 2.) calc limiter gain
476     // 2.1) peak detect
477     g = DynProc_calcPeak(&(pLim->peakDetector),x);
478     g = log10f(g);
479     // 2.3. if we are over the threshold
480     if (g > pLim->LTlog) {
481         g = -pLim->LS * (g - pLim->LTlog);
482     } else {
483         g = 0.0;
484     }
485     g = powf (10.0f, g);

```

```

486     g = DynProc_calcSmoothG(g, &(pLim->SmoothG));
487     // 3. calc result of delay mul compressor gain
488     y *= g;
489
490     // AND return
491     return y;
492 }
493
494 // -----
495 // compressor
496 // -----
497 // CT - compressor treshold (where the compressor starts
498 //      working)
499 // CS - compressor slope (how fast the compressor works)
500
501 //! init compressor
502 void DynProc_InitCompressor (
503     TDynProc_Compressor *    pComp,          //!< (in
504         /out) structure to be filled
505     float                    rmsTAV,
506         //!< (in) rms time average
507         coefficient
508     float                    smoothAT
509     ,                        //!< (in) smoothing attack time
510     float                    smoothRT
511     ,                        //!< (in) smoothing release time
512     float                    CT,
513         //!< (in) compressor treshold
514     float                    CS
515     ,                        //!< (in) compressor slope
516 ) {
517     DynProc_InitRMS(&(pComp->RMS), rmsTAV);
518     DynProc_InitSmoothG(&(pComp->SmoothG), smoothAT,
519         smoothRT);
520     pComp->CTlog = DynProc_calcDB(CT);
521     pComp->CSlog = -CS;
522     pComp->delay[0] = 0.0f;
523     pComp->delay[1] = 0.0f;
524 }
525
526 //! calc compressor
527 float DynProc_calcCompressor (
528     TDynProc_Compressor *    pComp,          //!< (in
529         ) the compressor
530     float                    x
531     ,                        //!< (in) input
532     signal
533 ) {
534     float y;
535     float g;
536
537     // 1.) calc direct feedthrough lane
538     y = pComp->delay[1];
539     // 1.1.) recalc delay
540     pComp->delay[1] = pComp->delay[0];

```

```

529     pComp->delay[0] = x;
530
531     // 2.) calc compressor gain
532     // 2.1) peak detect
533     g = DynProc_calcRMS(&(pComp->RMS),x);
534     // 2.2) lin/log
535     g = log10f(g);
536     g *= 10.0f; // rms = x -> log(x) = 2 log(x) ->
537                 Xdb = 20*log(x) -> 20 / 2 = 10
538
539     // 2.3. if we are over the threshold
540     if (g > pComp->CTlog) {
541         g = pComp->CSlog*(g-pComp->CTlog) / 20.0
542         f;
543     } else {
544         g = 0.0f;
545     }
546     g = powf (10.0f, g);
547     g = DynProc_calcSmoothG(g, &(pComp->SmoothG));
548     // 3. calc result of delay mul compressor gain
549     y *= g;
550
551     // AND return
552     return y;
553 }
554
555 // -----
556 // expander
557 // -----
558 // ET - expander threshold (where the expander starts
559 // working)
560 // ES - expander slope (how fast the expander works)
561
562 //! init expander
563 void DynProc_InitExpander (
564     TDynProc_Expander * pExp,          //!< (in
565     /out) structure to be filled
566     float rmsTAV,                      //!< (in) rms time average coefficient
567     float smoothAT,                    //!< (in) smoothing attack time
568     float smoothRT,                    //!< (in) smoothing release time
569     float ET,                          //!< (in) expander treshold in dB
570     float ES,                          //!< (in) expander slope in dB
571 ) {
572     DynProc_InitRMS(&(pExp->RMS),rmsTAV);
573     DynProc_InitSmoothG(&(pExp->SmoothG),smoothAT,
574         smoothRT);
575     pExp->ETlog = DynProc_calcDB(ET);
576     pExp->ESlog = ES;
577     pExp->delay[0] = 0.0f;

```

```

574         pExp->delay[1] = 0.0f;
575     }
576
577     //! calc expander
578     float DynProc_calcExpander (
579         TDynProc_Expander *          pExp,
580         //!< (in) the expander
581         float
582         x                          //!< (in) input
583         signal
584     ) {
585         float y;
586         float g;
587
588         // 1.) calc direct feedthrough lane
589         y = pExp->delay[1];
590         // 1.1.) recalc delay
591         pExp->delay[1] = pExp->delay[0];
592         pExp->delay[0] = x;
593
594         // 2.) calc compressor gain
595         // 2.1) peak detect
596         g = DynProc_calcRMS(&(pExp->RMS), x);
597         // 2.2) lin/log
598         g = log10f(g);
599         g *= 10.0f; // rms = x -> log(x) = 2 log(x) ->
600         Xdb = 20*log(x) -> 20 / 2 = 10
601
602         // 2.3. if we are over the threshold
603         if (g > pExp->ETlog) {
604             g = pExp->ESlog*(pExp->ETlog-g) / 20.0;
605         } else {
606             g = 0.0f;
607         }
608         g = powf (10.0f, g);
609         g = DynProc_calcSmoothG(g, &(pExp->SmoothG));
610         // 3. calc result of delay mul compressor gain
611         y *= g;
612
613         // AND return
614         return y;
615     }
616
617     // -----
618     // noisegate
619     // -----
620     // NT - noisegate threshold (till the noisegate works)
621
622     //! init noisegate
623     void DynProc_InitNoisegate (
624         TDynProc_Noisegate *      png,          //!< (in
625         /out) structure to be filled
626         float                      rmsTAV,
627         //!< (in) rms time average

```



```

        coefficient
623     float          smoothAT
        ,            //!< (in) smoothing attack time
624     float          smoothRT
        ,            //!< (in) smoothing release time
625     float          NT,
        ,            //!< (in) noisegate threshold
626     float          NS
        ,            //!< (in) noisegate slope
627 ) {
628     DynProc_InitRMS(&(pNG->RMS),rmsTAV);
629     DynProc_InitSmoothG(&(pNG->SmoothG),smoothAT,
        smoothRT);
630     pNG->NTlog = DynProc_calcDB(NT);
631     pNG->NSlog = NS;
632     pNG->delay[0] = 0.0f;
633     pNG->delay[1] = 0.0f;
634 }
635
636 //!< calc noisegate
637 float DynProc_calcNoisegate (
638     TDynProc_Noisegate *    pNG,    //!< (in)
        the noisegate
639     float
        x                //!< (in) input
        signal
640 ) {
641     float y;
642     float g;
643
644     // 1.) calc direct feedthrough lane
645     y = pNG->delay[1];
646     // 1.1.) recalc delay
647     pNG->delay[1] = pNG->delay[0];
648     pNG->delay[0] = x;
649
650     // 2.) calc noisegate gain
651     // 2.1) rms detect
652     g = DynProc_calcRMS(&(pNG->RMS), x);
653     // 2.2) lin/log
654     g = log10f(g);
655     g *= 10.0f; // rms = x -> log(x) = 2 log(x) ->
        Xdb = 20*log(x) -> 20 / 2 = 10
656
657     // 2.3. if we are over the threshold
658     if (g > pNG->NTlog) {
659         // calc linear function at log space
660         g = pNG->NSlog*(g-pNG->NTlog) / 20.0;
661         g = powf(10.f, g);
662     } else {
663         g = 0.0f;
664     }
665     // and smooth
666     g = DynProc_calcSmoothG(g, &(pNG->SmoothG));
667     // 3. calc result of delay mul noisegate gain

```

```

668         y *= g;
669
670         // AND return
671         return y;
672     }

```

3.1.6 biquad filters (generic)

includes:

c-Include	c-Library	system lib
math.h	m	yes
stdlib.h		yes

code:

```

1  // =====
2  // a generic biquad filter lib based on the math.h
3  // part I - defines
4  // =====
5
6  // no print functions (they are only for debug purposes)
7  // #define dBQF_implementPrintFunctions
8
9  // the biquad structure at the direct form II
10 typedef struct SBQF_BiquadDF2 {
11     float    k;        // gain
12     float    n1;        // numerator
13     float    n2;
14     float    d1; // denominator
15     float    d2;
16     float    s1; // delays
17     float    s2;
18 } TBQF_BiquadDF2;
19
20 // a cascade of biquad filters
21 typedef struct SBQF_BiquadCascade {
22     int
23     TBQF_BiquadDF2 *      pB;
24 } TBQF_BiquadCascade;
25
26 //! creates a cascade of biquad filters
27 int BQF_BQFcascadeCreate (
28     TBQF_BiquadCascade *      pBQC,    //!< (in
29                                     //!< /out) pointer to a struct to be
29                                     //!< filled
30     int
31     num                        //!< amount of
32     filters
33 );
34
35 //! deletes a cascade of biquad filters

```

```

33 void BQF_BQFcascadeDelete (
34     TBQF_BiquadCascade *    pBQC    //!< (in
        ) pointer to a struct to be freed
35 );
36
37 //!< convolut the input with the biquad filter cascade
    and generate the output (y = BQFc * x)
38 void BQF_BQFcascadeConvolute (
39     TBQF_BiquadCascade *    pBQC,    //!< (in
        ) the biquad cascade
40     float *
        px,                //!< (in) input
        signal vector
41     unsigned int            xNum,
        //!< (in) amount of elements at the
        input vector
42     float *
        py                //!< (out) output
        signal vector (size must be more or
        equal to px)
43 );
44
45 //!< Initializes a element of the biquad cascade as high-
    pass(HP) filter
46 int BQF_BQFcascadeInitHP (
47     TBQF_BiquadCascade *    pBQC,    //!< (in
        /out) the biquad cascade
48     unsigned int            index,
        //!< (in) the index of the filter
        witch should be used as HP
49     float
        fs,                //!< (in) sample
        frequency
50     float
        fc                //!< (in) cut off
        frequency
51 );
52
53 //!< initializes a element of the biquad cascade as low-
    pass(LP) filter
54 int BQF_BQFcascadeInitLP (
55     TBQF_BiquadCascade *    pBQC,    //!< (in
        /out) the biquad cascade
56     unsigned int            index,
        //!< (in) the index of the filter
        witch should be used as LP
57     float
        fs,                //!< (in) sample
        frequency
58     float
        fc                //!< (in) cut off
        frequency
59 );
60
61 //!< initializes a element of the biquad cascade as peak

```

```

boost/cut filter
62 int BQF_BQFcascadeInitPeak (
63     TBQF_BiquadCascade *    pBQC,    //!< (in
        /out) the biquad cascade
64     unsigned int            index,
        //!< (in) the index of the filter
        witch should be used as peak filter
65     float
        fs,                    //!< (in) sample
        frequency
66     float
        fc,                    //!< (in) center
        frequency
67     float
        q,                      //!< (in) quality (Q
        infinity)
68     float
        g                        //!< (in) gain
69 );
70
71 //!< initializes a element of the biquad cascade as low
    frequency shelving filter
72 int BQF_BQFcascadeInitLowFreqShelving (
73     TBQF_BiquadCascade *    pBQC,    //!< (in
        /out) the biquad cascade
74     unsigned int            index,
        //!< (in) the index of the filter
        witch should be used as peak filter
75     float
        fs,                    //!< (in) sample
        frequency
76     float
        f,                      //!< (in) cut/boost
        frequency
77     float
        q,                      //!< (in) quality (Q
        infinity)
78     float
        g                        //!< (in) gain
79 );
80
81 //!< initializes a element of the biquad cascade as high
    frequency shelving filter
82 int BQF_BQFcascadeInitHighFreqShelving (
83     TBQF_BiquadCascade *    pBQC,    //!< (in
        /out) the biquad cascade
84     unsigned int            index,
        //!< (in) the index of the filter
        witch should be used as peak filter
85     float
        fs,                    //!< (in) sample
        frequency
86     float
        f,                      //!< (in) cut/boost
        frequency

```

```

87         float
            q,                //!< (in) quality (Q
            infinty)
88         float
            g                //!< (in) gain
89     );
90
91     //!< initializes a element of the biquad cascade from a
92     array with the coefficients
93     void BQF_BQFinitFromCoefficients (
94         TBQF_BiquadCascade *    pBQC,
95         //!< (in/out) the biquad
96         cascade
97         int
98             index,                //!<
99         (in) the index of the filter witch
100         should be used as peak filter
101         float
102             numerator [3],    //!< (in) n0 to n2
103         float
104             denominator [3]  //!< (in) d0 to d2
105     );
106
107     //!< gets from an element of the biquad cascade the
108     coefficients
109     void BQF_BQFgetNumAndDenom (
110         TBQF_BiquadCascade *    pBQC,
111         //!< (in/out) the biquad
112         cascade
113         int
114             index,                //!<
115         (in) the index of the filter witch
116         should be used as peak filter
117         float *
118             pNumerator,                //!< (in) n0
119         to n2
120         float *
121             pDenominator    //!< (in) d0 to d2
122     );
123
124     #ifndef dBQF_implementPrintFunctions
125     // a simple print function (k, n1, n2, d1, d2)
126     void BQF_printBiquadDF2(TBQF_BiquadCascade *pBC, FILE *
127         stream);
128
129     // a simple print function (n0, n1, n2, d0, d1, d2)
130     void BQF_PrintBiquad(TBQF_BiquadCascade *pBC, FILE *
131         stream);
132     #endif
133
134     // =====
135     // a generic biquad filter lib based on the math.h
136     // part II - implementations
137     // =====
138
139

```

```

120 // math defines
121 #ifndef M_E
122     #define M_E 2.7182818284590452354
123 #endif
124
125 #ifndef M_LOG2E
126     #define M_LOG2E 1.4426950408889634074
127 #endif
128
129 #ifndef M_LOG10E
130     #define M_LOG10E 0.43429448190325182765
131 #endif
132
133 #ifndef M_LN2
134     #define M_LN2 0.69314718055994530942
135 #endif
136
137 #ifndef M_LN10
138     #define M_LN10 2.30258509299404568402
139 #endif
140
141 #ifndef M_PI
142     #define M_PI 3.14159265358979323846
143 #endif
144
145 #ifndef M_PI_2
146     #define M_PI_2 1.57079632679489661923
147 #endif
148
149 #ifndef M_PI_4
150     #define M_PI_4 0.78539816339744830962
151 #endif
152
153 #ifndef M_1_PI
154     #define M_1_PI 0.31830988618379067154
155 #endif
156
157 #ifndef M_2_PI
158     #define M_2_PI 0.63661977236758134308
159 #endif
160
161 #ifndef M_2_SQRTPI
162     #define M_2_SQRTPI 1.12837916709551257390
163 #endif
164
165 #ifndef M_SQRT2
166     #define M_SQRT2 1.41421356237309504880
167 #endif
168
169 #ifndef M_SQRT1_2
170     #define M_SQRT1_2 0.70710678118654752440
171 #endif
172
173
174

```

```

175  //! creates a cascade of biquad filters
176  int BQF_BQFcascadeCreate (
177      TBQF_BiquadCascade *    pBQC,    //!< (in
        /out) pointer to a struct to be
        filled
178      int
        num                //!< amount of
        filters
179  ) {
180      int i;
181
182      // alloc / realloc the biquad
183      if (pBQC->num != num) {
184          if (pBQC->pB) {
185              free(pBQC->pB);
186          }
187          pBQC->pB = malloc (sizeof(TBQF_BiquadDF2
        ) * num);
188          if (!pBQC->pB) {
189              pBQC->num = 0;
190              return -1;
191          }
192          pBQC->num = num;
193      }
194
195      // init data fields
196      for (i = 0; i < num; i++) {
197          pBQC->pB[i].k = 0.0f;
198          pBQC->pB[i].n1 = 0.0f;
199          pBQC->pB[i].n2 = 0.0f;
200          pBQC->pB[i].d1 = 0.0f;
201          pBQC->pB[i].d2 = 0.0f;
202          pBQC->pB[i].s1 = 0.0f;
203          pBQC->pB[i].s2 = 0.0f;
204      }
205      return 0;
206  }
207
208  //! deletes a cascade of biquad filters
209  void BQF_BQFcascadeDelete (
210      TBQF_BiquadCascade *    pBQC    //!< (in
        ) pointer to a struct to be freed
211  ) {
212      free (pBQC->pB);
213      pBQC->pB = NULL;
214      pBQC->num = 0;
215  }
216
217  //! convolut the input with the biquad filter cascade
        and generate the output (y = BQFc * x)
218  void BQF_BQFcascadeConvolute (
219      TBQF_BiquadCascade *    pBQC,    //!< (in
        ) the biqaud cascade
220      float *
        px,                //!< (in) input

```

```

221         signal vector
        unsigned int                xNum,
        //!< (in) amount of elements at the
        input vector
222     float *
        py                          //!< (out) output
        signal vector (size must be more or
        equal to px)

223     ) {
224     int                i, j;
225     float              d1, y;
226     TBQF_BiquadDF2 *   pF;

227
228     for (i = 0; i < xNum; i++) {
229         y = *px;
230         pF = pBQC->pB;
231         for (j = 0; j < pBQC->num; j++) {
232             // filter
233             d1 = - (pF->d2 * pF->s2 + pF->d1
                * pF->s1 + y); // y = x
234             y = pF->n2 * pF->s2 + pF->n1 *
                pF->s1 + d1;
235             y *= pF->k;
236
237             pF->s2 = pF->s1;
238             pF->s1 = d1;
239             // output is input for the next
                filter
240             pF++;
241         }
242
243         // save result to the output
244         *py = y;
245         px++;
246         py++;
247     }
248 }
249
250 //! initializes a element of the biquad cascade as high-
    pass(HP) filter
251 int BQF_BQFcascadeInitHP (
252     TBQF_BiquadCascade *   pBQC,    //!< (in
        /out) the biquad cascade
253     unsigned int            index,
        //!< (in) the index of the filter
        witch should be used as HP
254     float                    fs,
        //!< (in) sample
        frequency
255     float                    fc
        //!< (in) cut off
        frequency
256 ) {
257     if (index >= pBQC->num) {
258         return -1;

```



```

259     }
260     float k=tan(M_PI*fc/fs);
261     float k2 = k * k;
262     float sqrtTwo = M_SQRT2;
263     float dn = (1+sqrtTwo*k+k2);
264
265     float numerator[3];
266     float denominator[3];
267
268     numerator[0]=1/dn;
269     numerator[1]=-2/dn;
270     numerator[2]=1/dn;
271     denominator[0]=1;
272     denominator[1]=(2*(k2-1))/dn;
273     denominator[2]=(1-sqrtTwo*k+k2)/dn;
274
275     BQF_BQFinitFromCoefficients(pBQC,index, numerator
276                                , denominator);
277
278     return 0;
279 }
280
281 //! initializes a element of the biquad cascade as low-
282 //! pass(LP) filter
283 int BQF_BQFcascadeInitLP (
284     TBQF_BiquadCascade * pBQC,    //!< (in
285                                     /out) the biquad cascade
286     unsigned int index,           //!< (in) the index of the filter
287                                     witch should be used as LP
288     float fs,                    //!< (in) sample
289                                     frequency
290     float fc                      //!< (in) cut off
291                                     frequency
292 ) {
293     if (index >= pBQC->num) {
294         return -1;
295     }
296     float k=tan(M_PI*fc/fs);
297     float k2 = k * k;
298     float sqrtTwo = M_SQRT2;
299     float dn = (1+sqrtTwo*k+k2);
300
301     float numerator[3];
302     float denominator[3];
303
304     numerator[0]=k2/dn;
305     numerator[1]=2*k2/dn;
306     numerator[2]=k2/dn;
307     denominator[0]=1;
308     denominator[1]=(2*(k2-1))/dn;
309     denominator[2]=(1-sqrtTwo*k+k2)/dn;

```

```

305         BQF_BQFinitFromCoefficients(pBQC, index, numerator
306         , denominator);
307         return 0;
308     }
309     /*! initializes a element of the biquad cascade as peak
310     boost/cut filter
311     int BQF_BQFcascadeInitPeak (
312         TBQF_BiquadCascade *    pBQC,    /*!< (in
313         /out) the biquad cascade
314         unsigned int             index,
315         /*!< (in) the index of the filter
316         witch should be used as peak filter
317         float
318         fs,                      /*!< (in) sample
319         frequency
320         float
321         fc,                      /*!< (in) center
322         frequency
323         float
324         q,                      /*!< (in) quality (Q
325         infinty)
326         float
327         g                        /*!< (in) gain
328     ) {
329     // some info:
330     // @zlzer: g is in DB and is transformed via  $V0 = 10^{(G/20)}$  into v0 (DAFX p.55)
331     // @my function:
332     //     1. g is not in dB! g = V0
333     //     2. g < 1 => peak boost otherwise peak
334     cut
335
336     if (index >= pBQC->num) {
337         return -1;
338     }
339     if (q == 0.0f) {
340         return -2;
341     }
342     float k=tan(M_PI*fc/fs);
343     float k2 = k * k;
344     float dn;
345
346     float numerator[3];
347     float denominator[3];
348
349     if (g < 1) {
350         // boost
351         dn = 1.f+1.f/q*k+k2;
352         numerator[0] = (1.f+g/q*k+k2) / dn;
353         numerator[2] = (1.f-g/q*k+k2) / dn;
354         denominator[2] = (1.0f-1.0f/q*k+k2) / dn
355         ;
356     } else {
357         // cut

```

```

345         dn = (1.f+g/q*k+k2);
346         numerator[0] = (1.f+1.f/q*k+k2) / dn;
347         numerator[2] = (1.f-1.f/q*k+k2) / dn;
348         denominator[2] = (1.0f-g/q*k+k2) / dn;
349
350     }
351     numerator[1]=2.0f*(k2-1.0f)/dn;
352     denominator[0]=1.f;
353     denominator[1]=(2.f*(k2-1.f))/dn;
354
355     BQF_BQFinitFromCoefficients(pBQC,index, numerator
356     , denominator);
357     return 0;
358 }
359
360 //! initializes a element of the biquad cascade as low
361 // frequency shelving filter
362 int BQF_BQFcascadeInitLowFreqShelving (
363     TBQF_BiquadCascade * pBQC,    //!< (in
364     /out) the biquad cascade
365     unsigned int index,
366     //!< (in) the index of the filter
367     witch should be used as peak filter
368     float fs,
369     //!< (in) sample
370     frequency
371     float f,
372     //!< (in) cut/boost
373     frequency
374     float q,
375     //!< (in) quality (Q
376     infinty)
377     float g,
378     //!< (in) gain
379     ) {
380     // some info:
381     // @zlzer: g is in DB and is transformed via  $V0 = 10^{(G/20)}$  into v0 (DAFX p.55)
382     // @my function:
383     // 1. g is not in dB! g = V0
384     // 2. g < 1 => peak boost otherwise peak
385     cut
386
387     if (index >= pBQC->num) {
388         return -1;
389     }
390     if (q == 0.0f) {
391         return -2;
392     }
393     float k=tan(M_PI*f/fs);
394     float k2 = k * k;
395     float sqrtTwo = M_SQRT2;
396     float sqrt2V0 = sqrtf(2.f * g);
397
398     float dn;

```

```

386
387     float numerator[3];
388     float denominator[3];
389
390     if (g < 1) {
391         // boost
392         dn = 1.f+sqrtTwo*k+k2;
393         numerator[0] = (1.f+sqrt2V0*k+g*k2) / dn
394         ;
395         numerator[1] = (2.f * (g *k2 -1.f)) / dn
396         ;
397         numerator[2] = (1.f-sqrt2V0*k+g*k2) / dn
398         ;
399         denominator[1] = (2.f * (k2 -1.f)) / dn;
400         denominator[2] = (1.f-sqrtTwo*k+k2) / dn
401         ;
402     } else {
403         // cut
404         dn = 1.f+sqrt2V0*k+g*k2;
405         numerator[0] = (1.f+sqrtTwo*k+k2) / dn;
406         numerator[1] = (2.f * (k2 -1.f)) / dn;
407         numerator[2] = (1.f-sqrtTwo*k+k2) / dn;
408         denominator[1] = (2.f * (g*k2 -1.f)) /
409         dn;
410         denominator[2] = (1.f-sqrt2V0*k+g*k2) /
411         dn;
412     }
413     denominator[0]=1.f;
414
415     BQF_BQFinitFromCoefficients(pBQC,index, numerator
416     , denominator);
417     return 0;
418 }
419
420 //!! initializes a element of the biquad cascade as high
421 // frequency shelving filter
422 int BQF_BQFcascadeInitHighFreqShelving (
423     TBQF_BiquadCascade * pBQC,    //!!< (in
424     /out) the biquad cascade
425     unsigned int index,
426     //!!< (in) the index of the filter
427     witch should be used as peak filter
428     float
429     fs,                //!!< (in) sample
430     frequency
431     float
432     f,                //!!< (in) cut/boost
433     frequency
434     float
435     q,                //!!< (in) quality (Q
436     infinty)
437     float
438     g                //!!< (in) gain
439     ) {
440     // some info:

```

```

423 // @zlzer: g is in DB and is transformed via  $V0 = 10^{(G/20)}$  into v0 (DAFX p.55)
424 // @my function:
425 //           1. g is not in dB! g = V0
426 //           2. g < 1 => peak boost otherwise peak
    cut
427
428     if (index >= pBQC->num) {
429         return -1;
430     }
431     if (q == 0.0f) {
432         return -2;
433     }
434     float k=tan(M_PI*f/fs);
435     float k2 = k * k;
436     float sqrtTwo = M_SQRT2;
437     float sqrt2V0 = sqrtf(2.f * g);
438     float sqrt2divV0 = sqrtf(2.f / g);
439
440     float dn;
441     float dn2;
442
443     float numerator[3];
444     float denominator[3];
445
446     if (g < 1) {
447         // boost
448         dn = 1.f+sqrtTwo*k+k2;
449         numerator[0] = (g+sqrt2V0*k+k2) / dn;
450         numerator[1] = (2.f * (k2 -g)) / dn;
451         numerator[2] = (g-sqrt2V0*k+k2) / dn;
452         denominator[1] = (2.f * (k2 -1.f)) / dn;
453         denominator[2] = (1.f-sqrtTwo*k+k2) / dn
454         ;
455     } else {
456         // cut
457         dn = g+sqrt2V0*k+k2;
458         dn2 = 1 + sqrt2divV0 * k + k2/g;
459         numerator[0] = (1.f+sqrtTwo*k+k2) / dn;
460         numerator[1] = (2.f * (k2 -1.f)) / dn;
461         numerator[2] = (1.f-sqrtTwo*k+k2) / dn;
462         denominator[1] = (2.f * (k2/g -1.f)) /
            dn2;
463         denominator[2] = (1.f-sqrt2divV0*k+k2/g)
            / dn2;
464     }
465     denominator[0]=1.f;
466     BQF_BQFinitFromCoefficients(pBQC,index, numerator
467     , denominator);
468     return 0;
469 }
470
471 //! initializes a element of the biquad cascade from a

```

```

472 // array with the coefficients
473 // input
474 // H(z) = (n0*z0 + n1*z-1 + n2 * z-2) / (d0*z0 + d1*z-1
475 // + d2 * z-2)
476 // output
477 // 1/n0 * (n0/n0*z0 + n1/n0*z-1 + n2/n0 * z-2)
478 // H(z) =-----
479 // 1/d0 * (d0/d0*z0 + d1/d0*z-1 + d2/d0 * z-2)
480 // with now
481 // k = d0/n0
482 // n0 = 1
483 // n1 = n1/n0
484 // n2 = n2/n0
485 // d0 = 1
486 // d1 = d1/d0
487 // d2 = d2/d0
488 void BQF_BQFinitFromCoefficients (
489     TBQF_BiquadCascade * pBQC,
490     //!< (in/out) the biquad
491     cascade
492     int
493     index, //!<
494     (in) the index of the filter witch
495     should be used as peak filter
496     float
497     numerator [3], //!< (in) n0 to n2
498     float
499     denominator [3] //!< (in) d0 to d2
500 ) {
501     TBQF_BiquadDF2 * pBQ = pBQC->pB + index;
502     pBQ->n1 = numerator[1] / numerator[0];
503     pBQ->n2 = numerator[2] / numerator[0];
504     pBQ->d1 = denominator[1] / denominator[0];
505     pBQ->d2 = denominator[2] / denominator[0];
506     pBQ->k = denominator[0] / numerator[0];
507     pBQ->s1 = 0.f;
508     pBQ->s2 = 0.f;
509 }
510
511 //! gets from an element of the biquad cascade the
512 // coefficients
513 // H1 is the internal form
514 // H1(z) = k (1*z0 + n1*z-1 + n2 * z-2) / (1*z0 + d1*z-1
515 // + d2 * z-2)
516 // H2 is the external form
517 // H2(z) = (n0*z0 + n1*z-1 + n2 * z-2) / (d0*z0 + d1*z-1
518 // + d2 * z-2)
519 // Formula:
520 // n0 = k
521 // n1 = k*n1
522 // n2 = k*n2
523 // d0 = 1
524 // d1 = d1
525 // d2 = d2
526 void BQF_BQFgetNumAndDenom (

```

```

515         TBQF_BiquadCascade * pBQC,
           //!< (in/out) the biquad
           cascade
516         int
           index, //!<
           (in) the index of the filter witch
           should be used as peak filter
517         float *
           pNumerator, //!< (out) n0
           to n2
518         float *
           pDenominator //!< (out) d0 to d2
519     ) {
520         TBQF_BiquadDF2 * pBQ = pBQC->pB + index;
521         pNumerator[0] = pBQ->k;
522         pNumerator[1] = pBQ->k*pBQ->n1;
523         pNumerator[2] = pBQ->k*pBQ->n2;
524         pDenominator[0] = 1.0f;
525         pDenominator[1] = pBQ->d1;
526         pDenominator[2] = pBQ->d2;
527     }
528
529 #ifdef dBQF_implementPrintFunctions
530
531 // a simple print function (k, n1, n2, d1, d2)
532 void BQF_printBiquadDF2(TBQF_BiquadCascade *pBC, FILE *
    stream) {
533     int i;
534     TBQF_BiquadDF2 * pB;
535
536     fprintf(stream, "BQC:\n");
537     pB = pBC->pB;
538     for (i = 0; i < pBC->num; i++) {
539         fprintf(stream, "[%i]:\n", i);
540         fprintf(stream, "\tk = %f\n", pB->k);
541         fprintf(stream, "\tn1 = %f\n", pB->n1);
542         fprintf(stream, "\tn2 = %f\n", pB->n2);
543         fprintf(stream, "\td1 = %f\n", pB->d1);
544         fprintf(stream, "\td2 = %f\n", pB->d2);
545         pB++;
546     }
547     fflush(stream);
548 }
549
550 // a simple print function (n0, n1, n2, d0, d1, d2)
551 void BQF_PrintBiquad(TBQF_BiquadCascade *pBC, FILE *
    stream) {
552     int i, j;
553     float num[3];
554     float denom[3];
555
556     fprintf(stream, "BQC:\n");
557     for (i = 0; i < pBC->num; i++) {
558         BQF_BQFgetNumAndDenom(pBC, i, num, denom
            );

```

```

559         fprintf(stream,"[%i]:\n",i);
560         for (j = 0; j < 3; j++) {
561             fprintf(stream,"\tn%i = %f\n",j,
                    num[j]);
562         }
563         for (j = 0; j < 3; j++) {
564             fprintf(stream,"\td%i = %f\n",j,
                    denom[j]);
565         }
566     }
567     fflush(stream);
568 }
569 }
570
571 #endif

```

3.1.7 boost

code:

3.1.8 fftw3 & complex

includes:

c-Include	c-Library	system lib
fftw3.h	libfftw3f-3	no

code:

```

1 // =====
2 // fftw3 libis used for complex and fft & ifft functions
3 // =====

```

3.1.9 generic delay

includes:

c-Include	c-Library	system lib
stdlib.h		yes

code:

```

1 // =====

```



```

2 // generic delay implementation (start)
3 //
=====
4
5 //
=====

6 // a generic i/o optimized implementation of a delayline
7 //
8 // it relies on the idea of a ringbuffer
9 // it is implemented by the use of read/write pointers
10 // features:
11 // - full ANSI C89 compatible
12 // - it doesn't use memmove, memcpy, malloc, free etc
13 // - it minimizes the amount of local variables
14 // - it precalculates wrappings and avoids if clauses (
    instruction cache friendly)
15 // - it minimizes read/write operation on the same
    memory cells and only
16 // access the memory cells only once (data cache
    friendly)
17 //
=====

18
19 //! a delay
20 typedef struct SgenDelay {
21     float * pStart; //!< starting pointer
22     float * pEnd;    //!< end pointer
23     int      amount; //!< amount of elements
24     float * pR;      //!< read pointer
25     float * pW;      //!< write pointer
26 } TgenDelay;
27
28 //! a enum type for the read / write flags
29 typedef enum EgenDelayRWflag {
30     EgenDelayRWflag_readPointer = 0,
31         //!< only read pointer
32     EgenDelayRWflag_writePointer = 1,
33         //!< only write pointer
34     EgenDelayRWflag_readAndWritePointer = 2,
35         //!< read & write pointer
36 } TgenDelayRWflag;
37
38 //! inits the delay
39 void genDelay_init (
    TgenDelay *      pD,
        //!< (in/out) pointer
        to a delay struct which is filled
    float *          pStart,
        //!< (in) pointer to
        the memory used to store the values
    int              amount,
        //!< (in)

```

```

40         amount of elements at the memory
        int
        clearBufferFlag        //!< if not
                                zero the buffer get overwritten with
                                0.0f
41    ) {
42        // set pointers
43        pD->pStart = pStart;
44        pD->amount = amount;
45        pD->pEnd = pStart + amount;
46        pD->pW = pStart;
47        pD->pR = pStart;
48        if (clearBufferFlag) {
49            // and the buffer to 0.0f
50            float * pB = pStart;
51            while (amount) {
52                *pB = 0.0f;
53                pB++;
54                amount--;
55            }
56        }
57    }
58
59    //!< creates the structure and the buffer for a delay
60    TgenDelay * genDelay_create (int amountOfSamples) {
61        // malloc structure
62        TgenDelay * pD = malloc(sizeof(TgenDelay)*
        amountOfSamples);
63        if (!pD) return NULL;
64        if (amountOfSamples > 0) {
65            // malloc data buffer
66            pD->pStart = malloc(sizeof(float)*
        amountOfSamples);
67            if (!pD->pStart) {
68                free (pD);
69                return NULL;
70            }
71            // reset structure
72            genDelay_init(pD,pD->pStart,
        amountOfSamples, 1);
73        } else {
74            genDelay_init(pD,NULL,0, 0);
75        }
76        return pD;
77    }
78
79    //!< deletes the delay structure and the contained data
80    void genDelay_delete (TgenDelay * pD) {
81        if (!pD) return;
82        if (pD->pStart) free(pD->pStart);
83        free(pD);
84    }
85
86    //!<
87    int genDelay_resize (TgenDelay * pD, int newBufferSize)

```

```

88     {
89         // null pointer exception
90         if (!pD) return -1;
91         // have we really to alloc new memory?
92         if (pD->amount == newBufferSize) goto init;
93         // ok lets free the old one
94         free (pD->pStart);
95         // and alloc a new one
96         pD->pStart = malloc(sizeof(float)*newBufferSize)
97         ;
98         // check if it worked
99         if (!pD->pStart) {
100             // no - ok lets make stable state and
101             // report an error
102             genDelay_init(pD, NULL, 0, 0);
103             return -2;
104         }
105         // it worked lets init
106     }
107     init:
108         genDelay_init(pD, pD->pStart, newBufferSize, 1);
109         return 0;
110 }
111 // shuffle the read and/or write pointer
112 void genDelay_shuffle (
113     TgenDelay *      pD,
114     //!< (in/out) pointer to a delay
115     struct
116     TgenDelayRWflag what,          //!<
117     // what defines what pointer is affected
118     int              offset
119     //!< the offset of the R/W
120     pointer
121 ) {
122     int modf = offset % pD->amount;
123     // read pointer
124     if ((what == EgenDelayRWflag_readPointer) || (
125         what == EgenDelayRWflag_readAndWritePointer))
126     {
127         pD->pR += modf;
128         if (pD->pR > pD->pEnd) {
129             pD->pR = pD->pStart + (pD->pR -
130                 pD->pEnd);
131         } else {
132             if (pD->pR < pD->pStart) {
133                 pD->pR = pD->pEnd - (pD
134                     ->pStart - pD->pR);
135             }
136         }
137     }
138     // write pointer
139     if ((what == EgenDelayRWflag_writePointer) || (
140         what == EgenDelayRWflag_readAndWritePointer))

```

```

130         {
131             pD->pW += modf;
132             if (pD->pW > pD->pEnd) {
133                 pD->pW = pD->pStart + (pD->pW -
134                                     pD->pEnd);
135             } else {
136                 if (pD->pW < pD->pStart) {
137                     pD->pW = pD->pEnd - (pD
138                                     ->pStart - pD->pW);
139                 }
140             }
141         }
142     }
143
144     ///! writes a grain of values to the buffer
145     void genDelay_write (
146         TgenDelay *      pD,          ///!< (in
147         /out) pointer to a delay struct
148         float *          pSrc,        ///!< (in
149         ) pointer to the values put into the
150         delay
151         int               N           ///!< (in) amount of values to
152         be stored
153     ) {
154         while (N) {
155             // copy value
156             *pD->pW = *pSrc;
157             // inc src pointer
158             pSrc++;
159             // inc write pointer
160             pD->pW++;
161             // does we extends the border?
162             if (pD->pW >= pD->pEnd) {
163                 // yes - ok lets wrap
164                 pD->pW = pD->pStart;
165             }
166             N--;
167         }
168     }
169
170     ///! reads a grain of values form the buffer
171     void genDelay_read (
172         TgenDelay *      pD,          ///!< (in
173         ) pointer to a delay struct
174         float *          pDest,       ///!< (in
175         /out) pointer to which the values are
176         stored
177         int               N           ///!< (in) amount of values to
178         be stored
179     ) {
180         while (N) {
181             // copy value

```

```

172         *pDest = *pD->pR;
173         // inc src pointer
174         pDest++;
175         // inc read pointer
176         pD->pR++;
177         // does we extends the border?
178         if (pD->pR >= pD->pEnd) {
179             // yes - ok lets wrap
180             pD->pR = pD->pStart;
181         }
182         N--;
183     }
184 }
185
186 //! reads a grain of values form the buffer and copies
187 //! the same amount into the buffer
188 void genDelay_readWrite (
189     TgenDelay *      pD,          //!< (in
190     // pointer to a delay struct
191     float *          pSrc,        //!< (in
192     // a pointer from which the values
193     // should be copied
194     float *          pDest,      //!< (in
195     // /out) pointer to which the values are
196     // stored
197     int              N            //!< (in) amount of values to
198     // be stored
199 ) {
200     // 3 state approach to minimize the amount of i/
201     // o operations
202     // 1st state read till we reach the write
203     // pointer or N is 0
204     // 2nd state read & write till we read all data
205     // needed
206     // 3th state write till all data is written
207
208     // wrap counter
209     int wc;
210
211     // read number
212     int rn, rn2;
213     // lets calculate how many read operations are
214     // needed to reach the write pointer
215     // first check the position of the write pointer
216     if (pD->pW >= pD->pR) {
217         // the write pointer is in front of the
218         // read pointer
219         rn = pD->pW - pD->pR;
220     } else {
221         // the write pointer stands after the
222         // read pointer
223         rn = pD->amount - (pD->pR - pD->pW);
224     }
225     // trim

```

```

213     if (rn > N) rn = N;
214     rn2 = N - rn;
215
216     // 1st state
217     // 1.1. read till it wraps
218     wc = pD->pEnd - pD->pR;
219     if (wc > rn) wc = rn;
220     rn -= wc;
221     while (wc) {
222         wc--;
223         *pDest = *pD->pR;
224         pDest++;
225         pD->pR++;
226     }
227     // wrap if needed
228     if (pD->pR >= pD->pEnd) pD->pR = pD->pStart;
229     // 1.2 and read on
230     while (rn) {
231         rn--;
232         *pDest = *pD->pR;
233         pDest++;
234         pD->pR++;
235     }
236
237     // 2nd state
238     // the read/write operation
239     // we use only the pD->pW pointer for the
240     // operation and increment the pD->pR only once
241     // in rn2 are the remaining numbers stored
242     pD->pR += rn2;
243     N -= rn2;
244     // wrap pR if needed
245     if (pD->pR >= pD->pEnd) pD->pR = pD->pStart + (
246         pD->pR - pD->pEnd);
247
248     // get the wrap counter
249     wc = pD->pEnd - pD->pW;
250     if (wc > rn2) wc = rn2;
251
252     // get the remaining number
253     rn = rn2 - wc;
254
255     // write & read
256     while (wc) {
257         wc--;
258         *pDest = *pD->pW;
259         *pD->pW = *pSrc;
260         pDest++;
261         pSrc++;
262         pD->pW++;
263     }
264     // warp
265     if (pD->pW >= pD->pEnd) pD->pW = pD->pStart;
266     // write & read
267     while (rn) {

```

```

266         rn--;
267         *pDest = *pD->pW;
268         *pD->pW = *pSrc;
269         pDest++;
270         pSrc++;
271         pD->pW++;
272     }
273
274     // 3th state
275     // write the data
276     // at N are the number of the remaining data
277
278     // get the wrap counter
279     wc = pD->pEnd - pD->pW;
280     if (wc > N) wc = N;
281     N -= wc;
282     // write the data
283     while (wc) {
284         wc--;
285         *pD->pW = *pSrc;
286         pSrc++;
287         pD->pW++;
288     }
289     // wrap
290     if (pD->pW >= pD->pEnd) pD->pW = pD->pStart;
291     // write the remaining data
292     while (N) {
293         N--;
294         *pD->pW = *pSrc;
295         pSrc++;
296         pD->pW++;
297     }
298 }
299
300 //
301 // =====
302 // generic delay implementation (end)
303 //
304 // =====

```

3.1.10 gtk+ for Windows

includes:

c-Include	c-Library	system lib
gtk/gtk.h	gtk-win32-2.0	yes
glib.h	glib-2.0	yes
gthread.h	gthread-2.0	yes
glib-object.h	gobject-2.0	yes
cairo.h	cairo	yes
pango/pango.h	pango-1.0	yes
semaphore.h	pthread	yes
math.h	m	yes
gdk/gdk.h	gdk-win32-2.0	yes
string.h		yes

code:

```
1 // -----
2 // AP gtk helper
3 // -----
4
5 PangoFontDescription * gAPgtkMonoSpaceFont = NULL;
6
7
8 // a helper for destroying "undestroyed" UI's
9 void APgtk_helper_destroy (GtkWidget * pWG) {
10     if (pWG) {
11         if (pWG->object.parent_instance.
12             ref_count) {
13             gtk_widget_destroy(pWG);
14         }
15     }
16 }
17 // -----
18 // AP panel
19 // -----
20
21 // struct for a panel
22 typedef struct SAPgtkPanel {
23     GtkWidget *      pWnd;           //!<
24         pointer to the window
25     GtkWidget *      pBox;           //!<
26         placement box
27 } TAPgtkPanel;
28
29 // create a panel
30 int gtkAP_local_PanelCreate (
31     TAPgtkPanel *      pP,           //!<(in/
```



```

        out) structure to be filled
31         int                x,
           window            //!<(in) x position of the
32         int                y,
           window            //!<(in) y position of the
33         int                xle,
           //!<(in) x length of the window
34         int                yle
           //!<(in) x length of the
           window
35     ) {
36         // Create a new window
37         pP->pWnd = gtk_window_new (GTK_WINDOW_TOPLEVEL);
38         if (!pP->pWnd) return -1;
39
40         // UI's are at fixed positions
41         pP->pBox = gtk_fixed_new();
42         if (!pP->pBox) return -2;
43
44         gtk_container_add(GTK_CONTAINER(pP->pWnd), pP->
           pBox);
45
46         // setup main wnd
47         gtk_window_set_title (GTK_WINDOW (pP->pWnd), "AP
           Panel");
48         gtk_window_set_default_size (GTK_WINDOW (pP->
           pWnd),xle, yle);
49         gtk_window_move(GTK_WINDOW (pP->pWnd),x, y);
50
51         // It's a good idea to do this for all windows
52         //gtk_signal_connect (GTK_OBJECT (pP->pWnd), "
           destroy", GTK_SIGNAL_FUNC (gtk_exit), NULL);
53         gtk_signal_connect (GTK_OBJECT (pP->pWnd), "
           destroy", GTK_SIGNAL_FUNC (gtk_widget_destroy
           ), NULL);
54
55         //gtk_signal_connect (GTK_OBJECT (pP->pWnd), "
           delete_event", GTK_SIGNAL_FUNC (gtk_exit),
           NULL);
56         gtk_signal_connect (GTK_OBJECT (pP->pWnd), "
           delete_event", GTK_SIGNAL_FUNC (
           gtk_widget_destroy), NULL);
57
58         // Sets the border width of the window
59         gtk_container_set_border_width (GTK_CONTAINER (
           pP->pWnd), 1);
60
61         gtk_widget_realize(pP->pWnd);
62
63         gtk_widget_show_all(pP->pWnd);
64
65         return(0);
66     }

```

```

67
68
69 // destroys a panel
70 void gtkAP_local_PanelDestroy (
71     TAPgtkPanel *    pP
72 ) {
73     APgtk_helper_destroy(pP->pWnd);
74     APgtk_helper_destroy(pP->pBox);
75     pP->pWnd = NULL;
76     pP->pBox = NULL;
77 }
78
79 // -----
80 // AP button
81 // -----
82
83 // struct for a AP button
84 typedef struct SAPgtkButton {
85     GtkWidget *    pB;
86                     //!< pointer to the button
87     int            pressCounter;
88                     //!< counter of the press event
89 } TAPgtkButton;
90
91 // call back to handle click calls
92 void gtkAP_cb_ButtonClicked (GtkWidget *widget, gpointer
93     data) {
94     TAPgtkButton * pB = (TAPgtkButton *) data;
95     pB->pressCounter++;
96 }
97
98 // creates a AP button
99 int gtkAP_local_ButtonCreate (
100     TAPgtkButton *    pB,           //!<(in/
101                                     out) structure to be filled
102     TAPgtkPanel *    pP,           //!<(in)
103                                     panel witch holds the button
104     int                x,
105                                     //!<(in) x position of the
106     int                window      y,
107                                     //!<(in) y position of the
108     int                window      xle,
109                                     //!<(in) x length of the window
110     int                window      yle,
111                                     //!<(in) x length of the
112                                     window
113 ) {
114
115     pB->pressCounter = 0;
116     // create button
117     pB->pB = gtk_button_new_with_label  ("");
118     if (!pB->pB) return -1;

```

```

110
111     // bind it to the window
112     gtk_fixed_put(GTK_FIXED(pP->pBox), pB->pB, x, y)
113     ;
114     gtk_widget_set_size_request(pB->pB, xle, yle);
115
116     gtk_widget_modify_font(gtk_bin_get_child (
117         GTK_BIN(pB->pB)), gAPgtkMonoSpaceFont);
118
119     gtk_widget_show (pB->pB);
120
121     // set msg handler
122     g_signal_connect(G_OBJECT(pB->pB), "clicked",
123         G_CALLBACK(gtkAP_cb_ButtonClicked), pB);
124
125     return 0;
126 }
127
128 // destroys a button
129 void gtkAP_local_ButtonDestroy (
130     TAPgtkButton * pB
131 ) {
132     APgtk_helper_destroy(pB->pB);
133
134     pB->pB = NULL;
135     pB->pressCounter = 0;
136 }
137
138 // -----
139 // AP LED
140 // -----
141
142 // struct for a AP led
143 typedef struct SAPgtkLED {
144     GtkWidget * pLED;
145     //!< pointer to the LED
146     int onFlag;
147     double onColor[3];
148     double offColor[3];
149 } TAPgtkLED;
150
151 static gboolean APgtk_cb_LEDexpose (
152     GtkWidget * da,
153     GdkEventExpose * event,
154     gpointer data
155 )
156 {
157     cairo_t * cr;
158     TAPgtkLED * pLED = (TAPgtkLED *) data;
159     double r;
160     cr = gdk_cairo_create (da->window);
161     gdk_cairo_rectangle (cr, &event->area);
162     cairo_clip (cr);

```

```

161 if (pLED->onFlag) {
162     cairo_set_source_rgb (cr, pLED->onColor[0],
163                           pLED->onColor[1], pLED->onColor[2]);
164 } else {
165     cairo_set_source_rgb (cr, pLED->offColor[0],
166                           pLED->offColor[1], pLED->offColor[2]);
167 }
168 //cairo_rectangle(cr, event->area.x, event->area.y,
169 //                event->area.width, event->area.height);
170 r = (event->area.width > event->area.height) ? event->
171     area.width : event->area.height;
172 r /= 2.f;
173 cairo_arc (cr, event->area.x+r, event->area.y+r, r, 0.f,
174           2.f * M_PI);
175
176 cairo_fill(cr);
177 cairo_destroy (cr);
178 return TRUE;
179 }
180
181 // creates a AP button
182 int gtkAP_local_LEDCreate (
183     TAPgtkLED * pLED,    //!<(in/out)
184                          structure to be filled
185     TAPgtkPanel * pP,    //!<(in)
186                          panel witch holds the button
187     int x,
188                          //!<(in) x position of the
189     window
190     int y,
191                          //!<(in) y position of the
192     window
193     int xle,
194     //!<(in) x length of the window
195     int yle,
196     //!<(in) x length of the
197     window
198 ) {
199     pLED->onFlag = 0;
200     pLED->offColor[0] = 0.0;
201     pLED->offColor[1] = 0.0;
202     pLED->offColor[2] = 0.0;
203     pLED->onColor[0] = 1.0;
204     pLED->onColor[1] = 1.0;
205     pLED->onColor[2] = 1.0;
206
207     pLED->pLED = gtk_drawing_area_new ();
208     if (!pLED->pLED) return -1;
209     gtk_widget_set_size_request (pLED->pLED, xle,
210                                 yle);
211
212     gtk_fixed_put(GTK_FIXED(pP->pBox), pLED->pLED, x
213                  , y);

```

```

200         g_signal_connect (pLED->pLED, "expose-event",
201             G_CALLBACK (APgtk_cb_LEDexpose), pLED);
202
203         gtk_widget_show (pLED->pLED);
204
205         return 0;
206     }
207
208     // destroys a button
209     void gtkAP_local_LEDDestroy (
210         TAPgtkLED *      pLED
211     ) {
212         APgtk_helper_destroy(pLED->pLED);
213
214         pLED->pLED = NULL;
215         pLED->onFlag = 0;
216         pLED->offColor[0] = 0.0;
217         pLED->offColor[1] = 0.0;
218         pLED->offColor[2] = 0.0;
219         pLED->onColor[0] = 0.0;
220         pLED->onColor[1] = 0.0;
221         pLED->onColor[2] = 0.0;
222     }
223
224     // -----
225     // AP Display
226     // -----
227
228     // struct for a AP led
229     typedef struct SAPgtkDisplay {
230         GtkWidget *      pD;
231         //!< pointer to the display
232         GtkWidget *      pF;
233         //!< frame around the label
234         int               charsPerLine;
235         int               lineCount;
236     } TAPgtkDisplay;
237
238     // creates a AP display
239     int gtkAP_local_DisplayCreate (
240         TAPgtkDisplay * pD,          //!<(in/
241         out) structure to be filled
242         TAPgtkPanel *   pP,          //!<(in)
243         panel witch holds the button
244         int              x,
245         //!<(in) x position of the
246         window
247         int              y,
248         //!<(in) y position of the
249         window
250         int              xle,
251         //!<(in) x length of the window

```

```

245         int                                     yle
                //!(in) x length of the
                window
246     ) {
247
248         pD->pD = NULL;
249         pD->charsPerLine = 10;
250         pD->lineCount = 4;
251
252         pD->pD = gtk_label_new ("");
253         if (!pD->pD) return -1;
254         pD->pF = gtk_frame_new(NULL);
255         if (!pD->pF) return -2;
256
257         //gtk_widget_set_size_request (pD->pD, xle-2,
                yle-2);
258         gtk_widget_set_size_request (pD->pF, xle, yle);
259
260         //gtk_fixed_put(GTK_FIXED(pP->pBox), pD->pD, x,
                y);
261         gtk_fixed_put(GTK_FIXED(pP->pBox), pD->pF, x, y)
                ;
262         gtk_container_add(GTK_CONTAINER(pD->pF), pD->pD)
                ;
263
264         //gtk_widget_modify_font(gtk_bin_get_child (
                GTK_BIN(pD->pD)),gAPgtkMonoSpaceFont);
265         gtk_widget_modify_font(pD->pD,
                gAPgtkMonoSpaceFont);
266
267         /*
268         // change colors
269         GdkColor color;
270         gdk_color_parse ("yellow", &color);
271         gtk_widget_modify_fg (pD->pD, GTK_STATE_NORMAL,
                &color);
272         // background only works for the main wnd!
273         gdk_color_parse ("green", &color);
274         gtk_widget_modify_bg(pP->pWnd, GTK_STATE_NORMAL,
                &color);
275         */
276
277         gtk_widget_show (pD->pD);
278         gtk_widget_show (pD->pF);
279
280         return 0;
281     }
282
283     // destroys a display
284     void gtkAP_local_DisplayDestroy (
285         TAPgtkDisplay *                                pD
286     ) {
287         APgtk_helper_destroy(pD->pD);
288
289         pD->pD = NULL;

```

```

290         pD->charsPerLine = 10;
291         pD->lineCount = 2;
292     }
293
294
295     // destroys a button
296     void gtkAP_local_DisplaySetText(
297         TAPgtkDisplay *          pD,
298         char *                   szTxt
299     ) {
300         const char const cLF = 0xA;
301         char szdummyBuffer[pD->lineCount*pD->
302             charsPerLine+1];
303         char * pC = szdummyBuffer;
304         int x = 0;
305         int y = 0;
306
307         while (*szTxt) {
308             // check if we have a line break
309             if (*szTxt == cLF) {
310                 y++;
311                 x = 0;
312                 // check if we reached the end
313                 if (y == pD->lineCount) {
314                     *pC = 0;
315                     goto printLines;
316                 }
317
318                 // copy char
319                 *pC = *szTxt;
320                 szTxt++;
321                 pC++;
322                 x++;
323
324                 // check if we reached the end of the
325                 // line
326                 if (x == pD->charsPerLine) {
327                     y++;
328                     x=0;
329                     // check if we reached the end
330                     if (y < pD->lineCount) {
331                         *pC = cLF;
332                         pC++;
333                     } else {
334                         *pC = 0;
335                         goto printLines;
336                     }
337                 }
338             }
339             *pC = 0;
340         printLines:
341         gtk_label_set_text(GTK_LABEL(pD->pD),
342             szdummyBuffer);

```

```

342 }
343
344 // =====
345 // AP gtk+ interface
346 // =====
347
348 enum eAPgtkActionID {
349     eAPgtkActionID_exit
350         = 1,
351     eAPgtkActionID_redrawUI
352         = 10,
353     eAPgtkActionID_setDisplayText
354         = 11,
355     eAPgtkActionID_createPanel
356         = 20,
357     eAPgtkActionID_deletePanel
358         = 21,
359     eAPgtkActionID_createButton
360         = 30,
361     eAPgtkActionID_deleteButton
362         = 31,
363     eAPgtkActionID_createLED
364         = 40,
365     eAPgtkActionID_deleteLED
366         = 41,
367     eAPgtkActionID_createDisplay
368         = 50,
369     eAPgtkActionID_deleteDisplay
370         = 51,
371 };
372
373 typedef struct SAPgtk_command {
374     sem_t          ps;
375     process finsh semaphore
376     GMutex *       gm;
377     guarding mutex
378     int            newCmdFlag;
379     set to indicate that there is a new command
380
381     int            cmd;
382     void *         pData1;
383     void *         pData2;
384     int            x;
385     int            y;
386     int            xle;
387     int            yle;
388 } TAPgtk_command;
389
390 TAPgtk_command      gAPgtkMsg;
391 GThread *           gAPgtkThread;
392
393 void APgtk_setMsg (
394     int            id,
395     void *         pD1,
396     void *         pD2,
397     int            x,
398     int            y,
399     int            xle,
400     int            yle
401 ) {
402     g_mutex_lock(gAPgtkMsg.gm);

```



```

393     // set the command
394     gAPgtkMsg.cmd = id;
395     gAPgtkMsg.pData1 = pD1;
396     gAPgtkMsg.pData2 = pD2;
397     gAPgtkMsg.x = x;
398     gAPgtkMsg.y = y;
399     gAPgtkMsg.xle = xle;
400     gAPgtkMsg.yle = yle;
401     gAPgtkMsg.newCmdFlag = 1;
402     g_mutex_unlock(gAPgtkMsg.gm);
403
404     sem_wait(&gAPgtkMsg.ps);
405 }
406
407 gpointer APgtk_threadFunc (gpointer Args) {
408     // init gtk and bind it to the thread
409     if (FALSE == gtk_init_check(NULL, NULL)) return
        ((gpointer)-1);
410
411     gAPgtkMonoSpaceFont =
        pango_font_description_from_string("monospace
        ");
412
413     // process messages & user actions
414     for (;;) {
415         // check for message
416         g_mutex_lock(gAPgtkMsg.gm);
417         if (gAPgtkMsg.newCmdFlag) {
418             switch (gAPgtkMsg.cmd) {
419                 case eAPgtkActionID_exit
420                     :
421                     goto exit;
422                     break;
423                 case
424                     eAPgtkActionID_redrawUI
425                     :
426                     gtk_widget_queue_draw
427                         (GTK_WIDGET(
428                             gAPgtkMsg.
429                             pData1));
430                     break;
431                 case
432                     eAPgtkActionID_setDisplayText
433                     :
434                     gtkAP_local_DisplaySetText
435                         (gAPgtkMsg.
436                             pData1,
437                             gAPgtkMsg.
438                             pData2);
439                     break;
440                 case
441                     eAPgtkActionID_createPanel
442                     :

```

```

431         gtkAP_local_PanelCreate
432             (
433                 gAPgtkMsg
434                 .
435                 pData1
436                 ,
437                 gAPgtkMsg
438                 .
439                 x
440                 ,
441                 gAPgtkMsg
442                 .
443                 y
444                 ,
445                 gAPgtkMsg
446                 .
447                 xle
448                 ,
449                 gAPgtkMsg
450                 .
451                 yle
452                 ,
453                 );
454         break;
455     case
456         eAPgtkActionID_deletePanel
457         :
458             gtkAP_local_PanelDestroy
459                 (gAPgtkMsg.
460                 pData1);
461             break;
462     case
463         eAPgtkActionID_createButton
464         :
465             gtkAP_local_ButtonCreate
466                 (
467                 gAPgtkMsg
468                 .
469                 pData1
470                 ,
471                 gAPgtkMsg
472                 .
473                 pData2
474                 ,
475                 gAPgtkMsg
476                 .
477                 x
478                 ,
479                 gAPgtkMsg
480                 .
481                 y
482                 ,
483                 gAPgtkMsg
484                 .
485                 xle
486                 ,
487                 gAPgtkMsg
488                 .
489                 yle
490                 ,
491                 );
492             break;
493     }
494 }

```

```

448                                     gAPgtkMsg
                                        .
                                        y
                                        ,
449                                     gAPgtkMsg
                                        .
                                        xle
                                        ,
450                                     gAPgtkMsg
                                        .
                                        yle
                                        ,
451                                     );
452                                     break;
453     case
454         eAPgtkActionID_deleteButton
455         :
456             gtkAP_local_ButtonDestroy
457             (gAPgtkMsg.
458              pData1);
459             break;
460     case
461         eAPgtkActionID_createLED
462         :
463             gtkAP_local_LEDCreate
464             (
465                 gAPgtkMsg
466                 .
467                 pData1
468                 ,
469                 gAPgtkMsg
470                 .
471                 pData2
472                 ,
473                 gAPgtkMsg
474                 .
475                 x
476                 ,
477                 gAPgtkMsg
478                 .
479                 y
480                 ,
481                 gAPgtkMsg
482                 .
483                 xle
484                 ,

```

```

464                                     gAPgtkMsg
                                        .
                                        yle
465                                     );
466                                     break;
467     case
468         eAPgtkActionID_deleteLED
469         :
470             gtkAP_local_LEDDestroy
471             (gAPgtkMsg.
472              pData1);
473             break;
474     case
475         eAPgtkActionID_createDisplay
476         :
477             gtkAP_local_DisplayCreate
478             (
479                 gAPgtkMsg
480                 .
481                 pData1
482                 ,
483                 gAPgtkMsg
484                 .
485                 pData2
486                 ,
487                 gAPgtkMsg
488                 .
489                 x
490                 ,
491                 gAPgtkMsg
492                 .
493                 y
494                 ,
495                 gAPgtkMsg
496                 .
497                 xle
498                 ,
499                 gAPgtkMsg
500                 .
501                 yle
502             );
503             break;
504     case
505         eAPgtkActionID_deleteDisplay
506         :
507             gtkAP_local_DisplayDestroy
508             (gAPgtkMsg.

```

```

483                                     pData1);
484                                     break;
485                                     }
486                                     gAPgtkMsg.newCmdFlag = 0;
487                                     sem_post(&gAPgtkMsg.ps);
488                                     }
489                                     g_mutex_unlock(gAPgtkMsg.gm);
490                                     // message handling
491                                     if (TRUE == gtk_events_pending ()) {
492                                         gtk_main_iteration ();
493                                     }
494
495                                     }
496                                     goto end;
497 exit:
498     sem_post(&gAPgtkMsg.ps);
499     g_mutex_unlock(gAPgtkMsg.gm);
500 end:
501     pango_font_description_free(gAPgtkMonoSpaceFont)
502     ;
503     gAPgtkMonoSpaceFont = NULL;
504     return NULL;
505 }
506 int APgtk_start () {
507
508     if (!g_thread_supported ()) g_thread_init (NULL)
509     ;
510
511     gAPgtkMsg.newCmdFlag = 0; // no cmd ready
512     gAPgtkMsg.gm = g_mutex_new ();
513     sem_init(&gAPgtkMsg.ps,0,0);
514
515     gAPgtkThread = g_thread_create(APgtk_threadFunc ,
516     NULL,TRUE, NULL);
517     return 0;
518 }
519 void APgtk_end () {
520     APgtk_setMsg(eAPgtkActionID_exit, NULL, NULL, 0,
521     0, 0, 0);
522     g_thread_join(gAPgtkThread);
523     g_mutex_free(gAPgtkMsg.gm);
524     sem_destroy(&gAPgtkMsg.ps);
525 }
526 // -----
527 // AP interface
528 // -----
529
530 // create a panel
531 int gtkAP_PanelCreate (
532     TAPgtkPanel *    pP,

```

```

//!<(in/

```

```

533         out) structure to be filled
           int x,
           window //!<(in) x position of the
534           int y,
           window //!<(in) y position of the
535           int xle,
           //!<(in) x length of the window
536           int yle
           //!<(in) x length of the
           window
537     ) {
538
539     APgtk_setMsg (eAPgtkActionID_createPanel, pP,
540                  NULL, x, y, xle, yle);
541
542     return(0);
543 }
544
545 // destroys a panel
546 void gtkAP_PanelDestroy (
547     TAPgtkPanel * pP
548 ) {
549     APgtk_setMsg (eAPgtkActionID_deletePanel, pP,
550                  NULL, 0,0,0,0);
551 }
552
553 // create a button
554 int gtkAP_ButtonCreate (
555     TAPgtkButton * pB, //!<(in/
556     out) structure to be filled
557     TAPgtkPanel * pP, //!<(in)
558     panel witch holds the button
559     int x,
560     window //!<(in) x position of the
561     int y,
562     window //!<(in) y position of the
563     int xle,
564     //!<(in) x length of the window
565     int yle
566     //!<(in) x length of the
567     window
568 ) {
569
570     APgtk_setMsg (eAPgtkActionID_createButton, pB,
571                  pP, x, y, xle, yle);
572
573     return(0);
574 }

```

```

568
569 // destroys a button
570 void gtkAP_ButtonDestroy (
571     TAPgtkButton * pB
572 ) {
573     APgtk_setMsg (eAPgtkActionID_deleteButton, pB,
574                   NULL, 0,0,0,0);
575 }
576
577 // create a LED
578 int gtkAP_LEDCreate (
579     TAPgtkLED * pLED,    //!<(in/out)
580     structure to be filled
581     TAPgtkPanel * pP,    //!<(in)
582     panel witch holds the button
583     int x,
584     //!<(in) x position of the
585     window
586     int y,
587     //!<(in) y position of the
588     window
589     int xle,
590     //!<(in) x length of the window
591     int yle,
592     //!<(in) x length of the
593     window
594 ) {
595     APgtk_setMsg (eAPgtkActionID_createLED, pLED, pP
596                 , x, y, xle, yle);
597
598     return(0);
599 }
600
601 // destroys a LED
602 void gtkAP_LEDDestroy (
603     TAPgtkLED * pLED
604 ) {
605     APgtk_setMsg (eAPgtkActionID_deleteLED, pLED,
606                   NULL, 0,0,0,0);
607 }
608
609 // create a display
610 int gtkAP_DisplayCreate (
611     TAPgtkDisplay * pD,    //!<(in/
612     out) structure to be filled
613     TAPgtkPanel * pP,    //!<(in)
614     panel witch holds the button
615     int x,
616     //!<(in) x position of the
617     window
618     int y,
619     //!<(in) y position of the

```

```

        window
        int xle,
        //!(in) x length of the window
        int yle
        //!(in) x length of the
        window
    ) {
        APgtk_setMsg (eAPgtkActionID_createDisplay, pD,
            pP, x, y, xle, yle);
        return(0);
    }

// destroys a display
void gtkAP_DisplayDestroy (
    TAPgtkDisplay * pD
) {
    APgtk_setMsg (eAPgtkActionID_deleteDisplay, pD,
        NULL, 0,0,0,0);
}

// destroys a display
void gtkAP_DisplaySetText (
    TAPgtkDisplay * pD,
    char * pT
) {
    APgtk_setMsg (eAPgtkActionID_setDisplayText, pD,
        pT, 0,0,0,0);
}

// =====
// AP gtk general types and interface
// =====

// AP gtk+ type enums
enum eAPgtkUITypes {
    eAPgtkUIType_unknown = 0,
    eAPgtkUIType_panel = 1,
    eAPgtkUIType_button = 2,
    eAPgtkUIType_LED = 3,
    eAPgtkUIType_display = 4
};

// AP UI type
typedef struct SAPgtkUI {
    int uuid;
    int x;
    int y;
    int xle;
    int yle;
    int typeID;
    union uAPgtkUI {

```



```

654         TAPgtkPanel      panel;
655         TAPgtkButton      button;
656         TAPgtkDisplay      display;
657         TAPgtkLED          led;
658     } ui;
659 } TAPgtkUI;
660
661 // set the coordinates of the ui
662 void APgtkUI_setCoordinates (
663     TAPgtkUI *      pUI,
664     int             x,
665     int             y,
666     int             xle,
667     int             yle
668 ) {
669     pUI->x = x;
670     pUI->y = y;
671     pUI->xle = xle;
672     pUI->yle = yle;
673 }
674
675 // creates visible ui
676 int APgtkUI_createUI (
677     TAPgtkUI *      pUI,
678     TAPgtkUI *      pParentUI,
679     int             uuid,
680     int             typeID
681 ) {
682     pUI->typeID = typeID;
683     pUI->uuid = uuid;
684     switch (typeID) {
685         case eAPgtkUItype_panel:
686             return gtkAP_PanelCreate(
687                 &(pUI->ui.panel),
688                 pUI->x,
689                 pUI->y,
690                 pUI->xle,
691                 pUI->yle
692             );
693         case eAPgtkUItype_button:
694             return gtkAP_ButtonCreate(
695                 &(pUI->ui.button),
696                 &(pParentUI->ui.panel),
697                 pUI->x,
698                 pUI->y,
699                 pUI->xle,
700                 pUI->yle
701             );
702         case eAPgtkUItype_LED:
703             return gtkAP_LEDCreate(
704                 &(pUI->ui.led),
705                 &(pParentUI->ui.

```

```

706         panel),
707         pUI->x,
708         pUI->y,
709         pUI->xle,
710         pUI->yle
711     );
712     case eAPgtkUItype_display:
713         return gtkAP_DisplayCreate(
714             &(pUI->ui.
715                 display),
716             &(pParentUI->ui.
717                 panel),
718             pUI->x,
719             pUI->y,
720             pUI->xle,
721             pUI->yle
722         );
723     }
724     return -100;
725 }
726
727 // destroy visible ui
728 void APgtkUI_destroyUI (
729     TAPgtkUI *      pUI
730 ) {
731     switch (pUI->typeID) {
732         case eAPgtkUItype_panel:
733             gtkAP_PanelDestroy(&(pUI->ui.
734                 panel));
735             break;
736         case eAPgtkUItype_button:
737             gtkAP_ButtonDestroy(&(pUI->ui.
738                 button));
739             break;
740         case eAPgtkUItype_LED:
741             gtkAP_LEDDestroy(&(pUI->ui.led))
742             ;
743             break;
744         case eAPgtkUItype_display:
745             gtkAP_DisplayDestroy(&(pUI->ui.
746                 display));
747             break;
748     }
749     pUI->typeID = eAPgtkUItype_unknown;
750     pUI->uuid = 0;
751 }
752
753 // redraws a ui
754 void APgtkUI_redrawUI (
755     TAPgtkUI *      pUI
756 ) {
757     switch (pUI->typeID) {
758         case eAPgtkUItype_panel:
759             APgtk_setMsg (
760                 eAPgtkActionID_redrawUI, pUI

```

```

753         ->ui.panel.pWnd, NULL,
754         0,0,0,0);
755         break;
756     case eAPgtkUIType_button:
757         APgtk_setMsg (
758             eAPgtkActionID_redrawUI, pUI
759             ->ui.button.pB, NULL,
760             0,0,0,0);
761         break;
762     case eAPgtkUIType_LED:
763         APgtk_setMsg (
764             eAPgtkActionID_redrawUI, pUI
765             ->ui.led.pLED, NULL, 0,0,0,0)
766         ;
767         break;
768     case eAPgtkUIType_display:
769         APgtk_setMsg (
770             eAPgtkActionID_redrawUI, pUI
771             ->ui.display.pD, NULL,
772             0,0,0,0);
773         break;
774     }
775 }
776
777 // AP UI array type
778 typedef struct SAPgtkUIvector {
779     TAPgtkUI *      pUI;
780     int              number;
781 } TAPgtkUIvector;
782
783 TAPgtkUIvector * APgtkUI_createVector (
784     int number,
785     int uiType
786 ) {
787     TAPgtkUIvector *      pV;
788     int                    i;
789
790     pV = malloc (sizeof(TAPgtkUIvector));
791     if (!pV) return NULL;
792     pV->pUI = malloc (sizeof(TAPgtkUI)*number);
793     if (!pV->pUI) {
794         free (pV);
795         return NULL;
796     }
797     memset (pV->pUI, 0, sizeof(TAPgtkUI)*number);
798
799     pV->number = number;
800     for (i = 0; i < number; i++) {
801         pV->pUI[i].typeID = uiType;
802     }
803
804     return pV;
805 }

```

```

797
798 void APgtkUI_destroyVector (
799     TAPgtkUIvector * pUIv
800 ) {
801     int i;
802     for (i = 0; i < pUIv->number; i++) {
803         APgtkUI_destroyUI (&(pUIv->pUI[i]));
804     }
805     free (pUIv->pUI);
806     free (pUIv);
807 }

```

3.1.11 libsndfile frame based

includes:

c-Include	c-Library	system lib
sndfile.h	sndfile-1	yes
string.h		yes

code:

```

1 // #####
2 // block based processing without overlapped operations
3 // #####
4
5 #define dStjFrameWAVinitialFrameAmount (1024)
6
7 typedef struct SStjFrameWAVOpenInfo {
8     unsigned int    channelNumber;
9     //!< logical channel
10    char *          szFileName;
11    //!< name of the file
12    unsigned int     isInput;
13    //!< if <>0 then this file is an
14    input
15
16    unsigned int     sampleRate;
17    //!< the sample rate
18    unsigned int     fileChannelNumber;
19    //!< number of the channel at the file (
20    starting at 1)
21 } TStjFrameWAVOpenInfo;
22
23 typedef struct SStjFrameWAVFile {
24     unsigned int     channel;
25     //!< the channel
26     SNDFILE *        pSndF;
27     //!< the file
28     unsigned int     isInput;
29     //!< if 0 the channel is an input
30     channel

```

```

20
21
22         float *                pFrameBuffer;
           //!< the buffer witch is filled with
           the frames
23         int                    fileChannels;
           //!< the amount of sample
           channels at the file
24         int                    activeChannel;
           //!< the channel used to read
25         int                    frameAmount;
           //!< amount of frame
26 } TStjFrameWAVFile;
27
28 typedef struct SStjFrameWAVmodule {
29     TStjFrameWAVFile *        pChannels;
30     unsigned int              number;
31 } TStjFrameWAVmodule;
32
33 //! close the set of wav files
34 void FrameWAVmoduleExit (TStjFrameWAVmodule * pM);
35
36 //! initis a set of wav files for writing / reading
37 int FrameWAVmoduleInit (
38     int
           number,                //!<(in) amount of
           files
39     TStjFrameWAVOpenInfo *    pWAVFiles,
           //!<(in) file description
40     TStjFrameWAVmodule *      pModul
           //!<(in/out) modul descriptor witch
           is filled
41 );
42
43 //! looking for the right channel
44 TStjFrameWAVFile * FrameWAVmoduleFindChannel (
45     TStjFrameWAVmodule *      pM,
           //!< the module
46     int                        channel
           //!< channel to search for
47 );
48
49 //! reads a sample from a channel
50 int FrameWAVmoduleGetInput (
51     TStjFrameWAVmodule *      pM,
           //!<(in) the module
52     int                        channel,
           //!<
           in) the channel
53     int                        sampleNumber, //!<(in)
           number of samples to be read
54     float *                    pFrame
           //!<(out)
           pointer to the sample buffer witch is

```

```

55         filled
56     );
57     /*! writes a sample to an output
58     int FrameWAVmoduleSetOutput (
59         TStjFrameWAVmodule *    pM,
60         int                      /*!<(in) the module
61         channel,                  /*!<(
62         in) the channel
63         int                      sampleNumber,  /*!<(in)
64         number of samples to be write
65         float *
66         pFrame                    /*!<(in)
67         pointer to the sample buffer witch is
68         filled
69     );
70
71     /*! close the set of wav files
72     void FrameWAVmoduleExit (TStjFrameWAVmodule * pM) {
73         //1. close all sound files
74         int i;
75
76         TStjFrameWAVFile * pWF = pM->pChannels;
77         for (i = 0; i < pM->number;i++) {
78             sf_close(pWF->pSndF);
79             if (pWF->pFrameBuffer) free (pWF->
80             pFrameBuffer);
81             pWF++;
82         }
83         //2. free array
84         free (pM->pChannels);
85
86         //3. set all vars of the struct to default
87         pM->number = 0;
88         pM->pChannels = NULL;
89     }
90
91     /*! initis a set of wav files for writing / reading
92     int FrameWAVmoduleInit (
93         int
94         number,                    /*!<(in) amount of
95         files
96         TStjFrameWAVOpenInfo *    pWAVFiles,
97         /*!<(in) file description
98         TStjFrameWAVmodule *      pModul
99         /*!<(in/out) modul descriptor witch
100         is filled
101     ) {
102
103         pModul->pChannels = malloc (sizeof(
104             TStjFrameWAVFile) * number);
105         if (!pModul->pChannels) {

```

```

94         return -1;
95     }
96     memset (pModul->pChannels,0,sizeof(
97         TStjFrameWAVFile) * number);
98     pModul->number = number;
99     int i;
100     SF_INFO info;
101     TStjFrameWAVFile * pWF = pModul->pChannels;
102
103     for (i = 0;i < number;i++) {
104         // open sndfile interface
105         memset (&info,0,sizeof(SF_INFO));
106         if (pWAVFiles[i].isInput) {
107             pWF->pSndF = sf_open (pWAVFiles[
108                 i].szFileName,SFM_READ, &info
109             );
110             info.samplerate = pWAVFiles[i].
111                 sampleRate;
112             info.channels = 1;
113             info.format = SF_FORMAT_WAV |
114                 SF_FORMAT_FLOAT;
115             pWF->pSndF = sf_open (pWAVFiles[
116                 i].szFileName,SFM_WRITE, &
117                 info);
118             } else {
119                 info.samplerate = pWAVFiles[i].
120                     sampleRate;
121                 info.channels = 1;
122                 info.format = SF_FORMAT_WAV |
123                     SF_FORMAT_FLOAT;
124                 pWF->pSndF = sf_open (pWAVFiles[
125                     i].szFileName,SFM_WRITE, &
126                     info);
127             }
128             if (!pWF->pSndF) {
129                 goto error;
130             }
131             pWF->isInput = (unsigned int) pWAVFiles[
132                 i].isInput;
133             pWF->channel = (unsigned int) pWAVFiles[
134                 i].channelNumber;
135
136             if (pWAVFiles[i].isInput) {
137                 pWF->pFrameBuffer = malloc (
138                     sizeof(float)*info.channels*
139                     dStjFrameWAVinitialFrameAmount
140                 );
141                 if (!pWF->pFrameBuffer) {
142                     goto error;
143                 }
144                 pWF->frameAmount =
145                     dStjFrameWAVinitialFrameAmount
146                     ;
147             }
148
149             // the file channel starts at 1 - here
150             // we start at 0
151             pWF->activeChannel = pWAVFiles[i].
152                 fileChannelNumber - 1;
153             pWF->fileChannels = (int) info.channels;
154         }
155     }

```

```

133         pWF++;
134     }
135     return 0;
136 error:
137     FrameWAVmoduleExit(pModul);
138     return -2;
139 }
140
141 //! looking for the right channel
142 TStjFrameWAVFile * FrameWAVmoduleFindChannel (
143     TStjFrameWAVmodule * pM,
144     //!< the module
145     int
146     channel //!< channel to
147     search for
148 ) {
149     TStjFrameWAVFile * pWF = pM->pChannels;
150     int i;
151     for (i = 0; i < pM->number;i++) {
152         if (pWF->channel == channel) return pWF;
153         pWF++;
154     }
155     return NULL;
156 }
157
158 //! reads a sample from a channel
159 int FrameWAVmoduleGetInput (
160     TStjFrameWAVmodule * pM,
161     //!<(in) the module
162     int
163     channel, //!<(
164     in) the channel
165     int
166     sampleNumber, //!<(in)
167     number of samples to be read
168     float *
169     pFrame //!<(out)
170     pointer to the sample buffer witch is
171     filled
172 ) {
173     TStjFrameWAVFile * pWF =
174         FrameWAVmoduleFindChannel(pM,channel);
175     // channel not found
176     if (!pWF) {
177         return -1;
178     }
179     // channel is not an input
180     if (!pWF->isInput) {
181         return -2;
182     }
183     // check the temp buffer size
184     if (pWF->frameAmount < sampleNumber) {
185         pWF->pFrameBuffer = realloc(pWF->
186             pFrameBuffer, sizeof(float)*pWF->

```



```

175         fileChannels*sampleNumber);
176         if (!pWF->pFrameBuffer) {
177             return -3;
178         }
179         pWF->frameAmount = sampleNumber;
180     }
181     // fill the buffer
182     int am = (int) sf_readf_float (pWF->pSndF, pWF->
        pFrameBuffer, sampleNumber);
183
184     if (am <= 0) {
185         return -4;
186     }
187
188     // copy buffer
189     int i;
190     float * pD = pFrame;
191     float * pS = pWF->pFrameBuffer;
192     // add the offset so we start at the right pos
        at the frame
193     pS += pWF->activeChannel;
194     for (i = 0; i < am; i++) {
195         *pD = *pS;
196         pD++;
197         pS+= pWF->fileChannels;
198     }
199     // and fill the rest with 0.
200     for (; i < sampleNumber; i++) {
201         *pD = 0.0f;
202         pD++;
203     }
204     return 0;
205 }
206
207 //! writes a sample to an output
208 int FrameWAVmoduleSetOutput (
209     TStjFrameWAVmodule * pM,
        //!<(in) the module
210     int
        channel,
        //!<(
        in) the channel
211     int
        sampleNumber, //!<(in)
        number of samples to be write
212     float *
        pFrame //!<(in)
        pointer to the sample buffer witch is
        filled
213 ) {
214     TStjFrameWAVFile * pWF =
        FrameWAVmoduleFindChannel(pM, channel);
215     // channel not found
216     if (!pWF) {
217         return -1;

```

```

218     }
219     // channel is not an output
220     if (pWF->isInput) {
221         return -2;
222     }
223     // the last samples at the frame are new
224     if (sampleNumber != sf_write_float (pWF->pSndF,
225         pFrame, sampleNumber)) return -3;
226     return 0;
227 }

```

3.1.12 libsndfile overlapped frame based

includes:

c-Include	c-Library	system lib
sndfile.h	sndfile-1	no
string.h		yes

code:

```

1 //
2 // =====
3 // descr: some helpers for the lib-snd-file group
4 // author: Stefan Jaritz
5 //
6 // the lib is the "libsndfile-1"
7 //
8 // =====
9
10 // defines
11
12 typedef struct SStjFrameOverlappedWAVOpenInfo {
13     unsigned int    channelNumber;
14     //!< logical channel
15     char *          szFileName;
16     //!< name of the file
17     unsigned int    isInput;
18     //!< if <>0 then this file is an
19     input
20
21     unsigned int    sampleRate;
22     //!< the sample rate
23     unsigned int    fileChannelNumber;
24     //!< number of the channel at the file (
25     starting at 1)
26
27     unsigned int    frameSize;
28     //!< number of samples per frame
29 }

```

```

19     unsigned int    overlappingNumber;
        //!< the number of samples kept in the frame
        at reading/writing
20 } TStjFrameOverlappedWAVOpenInfo;
21
22 typedef struct SStjFrameOverlappedWAVFile {
23     unsigned int    channel;
        //!< the channel
24     SNDFILE *        pSndF;
        //!< the file
25     unsigned int    isInput;
        //!< if 0 the channel is an input
        channel
26
27     float *          pRB;
        //!< the ring buffer storing/
        writing the samples
28     unsigned int    rbFrameSize;
        //!< the size of the frames at the ring
        buffer
29     unsigned int    FrameSize;
        //!< amount of samples at the frame
30     unsigned int    nMax;
        //!< number of elements at the ring
        buffer
31     unsigned int    n;
        //!< actual element (for
        inputs)/ the offset (output)
32 } TStjFrameOverlappedWAVFile;
33
34 typedef struct SStjFrameOverlappedWAVmodule {
35     TStjFrameOverlappedWAVFile *    pChannels;
36     unsigned int    number;
37 } TStjFrameOverlappedWAVmodule;
38
39 //!< close the set of wav files
40 void FrameOverlappedWAVmoduleExit (
    TStjFrameOverlappedWAVmodule * pM);
41
42 //!< initis a set of wav files for writing / reading
43 int FrameOverlappedWAVmoduleInit (
44     int
        number,                //!<(in) amount of
        files
45     TStjFrameOverlappedWAVOpenInfo *
        pWAVFiles,            //!<(in) file
        description
46     TStjFrameOverlappedWAVmodule * pModul
        //!<(in/out) modul descriptor
        witch is filled
47     );
48
49 //!< looking for the right channel
50 TStjFrameOverlappedWAVFile *
    FrameOverlappedWAVmoduleFindChannel (

```

```

51         TStjFrameOverlappedWAVmodule * pM,
           //!< the module
52         int channel
           //!< channel to search for
53     );
54
55     //!< reads a sample from a channel
56     int FrameOverlappedWAVmoduleGetInput (
57         TStjFrameOverlappedWAVmodule * pM,
           //!<(in) the module
58         int channel,           //!<(in) the
           channel
59         float *
           pFrame           //!<(out) pointer to
           the sample buffer witch is filled
60     );
61
62     //!< writes a sample to an output
63     int FrameOverlappedWAVmoduleSetOutput (
64         TStjFrameOverlappedWAVmodule * pM,
           //!<(in) the module
65         int channel,           //!<(in) the
           channel
66         float *
           pFrame           //!<(in) pointer to
           the sample buffer witch is filled
67     );
68
69     // functions
70
71     //!< close the set of wav files
72     void FrameOverlappedWAVmoduleExit (
73         TStjFrameOverlappedWAVmodule * pM) {
74         //!<1. close all sound files
75         int i;
76
77         TStjFrameOverlappedWAVFile * pWF = pM->pChannels
78         ;
79         for (i = 0; i < pM->number;i++) {
80             sf_close(pWF->pSndF);
81             free (pWF->pRB);
82             pWF++;
83         }
84         //!<2. free array
85         free (pM->pChannels);
86
87         //!<3. set all vars of the struct to default
88         pM->number = 0;
89         pM->pChannels = NULL;
90     }
91
92     //!< initis a set of wav files for writing / reading

```

```

92 int FrameOverlappedWAVmoduleInit (
93     int
          number,          //!<(in) amount of
          files
94     TStjFrameOverlappedWAVOpenInfo *
          pWAVFiles,      //!<(in) file
          description
95     TStjFrameOverlappedWAVmodule * pModul
          //!<(in/out) modul descriptor
          witch is filled
96 ) {
97
98     pModul->pChannels = malloc (sizeof(
          TStjFrameOverlappedWAVFile) * number);
99     if (!pModul->pChannels) {
100         return -1;
101     }
102     memset (pModul->pChannels,0,sizeof(
          TStjFrameOverlappedWAVFile) * number);
103     pModul->number = number;
104
105     int i;
106     SF_INFO info;
107     TStjFrameOverlappedWAVFile * pWF = pModul->
          pChannels;
108
109     for (i = 0; i < number; i++) {
110         // open sndfile interface
111         memset (&info,0,sizeof(SF_INFO));
112         if (pWAVFiles[i].isInput) {
113             pWF->pSndF = sf_open (pWAVFiles[
          i].szFileName,SFM_READ, &info
          ) ;
114         } else {
115             info.samplerate = pWAVFiles[i].
          sampleRate;
116             info.channels = 1;
117             info.format = SF_FORMAT_WAV |
          SF_FORMAT_FLOAT;
118             pWF->pSndF = sf_open (pWAVFiles[
          i].szFileName,SFM_WRITE, &
          info) ;
119         }
120         if (!pWF->pSndF) {
121             goto error;
122         }
123         pWF->isInput = (unsigned int) pWAVFiles[
          i].isInput;
124         pWF->channel = (unsigned int) pWAVFiles[
          i].channelNumber;
125
126         // prepare buffers
127         pWF->FrameSize = pWAVFiles[i].frameSize;
128         if (pWAVFiles[i].frameSize % pWAVFiles[i]
          .overlappingNumber) {

```

```

129         goto error;
130     }
131     pWF->nMax = pWAVFiles[i].frameSize /
132     pWAVFiles[i].overlappingNumber;
133     pWF->rbFrameSize = pWAVFiles[i].
134     overlappingNumber;
135     // if it's an input we need a memory for
136     the old frames witch are read before
137     if (pWAVFiles[i].isInput) {
138         pWF->pRB = malloc(sizeof(float)*
139         pWAVFiles[i].frameSize);
140         if (!pWF->pRB){
141             goto error;
142         }
143         memset(pWF->pRB,0,sizeof(float)*
144         pWAVFiles[i].frameSize);
145         pWF->n = 0;
146     } else {
147         pWF->pRB = NULL;
148         // the n is used as offset
149         pWF->n = pWAVFiles[i].frameSize
150         - pWAVFiles[i].
151         overlappingNumber;
152     }
153     pWF++;
154 }
155 return 0;
156 error:
157     FrameOverlappedWAVmoduleExit(pModul);
158     return -2;
159 }
160
161 //! looking for the right channel
162 TStjFrameOverlappedWAVFile *
163     FrameOverlappedWAVmoduleFindChannel (
164     TStjFrameOverlappedWAVmodule * pM,
165     //!< the module
166     int channel
167     //!< channel to search for
168     ) {
169     TStjFrameOverlappedWAVFile * pWF = pM->pChannels
170     ;
171     int i;
172     for (i = 0; i < pM->number;i++) {
173         if (pWF->channel == channel) return pWF;
174         pWF++;
175     }
176     return NULL;
177 }
178
179 //! reads a sample from a channel
180 int FrameOverlappedWAVmoduleGetInput (
181     TStjFrameOverlappedWAVmodule * pM,
182     //!<(in) the module
183     int

```

```

172         channel,          //!<(in) the
           channel
           float *
           pFrame          //!<(out) pointer to
                           the sample buffer witch is filled
173     ) {
174     TStjFrameOverlappedWAVFile * pWF =
        FrameOverlappedWAVmoduleFindChannel(pM,
        channel);
175     // channel not found
176     if (!pWF) {
177         return -1;
178     }
179     // channel is not an input
180     if (!pWF->isInput) {
181         return -2;
182     }
183
184     // check the offset at the ringbuffer
185     if (pWF->n >= pWF->nMax) pWF->n = 0;
186
187     int offset;
188     int idx = pWF->rbFrameSize * pWF->n;
189     // fill the buffer
190
191     sf_count_t am = sf_readf_float (pWF->pSndF,&pWF
        ->pRB[idx], (sf_count_t) pWF->rbFrameSize);
192
193     if (am == 0) return -3;
194     if (am != (sf_count_t) pWF->rbFrameSize) {
195         offset = (sf_count_t) am;
196         // calc the missing samples
197         am = (sf_count_t) pWF->rbFrameSize - am;
198         // set the buffer to 0
199         memset(&pWF->pRB[idx+offset],0,sizeof(
            float)*am);
200     }
201     // build the frame
202     // on pos n is the newest frame
203     // n+1 is oldest frame
204     int nStart = pWF->n + 1;
205     int amount;
206
207     // copy all frames left from the newest block
        till the wrapping
208     amount = pWF->nMax - nStart;
209     if (amount) {
210         memcpy(pFrame,&pWF->pRB[nStart*pWF->
            rbFrameSize],sizeof(float)*amount*pWF
            ->rbFrameSize);
211     }
212     nStart = amount;
213     // wrap and copy the rest of it
214     amount = pWF->nMax - amount;
215     memcpy(&pFrame[nStart*pWF->rbFrameSize],pWF->pRB

```

```

216         , sizeof(float)*amount*pWF->rbFrameSize);
217         // remember that we received a frame(with
218         wrapping)
219         pWF->n++;
220         return 0;
221     }
222     /*! writes a sample to an output
223     int FrameOverlappedWAVmoduleSetOutput (
224         TStjFrameOverlappedWAVmodule * pM,
225         //!(in) the module
226         int
227         channel,          //!(in) the
228         channel
229         float *
230         pFrame            //!(in) pointer to
231         the sample buffer witch is filled
232     ) {
233     TStjFrameOverlappedWAVFile * pWF =
234         FrameOverlappedWAVmoduleFindChannel(pM,
235         channel);
236     // channel not found
237     if (!pWF) {
238         return -1;
239     }
240     // channel is not an output
241     if (pWF->isInput) {
242         return -2;
243     }
244     // the last samples at the frame are new
245     if (pWF->rbFrameSize != sf_write_float (pWF->
246         pSndF, &pFrame[pWF->n] , pWF->rbFrameSize))
247         return -3;
248
249     return 0;
250 }
251
252 //
=====

```

3.1.13 libsndfile sample based

includes:

c-Include	c-Library	system lib
sndfile.h	sndfile-1	no

code:

```

1 // =====
2 // descr: some helpers for the lib-snd-file group

```



```

3 // author: Stefan Jaritz
4 //
5 // the lib is the "libsndfile-1"
6 // =====
7
8 typedef struct SStjWAVOpenInfo {
9     unsigned int    channelNumber;
10        //!< logical channel
11    char *          szFileName;
12        //!< name of the file
13    unsigned int    isInput;
14        //!< if <>0 then this file is an
15        input
16    unsigned int    sampleRate;
17        //!< the sample rate
18    unsigned int    fileChannelNumber;
19        //!< number of the channel at the file (
20        starting at 1)
21    unsigned int    sampleAmountCacheSize; //!<
22        amount of samples cached before writing or/
23        at reading
24 } TStjWAVOpenInfo;
25
26 typedef struct SStjWAVFile {
27     unsigned int    channel;
28        //!< the channel
29    SNDFILE *        pSndF;
30        //!< the file
31    unsigned int    isInput;
32        //!< if <> 0 the channel is an input channel
33    unsigned int    channelsAmount; //!< the
34        amount of channels
35    unsigned int    sampleAmount; //!< the
36        amount of samples at the cache
37    unsigned int    maxSampleAmount; //!< the
38        maximum sample amount
39    unsigned int    dataOffset;
40        //!< the offset of the channel at the sample
41        cache
42    float *         pCache;
43        //!< the sample cache
44    float *         pD;
45        //!< pointer to the data
46 } TStjWAVFile;
47
48 typedef struct SStjWAVmodule {
49     TStjWAVFile *   pChannels;
50     unsigned int    number;
51 } TStjWAVmodule;
52
53 // fill the wav sample cache
54 int WAVmoduleFillCache (TStjWAVFile * pWF) {
55     sf_count_t amount = (sf_count_t) pWF->
56         maxSampleAmount;

```

```

38         // read a data frame(s)
39         amount = sf_readf_float (pWF->pSndF,pWF->pCache,
40             amount);
41         if (amount < 1) return -1;
42         // reset data pointers and indexes
43         pWF->pD = pWF->pCache;
44         pWF->sampleAmount = (unsigned int) amount;
45         return 0;
46     }
47     // flushes the wav sample cache to disk
48     int WAVmoduleFlushCache (TStjWAVFile * pWF) {
49         if (!pWF->sampleAmount) return 0;
50         sf_count_t amount = (sf_count_t) pWF->
51             sampleAmount;
52         if (amount != sf_write_float (pWF->pSndF,pWF->
53             pCache , amount)) return -1;
54         pWF->pD = pWF->pCache;
55         pWF->sampleAmount = 0;
56         return 0;
57     }
58     //!! close the set of wav files
59     void WAVmoduleExit (TStjWAVmodule * pM) {
60         //1. close all sound files
61         int i;
62
63         TStjWAVFile * pWF = pM->pChannels;
64         for (i = 0; i < pM->number;i++) {
65             // if it is an output flush the samples
66             // to disk
67             if (!pWF->isInput) {
68                 WAVmoduleFlushCache(pWF);
69             }
70             sf_close(pWF->pSndF);
71             if (pWF->pCache) {
72                 free (pWF->pCache);
73             }
74             pWF++;
75         }
76         //2. free array
77         free (pM->pChannels);
78
79         //3. set all vars of the struct to default
80         pM->number = 0;
81         pM->pChannels = NULL;
82     }
83
84     //!! initis a set of wav files for writing / reading
85     int WAVmoduleInit (
86         int                                     number,
87         //!(in) amount of files
88         TStjWAVOpenInfo * pWAVFiles,

```

```

88          //!(in) file description
          TStjWAVmodule *      pModul
          //!(in/out) modul descriptor witch
          is filled
89      ) {
90
91      pModul->pChannels = malloc (sizeof(TStjWAVFile)
          * number);
92      if (!pModul->pChannels) {
93          return -1;
94      }
95      memset (pModul->pChannels,0,sizeof(TStjWAVFile)
          * number);
96      pModul->number = number;
97
98      int          i;
99      SF_INFO      info;
100      TStjWAVFile * pWF = pModul->pChannels;
101      size_t        cacheSize;
102
103      for (i = 0;i < number;i++) {
104          memset (&info,0,sizeof(SF_INFO));
105          if (pWAVFiles[i].isInput) {
106              pWF->pSndF = sf_open (pWAVFiles[i].szFileName,SFM_READ, &info
                  ) ;
107          } else {
108              info.samplerate = pWAVFiles[i].
                  sampleRate;
109              info.channels = 1;
110              info.format = SF_FORMAT_WAV |
                  SF_FORMAT_FLOAT;
111              pWF->pSndF = sf_open (pWAVFiles[i].szFileName,SFM_WRITE, &
                  info) ;
112          }
113          if (!pWF->pSndF) {
114              goto error;
115          }
116          pWF->isInput = (unsigned int) pWAVFiles[i].isInput;
117          pWF->channel = (unsigned int) pWAVFiles[i].channelNumber;
118          pWF->channelsAmount = (unsigned int)
                  info.channels;
119          pWF->dataOffset = (unsigned int)
                  pWAVFiles[i].fileChannelNumber;
120          pWF->maxSampleAmount = (unsigned int)
                  pWAVFiles[i].sampleAmountCacheSize;
121          cacheSize = pWF->channelsAmount * pWF->
                  maxSampleAmount * sizeof(float);
122          pWF->pCache = malloc (cacheSize);
123          if (!pWF->pCache) goto error;
124          memset (pWF->pCache,0,cacheSize);
125          pWF->sampleAmount = 0;

```

```

126         // now fill the cache with data
127         if (pWF->isInput) {
128             if (WAVmoduleFillCache(pWF))
129                 goto error;
130             } else {
131                 pWF->pD = pWF->pCache;
132             }
133         pWF++;
134     }
135     return 0;
136 error:
137     WAVmoduleExit(pModul);
138     return -2;
139 }
140
141 //! looking for the right channel
142 TStjWAVFile * WAVmoduleFindChannel (
143     TStjWAVmodule * pM,                //!< the
144                                         module
145     int channel                        //!< channel to search for
146 ) {
147     TStjWAVFile * pWF = pM->pChannels;
148     int i;
149     for (i = 0; i < pM->number;i++) {
150         if (pWF->channel == channel) return pWF;
151         pWF++;
152     }
153     return NULL;
154 }
155
156 //! reads a sample from a channel
157 int WAVmoduleGetInput (
158     TStjWAVmodule * pM,                //!<(in) the module
159     int channel,                       //!<(in) the channel
160     float * pSample                    //!<(out) pointer to the sample
161                                     //!< buffer witch is filled
162 ) {
163     TStjWAVFile * pWF = WAVmoduleFindChannel(pM,
164         channel);
165     // channel not found
166     if (!pWF) {
167         return -1;
168     }
169     // channel is not an input
170     if (!pWF->isInput) {
171         return -2;
172     }
173     // if the cache is empty then fill it
174     if (!pWF->sampleAmount) {
175         if (WAVmoduleFillCache(pWF)) return -3;
176     }

```

```

173         // copy sample
174         *pSample = pWF->pD[pWF->dataOffset];
175         pWF->pD += pWF->channelsAmount;
176         // now there is one sample less at the buffer
177         pWF->sampleAmount--;
178         return 0;
179     }
180
181     //! writes a sample to an output
182     int WAVmoduleSetOutput (
183         TStjWAVmodule * pM,
184         int channel,
185         float sample,
186         //!<(in) the module
187         //!<(in) the channel
188         //!<(in) pointer to the sample buffer
189         witch is filled
190     ) {
191         TStjWAVFile * pWF = WAVmoduleFindChannel(pM,
192             channel);
193         // channel not found
194         if (!pWF) {
195             return -1;
196         }
197         // channel is not an output
198         if (pWF->isInput) {
199             return -2;
200         }
201         // if the cache is full then flush it
202         if (pWF->sampleAmount == pWF->maxSampleAmount) {
203             if (WAVmoduleFlushCache(pWF)) return -3;
204         }
205         // now store the value
206         pWF->pD[pWF->dataOffset] = sample;
207         pWF->pD += pWF->channelsAmount;
208         pWF->sampleAmount++;
209         return 0;
210     }

```

3.1.14 pthreads and semaphores

includes:

c-Include	c-Library	system lib
pthread.h	pthread	yes
semaphore.h		yes

code:

3.1.15 rational and integer generic

includes:

c-Include	c-Library	system lib
stdlib.h		yes
stdint.h		yes
math.h	m	yes

code:

```
1 // =====
2 // generic vector interface for integer and
3 // rational types based on the std. C lib
4 // =====
5
6 // -----
7 // generic integer type
8 // -----
9
10 // transforms the integer to a 'unique form'
11 inline uint32_t APgenericInteger_ToUniform (uint32_t v)
12 {
13     uint32_t u;
14     u = v & 0xFF;
15     u <<= 8;
16     v >>= 8;
17     u |= (v & 0xFF);
18     u <<= 8;
19     v >>= 8;
20     u |= (v & 0xFF);
21     u <<= 8;
22     v >>= 8;
23     u |= (v & 0xFF);
24     return u;
25 }
26
27 // inverse transformation of the unique integer to the '
28 // local form'
29 inline uint32_t APgenericInteger_FromUniform (uint32_t v)
30 {
31     uint32_t u;
32     u = v & 0xFF;
33     u <<= 8;
34     v >>= 8;
35     u |= v & 0xFF;
36     u <<= 8;
37     v >>= 8;
38     u |= v & 0xFF;
39     u <<= 8;
40     v >>= 8;
41     u |= v & 0xFF;
42     return u;
43 }
```

```

41
42
43
44 // type for a generic vector based int type
45 typedef struct SAPgenericIntegerVector {
46     int32_t *      pVal;
47     int            num;
48 } TAPgenericIntegerVector;
49
50 // create int vector
51 TAPgenericIntegerVector * APgenericIntegerVector_create
52     (int num) {
53     int i;
54     // alloc memory
55     TAPgenericIntegerVector * pI = malloc (sizeof(
56         TAPgenericIntegerVector));
57     if (!pI) return NULL;
58     pI->pVal = malloc (sizeof(int32_t)*num);
59     if (!pI->pVal) {
60         free (pI);
61         return NULL;
62     }
63     pI->num = num;
64     // set memory
65     for (i = 0; i < num; i++) {
66         pI->pVal[i] = 0;
67     }
68     return pI;
69 }
70
71 // destroy int vector
72 void APgenericIntegerVector_destroy (
73     TAPgenericIntegerVector * pV) {
74     if (pV) {
75         free(pV->pVal);
76         free(pV);
77     }
78 }
79
80 // resize int vector
81 int APgenericIntegerVector_resize (
82     TAPgenericIntegerVector * pV, int newNum) {
83     int i;
84
85     if (pV->num == newNum) return 0;
86     free (pV->pVal);
87     pV->pVal = malloc (sizeof(int32_t)*newNum);
88     if (!pV->pVal) return -1;
89     pV->num = newNum;
90     // set memory
91     for (i = 0; i < newNum; i++) {
92         pV->pVal[i] = 0;
93     }
94     return 0;
95 }

```

```

92
93 // assign a = values
94 int APgenericIntegerVector_assignConst (
    TAPgenericIntegerVector * pa, int iStart, int num,
    int32_t * pVals) {
95     int i = iStart;
96     int imax = i + num;
97     int32_t * pD = pa->pVal + iStart;
98     if (imax > pa->num) imax = pa->num;
99
100     for (; i < imax; i++) {
101         *pD = *pVals;
102         pD++;
103         pVals++;
104     }
105     return 0;
106 }
107
108 // assign a = b
109 int APgenericIntegerVector_assign (
    TAPgenericIntegerVector * pa, TAPgenericIntegerVector
    * pb) {
110     int imax = pa->num;
111     int i;
112     for (i = 0; i < imax; i++) {
113         pa->pVal[i] = pb->pVal[i];
114     }
115     return 0;
116 }
117
118
119 // add c = a + b
120 int APgenericIntegerVector_add (TAPgenericIntegerVector
    * pa, TAPgenericIntegerVector * pb,
    TAPgenericIntegerVector * pc) {
121     int imax = pc->num;
122     int i;
123     for (i = 0; i < imax; i++) {
124         pc->pVal[i] = pa->pVal[i] + pb->pVal[i];
125     }
126     return 0;
127 }
128
129 // sub c = a - b
130 int APgenericIntegerVector_sub (TAPgenericIntegerVector
    * pa, TAPgenericIntegerVector * pb,
    TAPgenericIntegerVector * pc) {
131     int imax = pc->num;
132     int i;
133     for (i = 0; i < imax; i++) {
134         pc->pVal[i] = pa->pVal[i] - pb->pVal[i];
135     }
136     return 0;
137 }
138

```



```

139 // mul c = a * b
140 int APgenericIntegerVector_mul (TAPgenericIntegerVector
    * pa, TAPgenericIntegerVector * pb,
    TAPgenericIntegerVector * pc) {
141     int imax = pc->num;
142     int i;
143     for (i = 0; i < imax; i++) {
144         pc->pVal[i] = pa->pVal[i] * pb->pVal[i];
145     }
146     return 0;
147 }
148
149 // div c = a * b
150 int APgenericIntegerVector_div (TAPgenericIntegerVector
    * pa, TAPgenericIntegerVector * pb,
    TAPgenericIntegerVector * pc) {
151     int imax = pc->num;
152     int i;
153     for (i = 0; i < imax; i++) {
154         pc->pVal[i] = pa->pVal[i] / pb->pVal[i];
155     }
156     return 0;
157 }
158
159 // modulo c = a mod b
160 int APgenericIntegerVector_mod (TAPgenericIntegerVector
    * pa, TAPgenericIntegerVector * pb,
    TAPgenericIntegerVector * pc) {
161     int imax = pc->num;
162     int i;
163     for (i = 0; i < imax; i++) {
164         pc->pVal[i] = pa->pVal[i] % pb->pVal[i];
165     }
166     return 0;
167 }
168
169 // cmp Element cmp(a[i],b[j]):
170 //         res = -1: a < b
171 //         res = 1 : a > b
172 //         res = 0: a = b
173 int APgenericIntegerVector_cmpEle (
    TAPgenericIntegerVector * pa, TAPgenericIntegerVector
    * pb, int ia, int ib) {
174     int zwv = pa->pVal[ia] - pb->pVal[ib];
175     if (!zwv) return 0;
176     if (zwv < 0) return -1;
177     return 1;
178 }
179
180
181 // -----
182 // generic rational type
183 // -----
184
185 // type for a generic vector based float type

```

```

186 typedef struct SAPgenericRationalVector {
187     float *      pVal;
188     int          num;
189 } TAPgenericRationalVector;
190
191 // create float vector
192 TAPgenericRationalVector *
193     APgenericRationalVector_create (int num) {
194     int i;
195     // alloc memory
196     TAPgenericRationalVector * pR = malloc (sizeof(
197         TAPgenericRationalVector));
198     if (!pR) return NULL;
199     pR->pVal = malloc (sizeof(float)*num);
200     if (!pR->pVal) {
201         free (pR);
202         return NULL;
203     }
204     pR->num = num;
205     // set memory
206     for (i = 0; i < num; i++) {
207         pR->pVal[i] = 0.f;
208     }
209     return pR;
210 }
211
212 // destroy float vector
213 void APgenericRationalVector_destroy (
214     TAPgenericRationalVector * pV) {
215     if (pV) {
216         free(pV->pVal);
217         free(pV);
218     }
219 }
220
221 // resize float vector
222 int APgenericRationalVector_resize (
223     TAPgenericRationalVector * pV, int newNum) {
224     int i;
225
226     if (pV->num == newNum) return 0;
227     free (pV->pVal);
228     pV->pVal = malloc (sizeof(float)*newNum);
229     if (!pV->pVal) return -1;
230     pV->num = newNum;
231     // set memory
232     for (i = 0; i < newNum; i++) {
233         pV->pVal[i] = 0.f;
234     }
235     return 0;
236 }
237
238 // assign a = values
239 int APgenericRationalVector_assignConst (
240     TAPgenericRationalVector * pa, int iStart, int num,

```

```

236     float * pVals) {
237         int i = iStart;
238         int imax = i + num;
239         float * pD = pa->pVal + iStart;
240         if (imax > pa->num) imax = pa->num;
241
242         for (; i < imax; i++) {
243             *pD = *pVals;
244             pD++;
245             pVals++;
246         }
247         return 0;
248     }
249
250 // assign a = b
251 int APgenericRationalVector_assign (
252     TAPgenericRationalVector * pa,
253     TAPgenericRationalVector * pb) {
254     int imax = pa->num;
255     int i;
256     for (i = 0; i < imax; i++) {
257         pa->pVal[i] = pb->pVal[i];
258     }
259     return 0;
260 }
261
262 // add c = a + b
263 int APgenericRationalVector_add (
264     TAPgenericRationalVector * pa,
265     TAPgenericRationalVector * pb,
266     TAPgenericRationalVector * pc) {
267     int imax = pc->num;
268     int i;
269     for (i = 0; i < imax; i++) {
270         pc->pVal[i] = pa->pVal[i] + pb->pVal[i];
271     }
272     return 0;
273 }
274
275 // sub c = a - b
276 int APgenericRationalVector_sub (
277     TAPgenericRationalVector * pa,
278     TAPgenericRationalVector * pb,
279     TAPgenericRationalVector * pc) {
280     int imax = pc->num;
281     int i;
282     for (i = 0; i < imax; i++) {
283         pc->pVal[i] = pa->pVal[i] - pb->pVal[i];
284     }
285     return 0;
286 }
287
288 // mul c = a * b
289 int APgenericRationalVector_mul (

```

```

TAPgenericRationalVector * pa,
TAPgenericRationalVector * pb,
TAPgenericRationalVector * pc) {
282     int imax = pc->num;
283     int i;
284     for (i = 0; i < imax; i++) {
285         pc->pVal[i] = pa->pVal[i] * pb->pVal[i];
286     }
287     return 0;
288 }
289
290 // div c = a * b
291 int APgenericRationalVector_div (
    TAPgenericRationalVector * pa,
    TAPgenericRationalVector * pb,
    TAPgenericRationalVector * pc) {
292     int imax = pc->num;
293     int i;
294     for (i = 0; i < imax; i++) {
295         pc->pVal[i] = pa->pVal[i] / pb->pVal[i];
296     }
297     return 0;
298 }
299
300 // modulo c = a mod b
301 int APgenericRationalVector_mod (
    TAPgenericRationalVector * pa,
    TAPgenericRationalVector * pb,
    TAPgenericRationalVector * pc) {
302     int imax = pc->num;
303     int i;
304     for (i = 0; i < imax; i++) {
305         pc->pVal[i] = fmodf(pa->pVal[i],pb->pVal
            [i]);
306     }
307     return 0;
308 }
309
310 // cmp Element cmp(a[i],b[j]):
311 //     res = -1: a < b
312 //     res = 1 : a > b
313 //     res = 0: a = b
314 int APgenericRationalVector_cmpEle (
    TAPgenericRationalVector * pa,
    TAPgenericRationalVector * pb, int ia, int ib) {
315     float zwv = pa->pVal[ia] - pb->pVal[ib];
316     if (zwv == 0.f) return 0;
317     if (zwv < 0.f) return -1;
318     return 1;
319 }

```

3.1.16 no group

3.2 Implementation of the HAL Variables

3.2.1 rational (rational and integer generic)

Informations:

variable type id:	1
variable type name:	rational
group:	rational and integer generic
description:	super generic rational

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```
1 // =====
2 // variable implementation for a rational number(var id
  = 1)
3 // =====
4
5 // updates a variable the AP
6 int HALimpl_1_recvUpdate (void * pVarData, void *
  pMsgData) {
7     uint32_t * pD = (uint32_t *) pMsgData;
8     TAPgenericRationalVector * pIV = pVarData;
9
10    // at the first position at the message is the
      global var index
11    pD++; // skip it (it's the varindex)
12    // and now we are at amount of values
13    APendianConversation32Bit(pD,eAP_littleEndian);
14    int imax = (int) *((int32_t *)pD);
15    int i;
16
17    if (APgenericRationalVector_resize(pIV,imax)) {
18        return -1;
19    }
20
21    pD++;
22
23    for (i = 0; i < imax; i++) {
24        pIV->pVal[i] = *((float *)pD);
25        pD++;
26    }
27    return 0;
28 }
29 // create a new variable
30 void * HALimpl_1_create (unsigned int numberOfElements)
```

```

31     {
32         return APgenericRationalVector_create((int)
33             numberOfElements);
34     }
35     // updates the vars at the other APs
36     int HALimpl_1_sendUpdate (void * pVarData, const void *
37         pDrv, uint32_t receiver, uint32_t mNum, int32_t i) {
38         TAPgenericRationalVector * pRV = pVarData;
39         TAPmsgDrv * pMD = (TAPmsgDrv *) pDrv;
40         int32_t dataAmount = (int32_t) 1 + pRV->num;
41         int32_t dummy;
42         // send header
43         pMD->pfkt_updateVariable(pMD->pDrvData, receiver
44             , mNum, i, dataAmount);
45         // send amount of elements
46         dummy = (int32_t) pRV->num;
47         pMD->pfkt_sendInteger32(pMD, 1, &dummy);
48         // send elements
49         pMD->pfkt_sendFloat32(pMD, pRV->num, pRV->pVal);
50         return 0;
51     }
52     // decode data for the HAL functions
53     void * HALimpl_1_decodeData (void * pVarData) {
54         return pVarData;
55     }
56     // delete the variable
57     void HALimpl_1_delete (void * pVarData) {
58         APgenericRationalVector_destroy(pVarData);
59     }

```

3.2.2 integer (rational and integer generic)

Informations:

variable type id:	2
variable type name:	integer
group:	rational and integer generic
description:	super generic integer

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====
2 // variable implementation for a integer number(var id =
3 // 2)
4 // =====
5 // updates a variable the AP

```

```

6  int HALimpl_2_recvUpdate (void * pVarData, void *
    pMsgData) {
7      uint32_t * pD = (uint32_t *) pMsgData;
8      TAPgenericIntegerVector * pIV = pVarData;
9
10     // at the first position at the message is the
        global var index
11     pD++; // skip it (it's the varindex)
12     // and now we are at amount of values
13     APendianConversation32Bit(pD,eAP_littleEndian);
14     int imax = (int) *((int32_t *)pD);
15
16     int i;
17
18     if (APgenericIntegerVector_resize(pIV,imax)) {
19         return -1;
20     }
21
22     pD++;
23
24     for (i = 0; i < imax; i++) {
25         APendianConversation32Bit(pD,
            eAP_littleEndian);
26         pIV->pVal[i] = *((int32_t *)pD);
27         pD++;
28     }
29     return 0;
30 }
31
32 // create a new variable
33 void * HALimpl_2_create (unsigned int numberOfElements)
    {
34     return APgenericIntegerVector_create((int)
        numberOfElements);
35 }
36 // updates the vars at the other APs
37 int HALimpl_2_sendUpdate (void * pVarData, const void *
    pDrv, uint32_t receiver, uint32_t mNum, int32_t i) {
38     TAPgenericIntegerVector * pIV = pVarData;
39     TAPMsgDrv * pMD = (TAPMsgDrv *) pDrv;
40     int32_t dataAmount = 1 + pIV->num;
41     int32_t dummy;
42     int indx;
43     // send header
44     pMD->pfkt_updateVariable(pMD->pDrvData, receiver
        , mNum, i, dataAmount);
45     // send amount of elements
46     dummy = (int32_t) pIV->num;
47     pMD->pfkt_sendInteger32(pMD,1,&dummy);
48     // send elements
49     for (indx = 0; indx < pIV->num; indx++) {
50         pMD->pfkt_sendInteger32(pMD,1,&pIV->pVal
            [indx]);
51     }
52     return 0;

```

```

53 }
54 // decode data for the HAL functions
55 void * HALimpl_2_decodeData (void * pVarData) {
56     return pVarData;
57 }
58 // delete the variable
59 void HALimpl_2_delete (void * pVarData) {
60     APgenericIntegerVector_destroy(pVarData);
61 }

```

3.2.3 complex (fftw3 & complex)

Informations:

variable type id:	4
variable type name:	complex
group:	fftw3 & complex
description:	complex number used/defined at the fftw3 library

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====
2 // variable implementation for a complex number(var id =
  4)
3 // =====
4
5 // the type for the complex data struct
6 typedef struct SAPvarComplex {
7     fftwf_complex * pC;
8     int num;
9 } TAPvarComplex;
10
11 // transmit the data
12 void HALimpl_4_init (void * pVarData, int frameNumber,
    void * pData) {
13
14 }
15
16 // start the init process
17 void HALimpl_4_initStart (void * pVarData, int
    frameNumber, int bigEndian) {
18
19 }
20
21 // get the frame Number and size
22 int HALimpl_4_frameGetNumber (void * pVarData, int *
    pframeSize) {
23     TAPvarComplex * pC = (TAPvarComplex *) pVarData;

```



```

24         *pframeSize = pC->num * sizeof(fftwf_complex);
25         return 1;
26     }
27
28     // end init process
29     void HALimpl_4_initEnd (void * pVarData) {
30
31     }
32
33     // create a new variable
34     void * HALimpl_4_create (unsigned int numberOfElements)
35     {
36         if (!numberOfElements) return NULL;
37
38         TAPvarComplex * pC;
39         pC = malloc(sizeof(TAPvarComplex));
40         if (!pC) return NULL;
41
42         pC->pC = fftwf_malloc(sizeof(fftwf_complex)*
43             numberOfElements);
44         if (!(pC->pC)) {
45             free (pC);
46             return NULL;
47         }
48         pC->num = numberOfElements;
49
50         memset (pC->pC,0,sizeof(fftwf_complex)*
51             numberOfElements);
52         return pC;
53     }
54
55     // fill the frame with data
56     void HALimpl_4_frameFill (void * pVarData, int
57         frameNumber, void * pFrame) {
58
59     }
60
61     // decode data for the HAL functions
62     void * HALimpl_4_decodeData (void * pVarData) {
63         TAPvarComplex * pC = (TAPvarComplex *) pVarData;
64         return pC->pC;
65     }
66
67     // delete the variable
68     void HALimpl_4_delete (void * pVarData) {
69         TAPvarComplex * pC = (TAPvarComplex *) pVarData;
70         if (pC) {
71             if (pC->pC) fftwf_free(pC->pC);
72             free (pC);
73         }
74     }

```

3.2.4 FFT (fftw3 & complex)

Informations:

variable type id:	30
variable type name:	FFT
group:	fftw3 & complex
description:	FFT / IFFT structure for performing ffts and iffts

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```
1 // =====
2 // variable implementation for FFT or IFFT sturcture(var
   id = 30)
3 // =====
4
5 // transmit the data
6 void HALimpl_30_init (void * pVarData, int frameNumber,
   void * pData) {
7
8 }
9 // start the init process
10 void HALimpl_30_initStart (void * pVarData, int
   frameNumber, int bigEndian) {
11
12 }
13 // get the frame Number and size
14 int HALimpl_30_frameGetNumber (void * pVarData, int *
   pframeSize) {
15 return 0;
16 }
17 // end init process
18 void HALimpl_30_initEnd (void * pVarData) {
19
20 }
21 // create a new variable
22 void * HALimpl_30_create (unsigned int numberOfElements)
   {
23     return NULL;
24 }
25 // fill the frame with data
26 void HALimpl_30_frameFill (void * pVarData, int
   frameNumber, void * pFrame) {
27
28 }
29 // decode data for the HAL functions
30 void * HALimpl_30_decodeData (void * pVarData) {
31     return pVarData;
32 }
33 // delete the variable
```

```

34 void HALimpl_30_delete (void * pVarData) {
35     if (pVarData) {
36         fftwf_destroy_plan(pVarData);
37     }
38 }

```

3.2.5 panel (MSP430-169STK)

Informations:

variable type id:	100
variable type name:	panel
group:	MSP430-169STK
description:	panel type for MSP430-169STK eval board

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1  // =====
2  // variable implementation for panel UI(var id = 100)
3  // =====
4
5  typedef struct SMSP430_panel {
6      unsigned char dummy;
7  } TMSP430_panel;
8
9  typedef struct SMSP430_panelVec {
10     TMSP430_panel * pP;
11     int num;
12 } TMSP430_panelVec;
13
14 // updates a variable the AP
15 int HALimpl_100_recvUpdate (void * pVarData, void *
    pMsgData) {
16     return -1;
17 }
18 // create a new variable
19 void * HALimpl_100_create (unsigned int numberOfElements
    ) {
20     TMSP430_panelVec * pPV;
21
22     pPV = malloc(sizeof(TMSP430_panelVec));
23     if (!pPV) return NULL;
24
25     pPV->pP = malloc(sizeof(TMSP430_panel)*
        numberOfElements);
26     if (!pPV->pP) {
27         free(pPV);
28         return NULL;

```

```

29         }
30         pPV->num = (int) numberOfElements;
31         return pPV;
32     }
33     // updates the vars at the other APs
34     int HALimpl_100_sendUpdate (void * pVarData, const void
        * pDrv, uint32_t receiver, uint32_t mNum, int32_t i)
    {
35     return -1;
36     }
37     // decode data for the HAL functions
38     void * HALimpl_100_decodeData (void * pVarData) {
39         return pVarData;
40     }
41     // delete the variable
42     void HALimpl_100_delete (void * pVarData) {
43         TMSP430_panelVec * pPV = (TMSP430_panelVec *)
            pVarData;
44         if (pPV) {
45             if (pPV->pP) free(pPV->pP);
46             free (pPV);
47         }
48     }

```

3.2.6 button (MSP430-169STK)

Informations:

variable type id:	101
variable type name:	button
group:	MSP430-169STK
description:	button type for MSP430-169STK eval board

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====
2 // variable implementation for button UI(var id = 101)
3 // =====
4
5 typedef struct SMSP430_button {
6     unsigned char port1Mask;
7 } TMSP430_button;
8
9 typedef struct SMSP430_buttonVec {
10     TMSP430_button * pB;
11     int num;
12 } TMSP430_buttonVec;
13

```

```

14
15 // updates a variable the AP
16 int HALimpl_101_recvUpdate (void * pVarData, void *
    pMsgData) {
17     return -1;
18 }
19 // create a new variable
20 void * HALimpl_101_create (unsigned int numberOfElements
    ) {
21     TMSP430_buttonVec * pBV;
22
23     pBV = malloc(sizeof(TMSP430_buttonVec));
24     if (!pBV) return NULL;
25
26     pBV->pB = malloc(sizeof(TMSP430_button)*
        numberOfElements);
27     if (!pBV->pB) {
28         free(pBV);
29         return NULL;
30     }
31
32     pBV->num = (int) numberOfElements;
33     memset (pBV->pB, 0, sizeof(TMSP430_button)*
        numberOfElements);
34     return pBV;
35 }
36 // updates the vars at the other APs
37 int HALimpl_101_sendUpdate (void * pVarData, const void
    * pDrv, uint32_t receiver, uint32_t mNum, int32_t i)
    {
38     return -1;
39 }
40 // decode data for the HAL functions
41 void * HALimpl_101_decodeData (void * pVarData) {
42     return pVarData;
43 }
44 // delete the variable
45 void HALimpl_101_delete (void * pVarData) {
46     TMSP430_buttonVec * pBV = (TMSP430_buttonVec *)
        pVarData;
47     if (pBV) {
48         if (pBV->pB) free(pBV->pB);
49         free (pBV);
50     }
51 }

```

3.2.7 led (MSP430-169STK)

Informations:

variable type id:	102
variable type name:	led
group:	MSP430-169STK
description:	LED type for MSP430-169STK eval board

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```
1 // =====
2 // variable implementation for led UI(var id = 102)
3 // =====
4
5 typedef struct SMSP430_LED {
6     unsigned char port3mask;
7 } TMSP430_LED;
8
9 typedef struct SMSP430_LEDvec {
10     TMSP430_LED * pL;
11     int num;
12 } TMSP430_LEDvec;
13
14 // updates a variable the AP
15 int HALimpl_102_recvUpdate (void * pVarData, void *
16     pMsgData) {
17     return -1;
18 }
19 // create a new variable
20 void * HALimpl_102_create (unsigned int numberOfElements
21     ) {
22     TMSP430_LEDvec * pLV;
23
24     pLV = malloc(sizeof(TMSP430_LEDvec));
25     if (!pLV) return NULL;
26
27     pLV->pL = malloc(sizeof(TMSP430_LED)*
28         numberOfElements);
29     if (!pLV->pL) {
30         free(pLV);
31         return NULL;
32     }
33
34     pLV->num = (int) numberOfElements;
35     memset (pLV->pL, 0, sizeof(TMSP430_LED)*
36         numberOfElements);
37     return pLV;
38 }
39 // updates the vars at the other APs
```

```

36 int HALimpl_102_sendUpdate (void * pVarData, const void
    * pDrv, uint32_t receiver, uint32_t mNum, int32_t i)
    {
37 return -1;
38 }
39 // decode data for the HAL functions
40 void * HALimpl_102_decodeData (void * pVarData) {
41 return pVarData;
42 }
43 // delete the variable
44 void HALimpl_102_delete (void * pVarData) {
45     TMSP430_LEDvec * pLV = (TMSP430_LEDvec *)
        pVarData;
46     if (pLV) {
47         if (pLV->pL) free(pLV->pL);
48         free (pLV);
49     }
50 }
51 }

```

3.2.8 display (MSP430-169STK)

Informations:

variable type id:	103
variable type name:	display
group:	MSP430-169STK
description:	display type for MSP430-169STK eval board

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====
2 // variable implementation for display UI(var id = 103)
3 // =====
4
5 typedef struct SMSP430_display {
6     unsigned char dummy;
7 } TMSP430_display;
8
9 typedef struct SMSP430_displayVec {
10     TMSP430_display * pD;
11     int num;
12 } TMSP430_displayVec;
13
14 // updates a variable the AP
15 int HALimpl_103_rcvUpdate (void * pVarData, void *
    pMsgData) {
16 return -1;

```

```

17 }
18 // create a new variable
19 void * HALimpl_103_create (unsigned int numberOfElements
20 ) {
21     TMSP430_displayVec * pDV;
22
23     pDV = malloc(sizeof(TMSP430_displayVec));
24     if (!pDV) return NULL;
25
26     pDV->pD = malloc(sizeof(TMSP430_display)*
27         numberOfElements);
28     if (!pDV->pD) {
29         free(pDV);
30         return NULL;
31     }
32
33     pDV->num = (int) numberOfElements;
34     memset (pDV->pD, 0, sizeof(TMSP430_display)*
35         numberOfElements);
36     return pDV;
37 }
38 // updates the vars at the other APs
39 int HALimpl_103_sendUpdate (void * pVarData, const void
40 * pDrv, uint32_t receiver, uint32_t mNum, int32_t i)
41 {
42     return -1;
43 }
44 // decode data for the HAL functions
45 void * HALimpl_103_decodeData (void * pVarData) {
46     return pVarData;
47 }
48 // delete the variable
49 void HALimpl_103_delete (void * pVarData) {
50     TMSP430_displayVec * pDV = (TMSP430_displayVec
51         *) pVarData;
52     if (pDV) {
53         if (pDV->pD) free(pDV->pD);
54         free (pDV);
55     }
56 }

```

3.2.9 panel (gtk+ for Windows)

Informations:

variable type id:	100
variable type name:	panel
group:	gtk+ for Windows
description:	panel type in gtk+ style

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```
1 // =====
2 // variable implementation for panel UI(var id = 100)
3 // =====
4
5 // updates a variable the AP
6 int HALimpl_100_recvUpdate (void * pVarData, void * pMsg
7     ) {
8     return -1;
9 }
10 // create a new variable
11 void * HALimpl_100_create (unsigned int numberOfElements
12     ) {
13     return APgtkUI_createVector(numberOfElements,
14         eAPgtkUItype_panel);
15 }
16 // updates the vars at the other APs
17 int HALimpl_100_sendUpdate (void * pVarData, const void
18     * pDrv, uint32_t receiver, uint32_t mNum, int32_t i)
19 {
20     return -1;
21 }
22 // decode data for the HAL functions
23 void * HALimpl_100_decodeData (void * pVarData) {
24     return pVarData;
25 }
26 // delete the variable
27 void HALimpl_100_delete (void * pVarData) {
28     APgtkUI_destroyVector(pVarData);
29 }
30 }
```

3.2.10 button (gtk+ for Windows)

Informations:

variable type id:	101
variable type name:	button
group:	gtk+ for Windows
description:	button type in gtk+ style

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```
1 // =====
2 // variable implementation for button UI(var id = 101)
```

```

3 // =====
4
5 // updates a variable the AP
6 int HALimpl_101_recvUpdate (void * pVarData, void * pMsg
7 ) {
8     return -1;
9 }
10 // create a new variable
11 void * HALimpl_101_create (unsigned int numberOfElements
12 ) {
13     return APgtkUI_createVector(numberOfElements,
14                                 eAPgtkUI_type_button);
15 }
16 // updates the vars at the other APs
17 int HALimpl_101_sendUpdate (void * pVarData, const void
18 * pDrv, uint32_t receiver, uint32_t mNum, int32_t i)
19 {
20     return -1;
21 }
22 // decode data for the HAL functions
23 void * HALimpl_101_decodeData (void * pVarData) {
24     return pVarData;
25 }
26 // delete the variable
27 void HALimpl_101_delete (void * pVarData) {
28     APgtkUI_destroyVector(pVarData);
29 }

```

3.2.11 led (gtk+ for Windows)

Informations:

variable type id:	102
variable type name:	led
group:	gtk+ for Windows
description:	LED type in gtk+ style

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====
2 // variable implementation for led UI(var id = 102)
3 // =====
4
5 // updates a variable the AP
6 int HALimpl_102_recvUpdate (void * pVarData, void * pMsg
7 ) {
8     return -1;
9 }

```

```

8 }
9 // create a new variable
10 void * HALimpl_102_create (unsigned int numberOfElements
    ) {
11     return APgtkUI_createVector(numberOfElements,
        eAPgtkUItype_LED);
12 }
13 // updates the vars at the other APs
14 int HALimpl_102_sendUpdate (void * pVarData, const void
    * pDrv, uint32_t receiver, uint32_t mNum, int32_t i)
    {
15 return -1;
16 }
17 // decode data for the HAL functions
18 void * HALimpl_102_decodeData (void * pVarData) {
19     return pVarData;
20 }
21 // delete the variable
22 void HALimpl_102_delete (void * pVarData) {
23     APgtkUI_destroyVector(pVarData);
24 }

```

3.2.12 display (gtk+ for Windows)

Informations:

variable type id:	103
variable type name:	display
group:	gtk+ for Windows
description:	display type in gtk+ style

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====
2 // variable implementation for display UI(var id = 103)
3 // =====
4
5 // updates a variable the AP
6 int HALimpl_103_recvUpdate (void * pVarData, void * pMsg
    ) {
7 return -1;
8 }
9 // create a new variable
10 void * HALimpl_103_create (unsigned int numberOfElements
    ) {
11     return APgtkUI_createVector(numberOfElements,
        eAPgtkUItype_display);
12 }

```

```

13 // updates the vars at the other APs
14 int HALimpl_103_sendUpdate (void * pVarData, const void
    * pDrv, uint32_t receiver, uint32_t mNum, int32_t i)
    {
15 return -1;
16 }
17 // decode data for the HAL functions
18 void * HALimpl_103_decodeData (void * pVarData) {
19     return pVarData;
20 }
21 // delete the variable
22 void HALimpl_103_delete (void * pVarData) {
23     APgtkUI_destroyVector(pVarData);
24 }

```

3.2.13 string (ANSI C strings)

Informations:

variable type id:	3
variable type name:	string
group:	ANSI C strings
description:	string based on stdlib

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====
2 // variable implementation for a string(var id = 3)
3 // =====
4
5 // updates a variable the AP
6 int HALimpl_3_recvUpdate (void * pVarData, void * pMsg)
    {
7     int32_t *                pD = (int32_t *)
        pMsg;
8     TAPstringVector *        pSV = pVarData;
9     TAPstring *              ps;
10    int                    i, num,
        imax;
11
12    // at the first possion at the message is the
        global var index
13    pD++; // skip it (it's the varindex)
14    //1. get amount of strings at the string vector
15    APendianConversation32Bit((uint32_t *)pD,
        eAP_littleEndian);
16    num = (int) *pD;
17

```

```

18     if (APstringVector_resizeVector(pSV,num)) {
19         return -1;
20     }
21
22     pD++;
23     ps = pSV->sv;
24     //2. resize and fill the strings
25     for (i = 0; i < num; i++) {
26         // get length
27         APendianConversation32Bit((uint32_t *)pD
28             ,eAP_littleEndian);
29         if (APstringVector_resize(pSV,i,(size_t)
30             *pD)) {
31             return -2;
32         }
33         imax = (int) *pD;
34         pD++;
35         for (i = 0; i < imax; i++) {
36             ps->szTxt[i] = (char) *pD;
37             pD++;
38         }
39         ps->szTxt[i] = 0;
40         ps++;
41     }
42     return 0;
43 }
44 // create a new variable
45 void * HALimpl_3_create (unsigned int numberOfElements)
46 {
47     return APstringVector_create((int)
48         numberOfElements);
49 }
50 // updates the vars at the other APs
51 int HALimpl_3_sendUpdate (void * pVarData, const void *
52     pDrv, uint32_t receiver, uint32_t mNum, int32_t i) {
53     // msgdata = (stringAmount)[(strlen)(string +
54     // fill bytes), ...]
55     // some vars
56     unsigned int          indx;
57     int                  stl;
58     TAPstring *          ps;
59     TAPstringVector *    pSV = (TAPstringVector
60         *)pVarData;
61     TAPMsgDrv *          pMD = (TAPMsgDrv *) pDrv
62         ;
63     int32_t              uint32Amount =
64         1; // vector size
65     int32_t              sendVal;
66     char *               psz;
67     // ok get the size of the message
68     ps = pSV->sv;
69     for (indx = 0; indx < pSV->num; indx++) {
70         // one element for the length
71         uint32Amount++;
72         // the string data

```

```

64         stl = ps->szTxt ? strlen(ps->szTxt) : 0;
65         uint32Amount+=stl;
66         // next string
67         ps++;
68     }
69     // send header
70     pMD->pfkt_updateVariable(pMD->pDrvData, receiver
71         , mNum, i, uint32Amount);
72     // send amount of elements
73     sendVal = (int32_t) pSV->num;
74     pMD->pfkt_sendInteger32(pMD,1,&sendVal);
75     ps = pSV->sv;
76     for (indx = 0; indx < pSV->num; indx++) {
77         if (ps->szTxt) {
78             // get length
79             sendVal = (int32_t) strlen(ps->
80                 szTxt);
81             // send length
82             pMD->pfkt_sendInteger32(pMD,1,&
83                 sendVal);
84             // send string
85             psz = ps->szTxt;
86             while (*psz) {
87                 sendVal = (int32_t) *psz
88                     ;
89                 pMD->pfkt_sendInteger32(
90                     pMD,1,&sendVal);
91                 psz++;
92             }
93         } else {
94             sendVal = 0;
95             // send length
96             pMD->pfkt_sendInteger32(pMD,1,&
97                 sendVal);
98         }
99         ps++;
100     }
101     return 0;
102 }
103 // decode data for the HAL functions
104 void * HALimpl_3_decodeData (void * pVarData) {
105     return pVarData;
106 }
107 // delete the variable
108 void HALimpl_3_delete (void * pVarData) {
109     APstringVector_free(pVarData);
110 }

```

3.2.14 biquad (biquad filters (generic))

Informations:

variable type id:	10
variable type name:	biquad
group:	biquad filters (generic)
description:	super generic biquad

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====
2 // variable implementation for a biquad filter(var id =
  10)
3 // =====
4
5 // updates a variable the AP
6 int HALimpl_10_recvUpdate (void * pVarData, void *
  pMsgData) {
7     uint32_t * pD = (uint32_t *) pMsgData;
8     TBQF_BiquadCascade * pV =(TBQF_BiquadCascade *)
      pVarData;
9     TBQF_BiquadDF2 * pE;
10    int i,j, imax;
11    float * pNum;
12    float * pDenom;
13
14
15    // at the first possion at the message is the
      global var index
16    pD++; // skip it (it's the varindex)
17    // and now we are at amount of values
18    if (gAPendianFlag != eAP_littleEndian) {
19        APendianConversation32Bit(pD,
      eAP_littleEndian);
20    }
21    imax = (int) *((int32_t *)pD);
22    pD++;
23    // 1. resize vector if needed
24    if (pV->num != imax) {
25        if (BQF_BQFcascadeCreate(pV,imax)) {
26            return -1;
27        }
28    }
29
30    // 2. fill parameters
31    pE = pV->pB;
32
33    for (i = 0; i < imax; i++) {
34        if (gAPendianFlag != eAP_littleEndian) {

```

```

35         for (j = 0; j < 6; j++) {
36             APendianConversation32Bit
                (pD+j,
                eAP_littleEndian);
37         }
38     }
39     // calc pointers
40     pNum = ((float *)pD);
41     pD += 3;
42     pDenom = ((float *)pD);
43     pD += 3;
44     //
45     BQF_BQFinitFromCoefficients(pV, i, pNum,
        pDenom);
46     pE++;
47 }
48 #ifdef dBQF_implementPrintFunctions
49     fprintf(stdout, "rx biquad:\n")
50     BQF_PrintBiquad(pV, stdout);
51     fflush(stdout);
52 #endif
53     return 0;
54 }
55 // create a new variable
56 void * HALimpl_10_create (unsigned int numberOfElements)
57 {
58     TBQF_BiquadCascade * pBC = malloc(sizeof(
        TBQF_BiquadCascade));
59     if (!pBC) return NULL;
60     pBC->num = 0;
61     pBC->pB = NULL;
62     if (BQF_BQFcreate(pBC, numberOfElements))
63     {
64         free(pBC);
65         return NULL;
66     }
67     return pBC;
68 }
69 // updates the vars at the other APs
70 int HALimpl_10_sendUpdate (void * pVarData, const void *
    pDrv, uint32_t receiver, uint32_t mNum, int32_t i) {
71     TBQF_BiquadCascade * pV =(TBQF_BiquadCascade *)
        pVarData;
72     TBQF_BiquadDF2 * pE;
73     TAPMsgDrv * pMD = (TAPMsgDrv *) pDrv;
74
75     int32_t dataAmount;
76     int32_t n;
77     float num[3];
78     float denom[3];
79
80     // calc amount of transmission bytes
81     dataAmount = (int32_t) 1 + pV->num * 6;

```



```

82
83 // 1 send header
84 pMD->pfkt_updateVariable(pMD->pDrvData, receiver
85 , mNum, i, dataAmount);
86
87 // 2 send vector elementwise
88
89 // 2.1 send amount of elements at the vector
90 pMD->pfkt_sendInteger32(pMD,1,&pV->num);
91
92 // 2.2 send element
93 pE = pV->pB;
94 for (n = 0; n < pV->num; n++) {
95     BQF_BQFgetNumAndDenom(pV, n, num, denom)
96     ;
97     pMD->pfkt_sendFloat32(pMD,3,num);
98     pMD->pfkt_sendFloat32(pMD,3,denom);
99     // inc
100     pE++;
101 }
102 #ifdef dBQF_implementPrintFunctions
103     fprintf(stdout,"tx biquad:\n")
104     BQF_PrintBiquad(pV,stdout);
105     fflush(stdout);
106 #endif
107
108 return 0;
109 }
110 // decode data for the HAL functions
111 void * HALimpl_10_decodeData (void * pVarData) {
112     return pVarData;
113 }
114 // delete the variable
115 void HALimpl_10_delete (void * pVarData) {
116     TBQF_BiquadCascade * pBC = (TBQF_BiquadCascade
117 *) pVarData;
118     if (pBC) {
119         BQF_BQFcascadeDelete(pBC);
120         free (pBC);
121     }
122 }
123 }

```

3.2.15 noisegate (audio dynamic processing (generic))

Informations:

variable type id:	11
variable type name:	noisegate
group:	audio dynamic processing (generic)
description:	generic noisegate

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====
2 // variable implementation for a noisegate(var id = 11)
3 // =====
4
5 // updates a variable the AP
6 int HALimpl_11_recvUpdate (void * pVarData, void *
    pMsgData) {
7     return -1;
8 }
9 // create a new variable
10 void * HALimpl_11_create (unsigned int numberOfElements)
    {
11     if (numberOfElements != 1) return NULL;
12     return malloc(sizeof(TDynProc_Noisegate));
13 }
14 // updates the vars at the other APs
15 int HALimpl_11_sendUpdate (void * pVarData, const void *
    pDrv, uint32_t receiver, uint32_t mNum, int32_t i) {
16     return -1;
17 }
18 // decode data for the HAL functions
19 void * HALimpl_11_decodeData (void * pVarData) {
20     return pVarData;
21 }
22 // delete the variable
23 void HALimpl_11_delete (void * pVarData) {
24     TDynProc_Noisegate * pNG = (TDynProc_Noisegate
        *) pVarData;
25     free (pNG);
26 }

```

3.2.16 expander (audio dynamic processing (generic))

Informations:

variable type id:	12
variable type name:	expander
group:	audio dynamic processing (generic)
description:	generic expander

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====
2 // variable implementation for a expander(var id = 12)
3 // =====
4
5 // updates a variable the AP
6 int HALimpl_12_recvUpdate (void * pVarData, void *
   pMsgData) {
7     return -1;
8 }
9 // create a new variable
10 void * HALimpl_12_create (unsigned int numberOfElements)
   {
11     if (numberOfElements != 1) return NULL;
12     return malloc(sizeof(TDynProc_Expander));
13 }
14
15 // updates the vars at the other APs
16 int HALimpl_12_sendUpdate (void * pVarData, const void *
   pDrv, uint32_t receiver, uint32_t mNum, int32_t i) {
17     return -1;
18 }
19 // decode data for the HAL functions
20 void * HALimpl_12_decodeData (void * pVarData) {
21     return pVarData;
22 }
23 // delete the variable
24 void HALimpl_12_delete (void * pVarData) {
25     TDynProc_Expander * pExp = (TDynProc_Expander *)
   pVarData;
26     free (pExp);
27 }
28 }

```

3.2.17 compressor (audio dynamic processing (generic))

Informations:

variable type id:	13
variable type name:	compressor
group:	audio dynamic processing (generic)
description:	generic compressor

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====
2 // variable implementation for a compressor(var id = 13)
3 // =====

```

```

4
5 // updates a variable the AP
6 int HALimpl_13_recvUpdate (void * pVarData, void *
    pMsgData) {
7     return -1;
8 }
9 // create a new variable
10 void * HALimpl_13_create (unsigned int numberOfElements)
    {
11     if (numberOfElements != 1) return NULL;
12     return malloc(sizeof(TDynProc_Compressor));
13 }
14 // updates the vars at the other APs
15 int HALimpl_13_sendUpdate (void * pVarData, const void *
    pDrv, uint32_t receiver, uint32_t mNum, int32_t i) {
16     return -1;
17 }
18 // decode data for the HAL functions
19 void * HALimpl_13_decodeData (void * pVarData) {
20     return pVarData;
21 }
22 // delete the variable
23 void HALimpl_13_delete (void * pVarData) {
24     TDynProc_Compressor * pCmp = (
        TDynProc_Compressor *) pVarData;
25     free (pCmp);
26 }

```

3.2.18 limiter (audio dynamic processing (generic))

Informations:

variable type id:	14
variable type name:	limiter
group:	audio dynamic processing (generic)
description:	generic limiter

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====
2 // variable implementation for a limiter(var id = 14)
3 // =====
4
5 // updates a variable the AP
6 int HALimpl_14_recvUpdate (void * pVarData, void *
    pMsgData) {
7     return -1;
8 }

```

```

9 // create a new variable
10 void * HALimpl_14_create (unsigned int numberOfElements)
11 {
12     if (numberOfElements != 1) return NULL;
13     return malloc(sizeof(TDynProc_Limiter));
14 }
15 // updates the vars at the other APs
16 int HALimpl_14_sendUpdate (void * pVarData, const void *
17     pDrv, uint32_t receiver, uint32_t mNum, int32_t i) {
18     return -1;
19 }
20 // decode data for the HAL functions
21 void * HALimpl_14_decodeData (void * pVarData) {
22     return pVarData;
23 }
24 // delete the variable
25 void HALimpl_14_delete (void * pVarData) {
26     TDynProc_Limiter * pLim = (TDynProc_Limiter *)
27         pVarData;
28     free (pLim);
29 }

```

3.2.19 delay (generic delay)

Informations:

variable type id:	20
variable type name:	delay
group:	generic delay
description:	a generic delay

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====
2 // variable implementation for delay (varID=20)
3 // =====
4
5 // updates a variable the AP
6 int HALimpl_20_recvUpdate ( void * pVarData, void *
7     pMsgData ) {
8     // rx function for delay
9     uint32_t * pD = (uint32_t *) pMsgData;
10     TgenDelay * pV =(TgenDelay *) pVarData;
11
12     int imax, i;
13
14     // at the first position at the message is the
15     // global var index

```

```

14     pD++; // skip it (it's the varindex)
15     // and now we are at amount of values
16     APendianConversation32Bit(pD,eAP_littleEndian);
17     imax = (int) *((int32_t *)pD);
18     pD++;
19
20     // 1. resize
21     if (genDelay_resize(pV, imax)) {
22         return -1;
23     }
24
25     // 2. fill parameters
26     for (i = 0; i < pV->amount; i++) {
27         // no endian conversation
28         pV->pStart[i]*=((float *)pD);
29         pD++;
30     }
31     return 0;
32 }
33 // create a new variable
34 void * HALimpl_20_create ( unsigned int numberOfElements
35 ) {
36     // only one element is allowed
37     if (numberOfElements != 1) {
38         return NULL;
39     }
40     // cerate with now data inside
41     return genDelay_create(0);
42 }
43 // updates the vars at the other APs
44 int HALimpl_20_sendUpdate ( void * pVarData, const void
45 * pDrv, uint32_t receiver, uint32_t mNum, int32_t i )
46 {
47     // tx function for delay
48     TgenDelay * pV =(TgenDelay *) pVarData;
49     TAPMsgDrv * pMD = (TAPMsgDrv *) pDrv;
50
51     int32_t dataAmount;
52
53     // calc amount of transmission bytes
54     // = size of the delay + amount of the elements
55     // at the delay
56     dataAmount = (int32_t) 1 + pV->amount;
57
58     // 1 send header
59     pMD->pfkt_updateVariable(pMD->pDrvData, receiver
60 , mNum, i, dataAmount);
61
62     // 2 send delay
63
64     // 2.1 send amount of elements at the vector
65     pMD->pfkt_sendInteger32(pMD,1,&pV->amount);
66
67     // 2.2 send elements
68     pMD->pfkt_sendFloat32(pMD,pV->amount,pV->pStart)

```

```

64         ;
65         return 0;
66     }
67     // decode data for the HAL functions
68     void * HALimpl_20_decodeData ( void * pVarData ) {
69         return pVarData;
70     }
71     // delete the variable
72     void HALimpl_20_delete ( void * pVarData ) {
73         TgenDelay * pD = (TgenDelay *) pVarData;
74         genDelay_delete(pD);
75     }

```

3.3 Implementation of the HAL functions

3.3.1 jump (no group)

Informations:

function HAL id:	50
variable type name:	jump
group:	no group
description:	increment / decrement instruction pointer

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1  // =====
2  // function implementation for jump(var id = 50)
3  // description: jumps n instructions
4  // =====
5
6  void HALfunc_ID50_jump(void * pIPcpu,
7      TuAPIInterpreterFunctionParameter * pParams) {
8      // parameter number descr: amount of instructions to
9      // jump
10     int32_t* pnumber = &( pParams[0].fp_integer);
11     ((TAPIInterpreterCPU *)pIPcpu)->pIP += pParams
12     [0].fp_integer;
13 }

```

3.3.2 jumpCF (no group)

Informations:

function HAL id:	51
variable type name:	jumpCF
group:	no group
description:	increment / decrement instruction pointer if the CF is set

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```
1 // =====
2 // function implementation for jumpCF(var id = 51)
3 // description: jumps if the carry flag is set n
4 // instructions
5 // =====
6 void HALfunc_ID51_jumpCF(void * pIPcpu,
7     TuAPIInterpreterFunctionParameter * pParams) {
8     if (((TAPIInterpreterCPU *)pIPcpu)->CF) {
9         ((TAPIInterpreterCPU *)pIPcpu)->pIP +=
10             pParams[0].fp_integer;
11     } else {
12         ((TAPIInterpreterCPU *)pIPcpu)->pIP++;
13     }
14 }
```

3.3.3 jumpNCF (no group)

Informations:

function HAL id:	52
variable type name:	jumpNCF
group:	no group
description:	increment / decrement instruction pointer if the CF is not set

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```
1 // =====
2 // function implementation for jumpNCF(var id = 52)
```



```

3 // description: jumps if the carry flag is not set n
  instructions
4 // =====
5
6 void HALfunc_ID52_jumpNCF(void * pIPcpu,
  TuAPIInterpreterFunctionParameter * pParams) {
7     if (!((TAPIInterpreterCPU *)pIPcpu)->CF) {
8         ((TAPIInterpreterCPU *)pIPcpu)->pIP +=
          pParams[0].fp_integer;
9     } else {
10         ((TAPIInterpreterCPU *)pIPcpu)->pIP++;
11     }
12 };

```

3.3.4 setCF (no group)

Informations:

function HAL id:	55
variable type name:	setCF
group:	no group
description:	sets the CF

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====
2 // function implementation for setCF(var id = 55)
3 // description: sets the carry flag
4 // =====
5
6 void HALfunc_ID55_setCF(void * pIPcpu,
  TuAPIInterpreterFunctionParameter * pParams) {
7 // parameter value descr: the value of the CF
8     if (pParams[0].fp_integer) {
9         ((TAPIInterpreterCPU *)pIPcpu)->CF = 1;
10    } else {
11        ((TAPIInterpreterCPU *)pIPcpu)->CF = 0;
12    }
13    ((TAPIInterpreterCPU *)pIPcpu)->pIP++;
14 };

```

3.3.5 update (no group)

Informations:

function HAL id:	56
variable type name:	update
group:	no group
description:	updates a variable

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```
1 // =====
2 // function implementation for update(var id = 56)
3 // description: updates a global variable
4 // =====
5
6 void HALfunc_ID56_update(void * pIPcpu,
7     TuAPIInterpreterFunctionParameter * pParams) {
8     if (TX_updateVariable(APIInterpreterGetAPfromIP
9         (((TAPIInterpreterCPU *)pIPcpu)->IP),(uint32_t
10             )pParams[0].fp_VarIndex)) {
11         ((TAPIInterpreterCPU *)pIPcpu)->EF =
12             -100;
13     } else {
14         ((TAPIInterpreterCPU *)pIPcpu)->pIP++;
15     }
16 }
```

3.3.6 assignConstInteger (rational and integer generic)

Informations:

function HAL id:	20
variable type name:	assignConstInteger
group:	rational and integer generic
description:	a = const val

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```
1 // =====
2 // function implementation for assignConstInteger(var id
3 // = 20)
```

```

3 // description: a = values
4 // =====
5
6 void HALfunc_ID20_assignConstInteger(void * pIPcpu,
   TuAPIInterpreterFunctionParameter * pParams) {
7 // parameter iv descr: integer vector
8     TAPgenericIntegerVector * piv = (
        TAPgenericIntegerVector *) pParams[0].fp_pD;
9 // parameter iStart descr: start index
10    int32_t iStart = pParams[1].fp_integer;
11 // parameter num descr: number of values
12    int32_t num = pParams[2].fp_integer;
13
14    int32_t * pSrc =(int32_t *) &pParams[3].fp_raw;
15
16    APgenericIntegerVector_assignConst(piv, (int)
        iStart, (int)num, pSrc);
17
18    ((TAPIInterpreterCPU *)pIPcpu)->pIP++;
19 };

```

3.3.7 assignInteger (rational and integer generic)

Informations:

function HAL id:	21
variable type name:	assignInteger
group:	rational and integer generic
description:	a = b

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====
2 // function implementation for assignInteger(var id =
   21)
3 // description: a = b
4 // =====
5
6 void HALfunc_ID21_assignInteger(void * pIPcpu,
   TuAPIInterpreterFunctionParameter * pParams) {
7 // parameter a descr: vector a
8     TAPgenericIntegerVector * pa = (
        TAPgenericIntegerVector *)pParams[0].fp_pD;
9 // parameter b descr: vector b
10    TAPgenericIntegerVector* pb = (
        TAPgenericIntegerVector *)pParams[1].fp_pD;
11
12    APgenericIntegerVector_assign(pa, pb);

```

```

13
14         ((TAPInterpreterCPU *)pIPcpu)->pIP++;
15     };

```

3.3.8 addInteger (rational and integer generic)

Informations:

function HAL id:	22
variable type name:	addInteger
group:	rational and integer generic
description:	$c = a + b$

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1  // =====
2  // function implementation for addInteger(var id = 22)
3  // description: c = a + b
4  // =====
5
6  void HALfunc_ID22_addInteger(void * pIPcpu,
7      TuAPIInterpreterFunctionParameter * pParams) {
8      // parameter a descr: vector a
9      TAPgenericIntegerVector * pa = (
10         TAPgenericIntegerVector *)pParams[0].fp_pD;
11      // parameter b descr: vector b
12      TAPgenericIntegerVector* pb = (
13         TAPgenericIntegerVector *)pParams[1].fp_pD;
14      // parameter c descr: vector c
15      TAPgenericIntegerVector* pc = (
16         TAPgenericIntegerVector *)pParams[2].fp_pD;
17
18      APgenericIntegerVector_add(pa, pb, pc);
19
20      ((TAPInterpreterCPU *)pIPcpu)->pIP++;
21  };

```

3.3.9 subInteger (rational and integer generic)

Informations:

function HAL id:	23
variable type name:	subInteger
group:	rational and integer generic
description:	$c = a - b$

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====
2 // function implementation for subInteger(var id = 23)
3 // description: c = a - b
4 // =====
5
6 void HALfunc_ID23_subInteger(void * pIPcpu,
7     TuAPIInterpreterFunctionParameter * pParams) {
8     // parameter a descr: vector a
9     TAPgenericIntegerVector * pa = (
10         TAPgenericIntegerVector *)pParams[0].fp_pD;
11     // parameter b descr: vector b
12     TAPgenericIntegerVector* pb = (
13         TAPgenericIntegerVector *)pParams[1].fp_pD;
14     // parameter c descr: vector c
15     TAPgenericIntegerVector* pc = (
16         TAPgenericIntegerVector *)pParams[2].fp_pD;
17
18     APgenericIntegerVector_sub(pa, pb, pc);
19
20     ((TAPIInterpreterCPU *)pIPcpu)->pIP++;
21 };

```

3.3.10 mulInteger (rational and integer generic)

Informations:

function HAL id:	24
variable type name:	mulInteger
group:	rational and integer generic
description:	$c = a * b$

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====
2 // function implementation for mulInteger(var id = 24)
3 // description: c = a * b
4 // =====
5
6 void HALfunc_ID24_mulInteger(void * pIPcpu,
7     TuAPIInterpreterFunctionParameter * pParams) {

```

```

7 // parameter a descr: vector a
8     TAPgenericIntegerVector * pa = (
9         TAPgenericIntegerVector *)pParams[0].fp_pD;
10 // parameter b descr: vector b
11     TAPgenericIntegerVector* pb = (
12         TAPgenericIntegerVector *)pParams[1].fp_pD;
13 // parameter c descr: vector c
14     TAPgenericIntegerVector* pc = (
15         TAPgenericIntegerVector *)pParams[2].fp_pD;
16
17     APgenericIntegerVector_mul(pa, pb, pc);
18
19     ((TAPInterpreterCPU *)pIPcpu)->pIP++;
20 };

```

3.3.11 divInteger (rational and integer generic)

Informations:

function HAL id:	25
variable type name:	divInteger
group:	rational and integer generic
description:	$c = a / b$

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====
2 // function implementation for divInteger(var id = 25)
3 // description: c = a / b
4 // =====
5
6 void HALfunc_ID25_divInteger(void * pIPcpu,
7     TuAPIInterpreterFunctionParameter * pParams) {
8     // parameter a descr: vector a
9     TAPgenericIntegerVector * pa = (
10         TAPgenericIntegerVector *)pParams[0].fp_pD;
11 // parameter b descr: vector b
12     TAPgenericIntegerVector* pb = (
13         TAPgenericIntegerVector *)pParams[1].fp_pD;
14 // parameter c descr: vector c
15     TAPgenericIntegerVector* pc = (
16         TAPgenericIntegerVector *)pParams[2].fp_pD;
17
18     APgenericIntegerVector_div(pa, pb, pc);
19
20     ((TAPInterpreterCPU *)pIPcpu)->pIP++;
21 };

```

3.3.12 modInteger (rational and integer generic)

Informations:

function HAL id:	26
variable type name:	modInteger
group:	rational and integer generic
description:	c = modulo(a,b)

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====
2 // function implementation for modInteger(var id = 26)
3 // description: c = mod(a,b)
4 // =====
5
6 void HALfunc_ID26_modInteger(void * pIPcpu,
7     TuAPIInterpreterFunctionParameter * pParams) {
8     // parameter a descr: vector a
9     TAPgenericIntegerVector * pa = (
10         TAPgenericIntegerVector *)pParams[0].fp_pD;
11     // parameter b descr: vector b
12     TAPgenericIntegerVector* pb = (
13         TAPgenericIntegerVector *)pParams[1].fp_pD;
14     // parameter c descr: vector c
15     TAPgenericIntegerVector* pc = (
16         TAPgenericIntegerVector *)pParams[2].fp_pD;
17
18     APgenericIntegerVector_mod(pa, pb, pc);
19
20     ((TAPIInterpreterCPU *)pIPcpu)->pIP++;
21 };

```

3.3.13 assignConstRational (rational and integer generic)

Informations:

function HAL id:	27
variable type name:	assignConstRational
group:	rational and integer generic
description:	a = const val

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====
2 // function implementation for assignConstRational(var
  id = 27)
3 // description: a = values
4 // =====
5 void HALfunc_ID27_assignConstRational(void * pIPcpu,
  TuAPIInterpreterFunctionParameter * pParams) {
6 // parameter iv descr: rational vector
7     TAPgenericRationalVector * piv = (
      TAPgenericRationalVector *) pParams[0].fp_pD;
8 // parameter iStart descr: start index
9     int32_t iStart = pParams[1].fp_integer;
10 // parameter num descr: amount of values used
11     int32_t num = pParams[2].fp_integer;
12
13     float * pSrc =(float *) &pParams[3].fp_raw;
14
15     APgenericRationalVector_assignConst (piv, (int)
      iStart, (int)num, pSrc);
16
17     ((TAPIInterpreterCPU *)pIPcpu)->pIP++;
18 };

```

3.3.14 assignRational (rational and integer generic)

Informations:

function HAL id:	28
variable type name:	assignRational
group:	rational and integer generic
description:	a = b

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====
2 // function implementation for assignRational(var id =
  28)
3 // description: a = b
4 // =====
5 void HALfunc_ID28_assignRational(void * pIPcpu,
  TuAPIInterpreterFunctionParameter * pParams) {
6 // parameter a descr: vector a
7     TAPgenericRationalVector * pa = (
      TAPgenericRationalVector *)pParams[0].fp_pD;
8 // parameter b descr: vector b

```



```

9      TAPgenericRationalVector* pb = (
10          TAPgenericRationalVector *)pParams[1].fp_pD;
11
12      APgenericRationalVector_assign(pa, pb);
13
14      ((TAPInterpreterCPU *)pIPcpu)->pIP++;
15  };

```

3.3.15 addRational (rational and integer generic)

Informations:

function HAL id:	29
variable type name:	addRational
group:	rational and integer generic
description:	$c = a + b$

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1  // =====
2  // function implementation for addRational(var id = 29)
3  // description: c = a + b
4  // =====
5  void HALfunc_ID29_addRational(void * pIPcpu,
6      TuAPIInterpreterFunctionParameter * pParams) {
7      // parameter a descr: vector a
8      TAPgenericRationalVector * pa = (
9          TAPgenericRationalVector *)pParams[0].fp_pD;
10     // parameter b descr: vector b
11     TAPgenericRationalVector * pb = (
12         TAPgenericRationalVector *)pParams[1].fp_pD;
13     // parameter c descr: vector c
14     TAPgenericRationalVector * pc = (
15         TAPgenericRationalVector *)pParams[2].fp_pD;
16
17     APgenericRationalVector_add(pa, pb, pc);
18
19     ((TAPInterpreterCPU *)pIPcpu)->pIP++;
20 }

```

3.3.16 subRational (rational and integer generic)

Informations:

function HAL id:	30
variable type name:	subRational
group:	rational and integer generic
description:	$c = a - b$

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```
1 // =====
2 // function implementation for subRational(var id = 30)
3 // description: c = a - b
4 // =====
5 void HALfunc_ID30_subRational(void * pIPcpu,
6     TuAPIInterpreterFunctionParameter * pParams) {
7     TAPgenericRationalVector * pa = (
8         TAPgenericRationalVector *)pParams[0].fp_pD;
9     TAPgenericRationalVector * pb = (
10         TAPgenericRationalVector *)pParams[1].fp_pD;
11     TAPgenericRationalVector * pc = (
12         TAPgenericRationalVector *)pParams[2].fp_pD;
13     APgenericRationalVector_sub(pa, pb, pc);
14     ((TAPIInterpreterCPU *)pIPcpu)->pIP++;
15 };
```

3.3.17 mulRational (rational and integer generic)

Informations:

function HAL id:	31
variable type name:	mulRational
group:	rational and integer generic
description:	$c = a * b$

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====
2 // function implementation for mulRational(var id = 31)
3 // description: c = a * b
4 // =====
5 void HALfunc_ID31_mulRational(void * pIPcpu,
6     TuAPIInterpreterFunctionParameter * pParams) {
7     // parameter a descr: vector a
8     TAPgenericRationalVector * pa = (
9         TAPgenericRationalVector *)pParams[0].fp_pD;
10    // parameter b descr: vector b
11    TAPgenericRationalVector * pb = (
12        TAPgenericRationalVector *)pParams[1].fp_pD;
13    // parameter c descr: vector c
14    TAPgenericRationalVector * pc = (
15        TAPgenericRationalVector *)pParams[2].fp_pD;
16
17    APgenericRationalVector_mul(pa, pb, pc);
18
19    ((TAPIInterpreterCPU *)pIPcpu)->pIP++;
20 };

```

3.3.18 divRational (rational and integer generic)

Informations:

function HAL id:	32
variable type name:	divRational
group:	rational and integer generic
description:	$c = a / b$

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====
2 // function implementation for divRational(var id = 32)
3 // description: c = a / b
4 // =====
5 void HALfunc_ID32_divRational(void * pIPcpu,
6     TuAPIInterpreterFunctionParameter * pParams) {
7     // parameter a descr: vector a
8     TAPgenericRationalVector * pa = (
9         TAPgenericRationalVector *)pParams[0].fp_pD;
10    // parameter b descr: vector b
11    TAPgenericRationalVector * pb = (
12        TAPgenericRationalVector *)pParams[1].fp_pD;
13    // parameter c descr: vector c
14    TAPgenericRationalVector * pc = (
15        TAPgenericRationalVector *)pParams[2].fp_pD;

```

```

12         APgenericRationalVector_div(pa, pb, pc);
13
14         ((TAPInterpreterCPU *)pIPcpu)->pIP++;
15     };
16

```

3.3.19 modRational (rational and integer generic)

Informations:

function HAL id:	33
variable type name:	modRational
group:	rational and integer generic
description:	c = modulo(a ,b)

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1  // =====
2  // function implementation for modRational(var id = 33)
3  // description: c = mod(a,b)
4  // =====
5  void HALfunc_ID33_modRational(void * pIPcpu,
6      TuAPIInterpreterFunctionParameter * pParams) {
7      // parameter a descr: vector a
8      TAPgenericRationalVector * pa = (
9          TAPgenericRationalVector *)pParams[0].fp_pD;
10     // parameter b descr: vector b
11     TAPgenericRationalVector * pb = (
12         TAPgenericRationalVector *)pParams[1].fp_pD;
13     // parameter c descr: vector c
14     TAPgenericRationalVector * pc = (
15         TAPgenericRationalVector *)pParams[2].fp_pD;
16
17     APgenericRationalVector_mod(pa, pb, pc);
18
19     ((TAPInterpreterCPU *)pIPcpu)->pIP++;
20 }

```

3.3.20 compareRationalLess (rational and integer generic)

Informations:

function HAL id:	34
variable type name:	compareRationalLess
group:	rational and integer generic
description:	a<b

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====
2 // function implementation for compareRationalLess(var
  id = 34)
3 // description: a < b ? CF = 1 : CF = 0
4 // =====
5 void HALfunc_ID34_compareRationalLess(void * pIPcpu,
  TuAPIInterpreterFunctionParameter * pParams) {
6 // parameter a descr: vector a
7     TAPgenericRationalVector * pa = (
      TAPgenericRationalVector *)pParams[0].fp_pD;
8 // parameter ia descr: a index
9     int32_t ia = pParams[1].fp_integer;
10 // parameter b descr: vector b
11     TAPgenericRationalVector* pb = (
      TAPgenericRationalVector *)pParams[2].fp_pD;
12 // parameter ib descr: b index
13     int32_t ib = pParams[3].fp_integer;
14
15     if (APgenericRationalVector_cmpEle(pa, pb, ia,
      ib) < 0) {
16         ((TAPIInterpreterCPU *)pIPcpu)->CF = 1;
17     } else {
18         ((TAPIInterpreterCPU *)pIPcpu)->CF = 0;
19     }
20     ((TAPIInterpreterCPU *)pIPcpu)->pIP++;
21
22 }
23

```

3.3.21 compareRationalMore (rational and integer generic)

Informations:

function HAL id:	35
variable type name:	compareRationalMore
group:	rational and integer generic
description:	a>b

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====
2 // function implementation for compareRationalMore(var
  id = 35)
3 // description: a > b ? CF = 1 : CF = 0
4 // =====
5 void HALfunc_ID35_compareRationalMore(void * pIPcpu,
  TuAPIInterpreterFunctionParameter * pParams) {
6 // parameter a descr: vector a
7     TAPgenericRationalVector * pa = (
      TAPgenericRationalVector *)pParams[0].fp_pD;
8 // parameter ia descr: a index
9     int32_t ia = pParams[1].fp_integer;
10 // parameter b descr: vector b
11     TAPgenericRationalVector* pb = (
      TAPgenericRationalVector *)pParams[2].fp_pD;
12 // parameter ib descr: b index
13     int32_t ib = pParams[3].fp_integer;
14
15     if (APgenericRationalVector_cmpEle(pa, pb, ia,
      ib) > 0) {
16         ((TAPIInterpreterCPU *)pIPcpu)->CF = 1;
17     } else {
18         ((TAPIInterpreterCPU *)pIPcpu)->CF = 0;
19     }
20     ((TAPIInterpreterCPU *)pIPcpu)->pIP++;
21 };

```

3.3.22 compareRationalEqual (rational and integer generic)

Informations:

function HAL id:	36
variable type name:	compareRationalEqual
group:	rational and integer generic
description:	a==b

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====
2 // function implementation for compareRationalEqual(var
  id = 36)
3 // description: a == b ? CF = 1 : CF = 0
4 // =====
5 void HALfunc_ID36_compareRationalEqual(void * pIPcpu,
  TuAPIInterpreterFunctionParameter * pParams) {
6 // parameter a descr: vector a
7     TAPgenericRationalVector * pa = (
      TAPgenericRationalVector *)pParams[0].fp_pD;

```

```

8 // parameter ia descr: a index
9     int32_t ia = pParams[1].fp_integer;
10 // parameter b descr: vector b
11     TAPgenericRationalVector* pb = (
12         TAPgenericRationalVector *)pParams[2].fp_pD;
13 // parameter ib descr: b index
14     int32_t ib = pParams[3].fp_integer;
15
16     if (APgenericRationalVector_cmpEle(pa, pb, ia,
17         ib) == 0) {
18         ((TAPInterpreterCPU *)pIPcpu)->CF = 1;
19     } else {
20         ((TAPInterpreterCPU *)pIPcpu)->CF = 0;
21     }
22     ((TAPInterpreterCPU *)pIPcpu)->pIP++;
23 };

```

3.3.23 compareRationalNEqual (rational and integer generic)

Informations:

function HAL id:	37
variable type name:	compareRationalNEqual
group:	rational and integer generic
description:	$a <> b$

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====
2 // function implementation for compareRationalNEqual(var
3 // id = 37)
4 // description: a != b ? CF = 1 : CF = 0
5 // =====
6 void HALfunc_ID37_compareRationalNEqual(void * pIPcpu,
7     TuAPIInterpreterFunctionParameter * pParams) {
8     // parameter a descr: vector a
9     TAPgenericRationalVector * pa = (
10         TAPgenericRationalVector *)pParams[0].fp_pD;
11 // parameter ia descr: a index
12     int32_t ia = pParams[1].fp_integer;
13 // parameter b descr: vector b
14     TAPgenericRationalVector* pb = (
15         TAPgenericRationalVector *)pParams[2].fp_pD;
16 // parameter ib descr: b index
17     int32_t ib = pParams[3].fp_integer;
18
19     if (APgenericRationalVector_cmpEle(pa, pb, ia,
20         ib) == 0) {
21         ((TAPInterpreterCPU *)pIPcpu)->CF = 1;
22     } else {
23         ((TAPInterpreterCPU *)pIPcpu)->CF = 0;
24     }
25     ((TAPInterpreterCPU *)pIPcpu)->pIP++;
26 };

```

```

15         if (APgenericRationalVector_cmpEle(pa, pb, ia,
16             ib) != 0) {
17             ((TAPInterpreterCPU *)pIPcpu)->CF = 1;
18         } else {
19             ((TAPInterpreterCPU *)pIPcpu)->CF = 0;
20         }
21         ((TAPInterpreterCPU *)pIPcpu)->pIP++;
    };

```

3.3.24 compareRationalLessEqual (rational and integer generic)

Informations:

function HAL id:	38
variable type name:	compareRationalLessEqual
group:	rational and integer generic
description:	a<=b

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1  // =====
2  // function implementation for compareRationalLessEqual(
3  //   var id = 38)
4  // description: a <= b ? CF = 1 : CF = 0
5  // =====
6  void HALfunc_ID38_compareRationalLessEqual(void * pIPcpu
7      , TuAPInterpreterFunctionParameter * pParams) {
8      // parameter a descr: vector a
9      TAPgenericRationalVector * pa = (
10         TAPgenericRationalVector *)pParams[0].fp_pD;
11      // parameter ia descr: a index
12      int32_t ia = pParams[1].fp_integer;
13      // parameter b descr: vector b
14      TAPgenericRationalVector* pb = (
15         TAPgenericRationalVector *)pParams[2].fp_pD;
16      // parameter ib descr: b index
17      int32_t ib = pParams[3].fp_integer;
18
19      if (APgenericRationalVector_cmpEle(pa, pb, ia,
20         ib) > 0) {
21         ((TAPInterpreterCPU *)pIPcpu)->CF = 0;
22     } else {
23         ((TAPInterpreterCPU *)pIPcpu)->CF = 1;
24     }
25     ((TAPInterpreterCPU *)pIPcpu)->pIP++;
26 }

```


3.3.25 compareRationalMoreEqual (rational and integer generic)

Informations:

function HAL id:	39
variable type name:	compareRationalMoreEqual
group:	rational and integer generic
description:	a=>b

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====
2 // function implementation for compareRationalMoreEqual(
  var id = 39)
3 // description: a >= b ? CF = 1 : CF = 0
4 // =====
5 void HALfunc_ID39_compareRationalMoreEqual(void * pIPcpu
  , TuAPIInterpreterFunctionParameter * pParams) {
6 // parameter a descr: vector a
7     TAPgenericRationalVector * pa = (
      TAPgenericRationalVector *)pParams[0].fp_pD;
8 // parameter ia descr: a index
9     int32_t ia = pParams[1].fp_integer;
10 // parameter b descr: vector b
11     TAPgenericRationalVector* pb = (
      TAPgenericRationalVector *)pParams[2].fp_pD;
12 // parameter ib descr: b index
13     int32_t ib = pParams[3].fp_integer;
14
15     if (APgenericRationalVector_cmpEle(pa, pb, ia,
      ib) < 0) {
16         ((TAPIInterpreterCPU *)pIPcpu)->CF = 0;
17     } else {
18         ((TAPIInterpreterCPU *)pIPcpu)->CF = 1;
19     }
20     ((TAPIInterpreterCPU *)pIPcpu)->pIP++;
21 };

```

3.3.26 compareIntegerLess (rational and integer generic)

Informations:

function HAL id:	40
variable type name:	compareIntegerLess
group:	rational and integer generic
description:	a<b

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====
2 // function implementation for compareIntegerLess(var id
  = 40)
3 // description: a < b ? CF = 1 : CF = 0
4 // =====
5
6 void HALfunc_ID40_compareIntegerLess(void * pIPcpu,
  TuAPIInterpreterFunctionParameter * pParams) {
7 // parameter a descr: a vector
8     TAPgenericIntegerVector * pa = (
  TAPgenericIntegerVector *) pParams[0].fp_pD;
9 // parameter ia descr: a index
10    int32_t ia = pParams[1].fp_integer;
11 // parameter b descr: b vector
12    TAPgenericIntegerVector * pb = (
  TAPgenericIntegerVector *) pParams[2].fp_pD;
13 // parameter ib descr: b index
14    int32_t ib = pParams[3].fp_integer;
15
16    if (APgenericIntegerVector_cmpEle(pa, pb, ia, ib
  ) < 0) {
17        ((TAPIInterpreterCPU *)pIPcpu)->CF = 1;
18    } else {
19        ((TAPIInterpreterCPU *)pIPcpu)->CF = 0;
20    }
21    ((TAPIInterpreterCPU *)pIPcpu)->pIP++;
22 };

```

3.3.27 compareIntegerMore (rational and integer generic)

Informations:

function HAL id:	41
variable type name:	compareIntegerMore
group:	rational and integer generic
description:	a>b

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====

```

```

2 // function implementation for comparaIntegerMore(var id
  = 41)
3 // description: a > b ? CF = 1 : CF = 0
4 // =====
5
6 void HALfunc_ID41_compareIntegerMore(void * pIPcpu,
  TuAPIInterpreterFunctionParameter * pParams) {
7 // parameter a descr: a vector
8     TAPgenericIntegerVector * pa = (
  TAPgenericIntegerVector *) pParams[0].fp_pD;
9 // parameter ia descr: a index
10    int32_t ia = pParams[1].fp_integer;
11 // parameter b descr: b vector
12    TAPgenericIntegerVector * pb = (
  TAPgenericIntegerVector *) pParams[2].fp_pD;
13 // parameter ib descr: b index
14    int32_t ib = pParams[3].fp_integer;
15
16    if (APgenericIntegerVector_cmpEle(pa, pb, ia, ib
  ) > 0) {
17        ((TAPIInterpreterCPU *)pIPcpu)->CF = 1;
18    } else {
19        ((TAPIInterpreterCPU *)pIPcpu)->CF = 0;
20    }
21    ((TAPIInterpreterCPU *)pIPcpu)->pIP++;
22 };

```

3.3.28 compareIntegerEqual (rational and integer generic)

Informations:

function HAL id:	42
variable type name:	compareIntegerEqual
group:	rational and integer generic
description:	a==b

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====
2 // function implementation for comparaIntegerEqual(var
  id = 42)
3 // description: a == b ? CF = 1 : CF = 0
4 // =====
5
6 void HALfunc_ID42_compareIntegerEqual(void * pIPcpu,
  TuAPIInterpreterFunctionParameter * pParams) {
7 // parameter a descr: a vector

```

```

8      TAPgenericIntegerVector * pa = (
          TAPgenericIntegerVector *) pParams[0].fp_pD;
9  // parameter ia descr: a index
10     int32_t ia = pParams[1].fp_integer;
11  // parameter b descr: b vector
12     TAPgenericIntegerVector * pb = (
          TAPgenericIntegerVector *) pParams[2].fp_pD;
13  // parameter ib descr: b index
14     int32_t ib = pParams[3].fp_integer;
15
16     if (APgenericIntegerVector_cmpEle(pa, pb, ia, ib
17         ) == 0) {
18         ((TAPInterpreterCPU *)pIPcpu)->CF = 1;
19     } else {
20         ((TAPInterpreterCPU *)pIPcpu)->CF = 0;
21     }
22     ((TAPInterpreterCPU *)pIPcpu)->pIP++;
};

```

3.3.29 compareIntegerNEqual (rational and integer generic)

Informations:

function HAL id:	43
variable type name:	compareIntegerNEqual
group:	rational and integer generic
description:	a<>b

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1  // =====
2  // function implementation for comparaIntegerNEqual(var
   id = 43)
3  // description: a != b ? CF = 1 : CF = 0
4  // =====
5
6  void HALfunc_ID43_compareIntegerNEqual(void * pIPcpu,
   TuAPIInterpreterFunctionParameter * pParams) {
7  // parameter a descr: a vector
8     TAPgenericIntegerVector * pa = (
          TAPgenericIntegerVector *) pParams[0].fp_pD;
9  // parameter ia descr: a index
10     int32_t ia = pParams[1].fp_integer;
11  // parameter b descr: b vector
12     TAPgenericIntegerVector * pb = (
          TAPgenericIntegerVector *) pParams[2].fp_pD;
13  // parameter ib descr: b index
14     int32_t ib = pParams[3].fp_integer;

```

```

15         if (APgenericIntegerVector_cmpEle(pa, pb, ia, ib
16             ) != 0) {
17             ((TAPInterpreterCPU *)pIPcpu)->CF = 1;
18         } else {
19             ((TAPInterpreterCPU *)pIPcpu)->CF = 0;
20         }
21         ((TAPInterpreterCPU *)pIPcpu)->pIP++;
22     };
23

```

3.3.30 compareIntegerLessEqual (rational and integer generic)

Informations:

function HAL id:	44
variable type name:	compareIntegerLessEqual
group:	rational and integer generic
description:	$a \leq b$

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1  // =====
2  // function implementation for comparaIntegerLessEqual(
3  //   var id = 44)
4  // description: a <= b ? CF = 1 : CF = 0
5  // =====
6  void HALfunc_ID44_compareIntegerLessEqual(void * pIPcpu,
7      TuAPInterpreterFunctionParameter * pParams) {
8      // parameter a descr: a vector
9      TAPgenericIntegerVector * pa = (
10         TAPgenericIntegerVector *) pParams[0].fp_pD;
11      // parameter ia descr: a index
12      int32_t ia = pParams[1].fp_integer;
13      // parameter b descr: b vector
14      TAPgenericIntegerVector * pb = (
15         TAPgenericIntegerVector *) pParams[2].fp_pD;
16      // parameter ib descr: b index
17      int32_t ib = pParams[3].fp_integer;
18
19      if (APgenericIntegerVector_cmpEle(pa, pb, ia, ib
20          ) > 0) {
21          ((TAPInterpreterCPU *)pIPcpu)->CF = 0;
22      } else {
23          ((TAPInterpreterCPU *)pIPcpu)->CF = 1;
24      }
25  }

```

```

20     }
21     ((TAPInterpreterCPU *)pIPcpu)->pIP++;
22 };

```

3.3.31 compareIntegerMoreEqual (rational and integer generic)

Informations:

function HAL id:	45
variable type name:	compareIntegerMoreEqual
group:	rational and integer generic
description:	a>=>b

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1  // =====
2  // function implementation for comparaIntegerMoreEqual(
   // var id = 45)
3  // description: a >= b ? CF = 1 : CF = 0
4  // =====
5
6  void HALfunc_ID45_compareIntegerMoreEqual(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {
7  // parameter a descr: a vector
8      TAPgenericIntegerVector * pa = (
   TAPgenericIntegerVector *) pParams[0].fp_pD;
9  // parameter ia descr: a index
10     int32_t ia = pParams[1].fp_integer;
11  // parameter b descr: b vector
12     TAPgenericIntegerVector * pb = (
   TAPgenericIntegerVector *) pParams[2].fp_pD;
13  // parameter ib descr: b index
14     int32_t ib = pParams[3].fp_integer;
15
16     if (APgenericIntegerVector_cmpEle(pa, pb, ia, ib
17     ) < 0) {
18         ((TAPInterpreterCPU *)pIPcpu)->CF = 0;
19     } else {
20         ((TAPInterpreterCPU *)pIPcpu)->CF = 1;
21     }
22     ((TAPInterpreterCPU *)pIPcpu)->pIP++;
23 };

```

3.3.32 readSample (libsndfile sample based)

Informations:

function HAL id:	60
variable type name:	readSample
group:	libsndfile sample based
description:	reads a sample form a wav file

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====
2 // function implementation for readSample(var id = 60)
3 // description: reading a sample from an input
4 // =====
5
6 void HALfunc_ID60_readSample(void * pIPcpu,
7     TuAPIInterpreterFunctionParameter * pParams) {
8     // parameter channel descr: the channel
9     // int32_t* pchannel = pParams[0].fp_integer;
10    // parameter resValue descr: the result of the action
11    // int32_t* pSample = pParams[1].fp_pD;
12
13    // forward declaration of the global variable
14    extern TStjWAVmodule gWAVModule;
15
16    if (WAVmoduleGetInput(&gWAVModule, pParams[0].
17        fp_integer, ((TAPvarRational *)pParams[1].
18        fp_pD)->pR)){
19        ((TAPInterpreterCPU *)pIPcpu)->EF = 1;
20    }
21
22    ((TAPInterpreterCPU *)pIPcpu)->pIP++;
23 };
```

3.3.33 writeSample (libsndfile sample based)

Informations:

function HAL id:	61
variable type name:	writeSample
group:	libsndfile sample based
description:	writes a sample to a wav file

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====
2 // function implementation for writeSample(var id = 61)
3 // description: writes a sample to a output
4 // =====
5
6 void HALfunc_ID61_writeSample(void * pIPcpu,
7     TuAPIInterpreterFunctionParameter * pParams) {
8     // parameter channel descr: the channel
9     //     int32_t* pchannel = &(amp; pParams[0].fp_integer);
10    // parameter value descr: the value to be written to the
11    //     output
12    //     void ** pvalue = &(amp; pParams[1].fp_pD);
13
14    // forward declaration of the global variable
15    extern TStjWAVmodule gWAVModule;
16    if (WAVmoduleSetOutput(&gWAVModule, pParams[0].
17        fp_integer, *((TAPvarRational *)pParams[1].
18        fp_pD)->pR)){
19        ((TAPIInterpreterCPU *)pIPcpu)->EF = 1;
20    }
21
22    ((TAPIInterpreterCPU *)pIPcpu)->pIP++;
23 };

```

3.3.34 initRFFT (fftw3 & complex)

Informations:

function HAL id:	130
variable type name:	initRFFT
group:	fftw3 & complex
description:	inits the fft structure as real input fft

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====
2 // function implementation for initRFFT(var id = 130)
3 // description: init FFT structure as real input FFT
4 // =====
5
6 void HALfunc_ID130_initRFFT(void * pIPcpu,
7     TuAPIInterpreterFunctionParameter * pParams) {
8     // parameter N descr: FFT length
9     int32_t N = pParams[0].fp_integer;

```



```

10 // parameter x descr: input
11 float * px = (float *) pParams[1].fp_pD;
12
13 // parameter y descr: output
14 fftwf_complex * py = (fftwf_complex *) pParams
    [2].fp_pD;
15
16 // parameter fftStruct descr: FFT structure
17 TAPInterpreterVariable * pFFTVar = pParams[3].
    fp_pV;
18
19 if (pFFTVar->pData) fftwf_destroy_plan(pFFTVar->
    pData);
20 pFFTVar->pData = fftwf_plan_dft_r2c_1d(N, px, py
    , FFTW_ESTIMATE);
21
22 // increment IP
23 ((TAPInterpreterCPU *)pIPcpu)->pIP++;
24 };

```

3.3.35 initIFFT (fftw3 & complex)

Informations:

function HAL id:	131
variable type name:	initIFFT
group:	fftw3 & complex
description:	inits the fft structure as ifft

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====
2 // function implementation for initIFFT(var id = 131)
3 // description: init inverse FFT structure
4 // =====
5
6 void HALfunc_ID131_initIFFT(void * pIPcpu,
    TuAPIInterpreterFunctionParameter * pParams) {
7     // parameter N descr: FFT length
8     int32_t N = pParams[0].fp_integer;
9
10    // parameter x descr: input
11    fftwf_complex * px = (fftwf_complex *) pParams
        [1].fp_pD;
12
13    // parameter y descr: output
14    float * py = (float *) pParams[2].fp_pD;
15

```

```

16 // parameter fftStruct descr: FFT structure
17 TAPInterpreterVariable * pFFTVar = pParams[3].
   fp_pV;
18
19 if (pFFTVar->pData) fftwf_destroy_plan(pFFTVar->
   pData);
20 pFFTVar->pData = fftwf_plan_dft_c2r_1d(N, px, py
   , FFTW_ESTIMATE);
21
22 // increment IP
23 ((TAPInterpreterCPU *)pIPcpu)->pIP++;
24 };

```

3.3.36 processRFFT (fftw3 & complex)

Informations:

function HAL id:	132
variable type name:	processRFFT
group:	fftw3 & complex
description:	processes the FFT

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====
2 // function implementation for processRFFT(var id = 132)
3 // description: processes the real input FFT
4 // =====
5
6 void HALfunc_ID132_processRFFT(void * pIPcpu,
   TuAPIInterpreterFunctionParameter * pParams) {
7     // parameter fftStruct descr: the fft structure
8     fftwf_execute(pParams[0].fp_pV->pData);
9
10    // increment IP
11    ((TAPInterpreterCPU *)pIPcpu)->pIP++;
12 };

```

3.3.37 processIFFT (fftw3 & complex)

Informations:

function HAL id:	133
variable type name:	processIFFT
group:	fftw3 & complex
description:	processes the IFFT

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====
2 // function implementation for processIFFT(var id = 133)
3 // description: processes the IFFT
4 // =====
5
6 void HALfunc_ID133_processIFFT(void * pIPcpu,
7     TuAPIInterpreterFunctionParameter * pParams) {
8     // parameter fftStruct descr: the fft structure
9     fftwf_execute(pParams[0].fp_pV->pData);
10
11     // increment IP
12     ((TAPIInterpreterCPU *)pIPcpu)->pIP++;
13 }
```

3.3.38 readSampleFrame (libsndfile overlapped frame based)

Informations:

function HAL id:	62
variable type name:	readSampleFrame
group:	libsndfile overlapped frame based
description:	reads a block of samples

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====
2 // function implementation for readSampleFrame(var id =
3 // 62)
4 // description: reading a frames of sample from an input
5 // =====
6
7 void HALfunc_ID62_readSampleFrame(void * pIPcpu,
8     TuAPIInterpreterFunctionParameter * pParams) {
9     // parameter channel descr: the channel
10    int32_t channel = pParams[0].fp_integer;
11    // parameter frameBuffer descr: the buffer witch
12    // receives the samples
13    float * pFrame = (float *) pParams[1].fp_pD;
14 }
```

```

11 // parameter waitForNewFrame descr: if not zero
12 // the function waits for a new sample frame
13 int waitForNewFrame = (int)pParams[2].fp_integer
14 ;
15
16 extern TStjFrameWAVmodule gFrameWAVModule;
17
18 if (FrameWAVmoduleGetInput(&gFrameWAVModule,
19 channel,pFrame)){
20 ((TAPInterpreterCPU *)pIPcpu)->EF = 1;
21 }
22 ((TAPInterpreterCPU *)pIPcpu)->pIP++;
23 };

```

3.3.39 writeSampleFrame (libsndfile overlapped frame based)

Informations:

function HAL id:	63
variable type name:	writeSampleFrame
group:	libsndfile overlapped frame based
description:	writes a block of samples

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====
2 // function implementation for writeSampleFrame(var id =
3 // 63)
4 // description: writes a frame of samples to a output
5 // =====
6 void HALfunc_ID63_writeSampleFrame(void * pIPcpu,
7 TuAPInterpreterFunctionParameter * pParams) {
8 // parameter channel descr: the channel
9 int32_t channel = pParams[0].fp_integer;
10 // parameter frameBuffer descr: the buffer which
11 // is written to the channel
12 error rational vartype
13 float * pFrame = (float *) pParams[1].fp_pD;
14
15 extern TStjFrameWAVmodule gFrameWAVModule;
16
17 if (FrameWAVmoduleSetOutput(&gFrameWAVModule,
18 channel,pFrame)){
19 ((TAPInterpreterCPU *)pIPcpu)->EF = 1;
20 }
21 }

```

```

17     }
18     ((TAPInterpreterCPU *)pIPcpu)->pIP++;
19 }

```

3.3.40 readSampleFrame (ADSP 21369 blockbased, ADC in, DAC3 & DAC4 out, UART @9600,n,8,1)

Informations:

function HAL id:	62
variable type name:	readSampleFrame
group:	ADSP 21369 blockbased, ADC in, DAC3 & DAC4 out, UART @9600,n,8,1
description:	reads a block of samples

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1  // =====
2  // function implementation for readSampleFrame(var id =
   62)
3  // description: reading a frames of sample from an input
4  // =====
5
6  void HALfunc_ID62_readSampleFrame(void * pIPcpu,
   TuAPIInterpreterFunctionParameter * pParams) {
7  // parameter channel descr: the channel
   // pParams[0].fp_integer
8  // parameter resValue descr: the result of the action
9  TAPgenericRationalVector * pRV = (
   TAPgenericRationalVector *) pParams[1].fp_pD;
10 // parameter waitForNewFrame descr: if not zero the
   function waits for a new sample frame
11     int waitForNewFrame = (int)pParams[2].fp_integer
12     ;
13
14     TCodecChannel * pC;
15
16     // the channels at the AP starts with 1; at the
   ADSP with 0
17     pC = ADSP_getChannel((int)pParams[0].fp_integer
   -1);
18     if (!pC) {
19         ((TAPInterpreterCPU *)pIPcpu)->EF = -1;
20         return;
21     }
22

```

```

23     ADSP_readSamplesFromChannel(pC,pRV->pVal,pRV->
        num, waitForNewFrame);
24
25     ((TAPInterpreterCPU *)pIPcpu)->pIP++;
26 };

```

3.3.41 writeSampleFrame (ADSP 21369 blockbased, ADC in, DAC3 & DAC4 out, UART @9600,n,8,1)

Informations:

function HAL id:	63
variable type name:	writeSampleFrame
group:	ADSP 21369 blockbased, ADC in, DAC3 & DAC4 out, UART @9600,n,8,1
description:	writes a block of samples

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1  // =====
2  // function implementation for writeSampleFrame(var id =
    63)
3  // description: writes a frame of samples to a output
4  // =====
5
6  void HALfunc_ID63_writeSampleFrame(void * pIPcpu,
    TuAPInterpreterFunctionParameter * pParams) {
7  // parameter channel descr: the channel
8  // pParams[0].fp_integer
9  // parameter resValue descr: the result of the action
10     TAPgenericRationalVector * pRV = (
        TAPgenericRationalVector *) pParams[1].fp_pD;
11
12     TCodecChannel * pC;
13
14     // the channels at the AP starts with 1; at the
        ADSP with 0
15     pC = ADSP_getChannel((int)pParams[0].fp_integer
        -1);
16     if (!pC) {
17         ((TAPInterpreterCPU *)pIPcpu)->EF = -1;
18         return;
19     }
20
21     ADSP_writesSamplesToChannel(pC,pRV->pVal,pRV->
        num);
22

```

```

23         ((TAPInterpreterCPU *)pIPcpu)->pIP++;
24     };

```

3.3.42 readSampleFrame (libsndfile frame based)

Informations:

function HAL id:	62
variable type name:	readSampleFrame
group:	libsndfile frame based
description:	reads a block of samples

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1  // =====
2  // function implementation for readSampleFrame(var id =
   // 62)
3  // description: reading a frames of sample from an input
4  // =====
5
6  void HALfunc_ID62_readSampleFrame(void * pIPcpu,
   TuAPIInterpreterFunctionParameter * pParams) {
7  // parameter channel descr: the channel
8      int32_t channel = pParams[0].fp_integer;
9  // parameter frameBuffer descr: the buffer witch
   receives the samples
10     TAPvarRational * pRA = (TAPvarRational *)
       pParams[1].fp_pD;
11 // parameter waitForNewFrame descr: if not zero the
   function waits for a new sample frame
12     int waitForNewFrame = (int)pParams[2].fp_integer
       ;
13
14     extern TStjFrameWAVmodule gFrameWAVModule;
15
16     if (FrameWAVmoduleGetInput(&gFrameWAVModule,(int)
       channel,pRA->num,pRA->pR)) {
17         ((TAPInterpreterCPU *)pIPcpu)->EF = -1;
18     }
19     ((TAPInterpreterCPU *)pIPcpu)->pIP++;
20 };

```

3.3.43 writeSampleFrame (libsndfile frame based)

Informations:

function HAL id:	63
variable type name:	writeSampleFrame
group:	libsndfile frame based
description:	writes a block of samples

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====
2 // function implementation for writeSampleFrame(var id =
   // 63)
3 // description: writes a frame of samples to a output
4 // =====
5
6 void HALfunc_ID63_writeSampleFrame(void * pIPcpu,
   TuAPIInterpreterFunctionParameter * pParams) {
7 // parameter channel descr: the channel
8     int32_t channel = pParams[0].fp_integer;
9 // parameter frameBuffer descr: the buffer which is
   // written to the channel
10     TAPvarRational * pRA = (TAPvarRational *)
        pParams[1].fp_pD;
11
12     extern TStjFrameWAVmodule gFrameWAVModule;
13
14     if (FrameWAVmoduleSetOutput(&gFrameWAVModule, (
        int)channel, pRA->num, pRA->pR)) {
15         ((TAPIInterpreterCPU *)pIPcpu)->EF = -1;
16     }
17     ((TAPIInterpreterCPU *)pIPcpu)->pIP++;
18 };

```

3.3.44 uiSetDim (MSP430-169STK)

Informations:

function HAL id:	200
variable type name:	uiSetDim
group:	MSP430-169STK
description:	sets the dimension of the UI (empty because of real hardware)

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====
2 // function implementation for uiSetDim(var id = 200)
3 // description: sets the dimension of a UI
4 // =====
5
6 void HALfunc_ID200_uiSetDim(void * pIPcpu,
7     TuAPInterpreterFunctionParameter * pParams) {
8     ((TAPInterpreterCPU *)pIPcpu)->pIP++;
9 };
```

3.3.45 uiInitPanel (MSP430-169STK)

Informations:

function HAL id:	201
variable type name:	uiInitPanel
group:	MSP430-169STK
description:	inits the panel for MSP430-169STK eval board

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====
2 // function implementation for uiInitPanel(var id = 201)
3 // description: inits a panel
4 // =====
5
6 void HALfunc_ID201_uiInitPanel(void * pIPcpu,
7     TuAPInterpreterFunctionParameter * pParams) {
8     // do nothing
9     ((TAPInterpreterCPU *)pIPcpu)->pIP++;
10 };
```

3.3.46 uiInitButton (MSP430-169STK)

Informations:

function HAL id:	202
variable type name:	uiInitButton
group:	MSP430-169STK
description:	inits the button for MSP430-169STK eval board

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====
2 // function implementation for uiInitButton(var id =
  // 202)
3 // description: inits a button
4 // =====
5
6 void HALfunc_ID202_uiInitButton(void * pIPcpu,
  TuAPInterpreterFunctionParameter * pParams) {
7 // parameter uiUUID descr: uuid of the button
8   int32_t UUID = pParams[0].fp_integer;
9 // parameter b descr: button
10  TMS430_buttonVec * pbv = (TMS430_buttonVec *)
    pParams[1].fp_pD;
11 // parameter bIndex descr: button index
12  int32_t bIndex = pParams[2].fp_integer;
13 // parameter p descr: panel
14  TMS430_panelVec * pp = (TMS430_panelVec *)
    pParams[3].fp_pD;
15 // parameter pIndex descr: panel index
16  int32_t pIndex = pParams[4].fp_integer;
17
18
19  if ((bIndex < 0) || (bIndex >= pbv->num)) {
20      ((TAPInterpreterCPU *)pIPcpu)->EF =
        -202;
21      return;
22  }
23  switch (UUID) {
24      case 21:
25          pbv->pB[bIndex].port1Mask = BIT5
            ;
26          break;
27      case 22:
28          pbv->pB[bIndex].port1Mask = BIT6
            ;
29          break;
30      case 23:
31          pbv->pB[bIndex].port1Mask = BIT7
            ;
32          break;
33      default:
34          ((TAPInterpreterCPU *)pIPcpu)->
            EF = -202;
35          return;
36  }
37  ((TAPInterpreterCPU *)pIPcpu)->pIP++;
38 };

```

3.3.47 uiInitDisplay (MSP430-169STK)

Informations:

function HAL id:	203
variable type name:	uiInitDisplay
group:	MSP430-169STK
description:	inits the display for MSP430-169STK eval board

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```
1 // =====
2 // function implementation for uiInitDisplay(var id =
3 // 203)
4 // description: inits a display
5 // =====
6 void HALfunc_ID203_uiInitDisplay(void * pIPcpu,
7     TuAPIInterpreterFunctionParameter * pParams) {
8     // parameter uiUUID descr: uuid of the display
9     int32_t UUID = pParams[0].fp_integer;
10    // parameter b descr: button
11    TMSP430_displayVec * pdv = (TMSP430_displayVec
12        *) pParams[1].fp_pD;
13    // parameter dIndex descr: display index
14    int32_t dIndex = pParams[2].fp_integer;
15    // parameter p descr: panel
16    TMSP430_panelVec * ppv = pParams[3].fp_pD;
17    // parameter pIndex descr: panel index
18    int32_t pIndex = pParams[4].fp_integer;
19
20    if ((dIndex < 0) || (dIndex >= pdv->num)) {
21        ((TAPIInterpreterCPU *)pIPcpu)->EF =
22            -203;
23        return;
24    }
25    if (UUID != 11) {
26        ((TAPIInterpreterCPU *)pIPcpu)->EF =
27            -203;
28        return;
29    }
30    // do nothing
31    ((TAPIInterpreterCPU *)pIPcpu)->pIP++;
32 }
```

3.3.48 uiInitLED (MSP430-169STK)

Informations:

function HAL id:	204
variable type name:	uiInitLED
group:	MSP430-169STK
description:	inits the LED for MSP430-169STK eval board

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```
1 // =====
2 // function implementation for uiInitLED(var id = 204)
3 // description: inits a LED
4 // =====
5
6 void HALfunc_ID204_uiInitLED(void * pIPcpu,
7     TuAPIInterpreterFunctionParameter * pParams) {
8     // parameter uiUUID descr: uuid of the LED
9     int32_t UUID = pParams[0].fp_integer;
10    // parameter l descr: led
11    TMSP430_LEDvec * plv = (TMSP430_LEDvec *)
12        pParams[1].fp_pD;
13    // parameter lIndex descr: LED index
14    int32_t lIndex = pParams[2].fp_integer;
15    // parameter p descr: panel
16    TMSP430_panelVec * ppv = (TMSP430_panelVec *)
17        pParams[3].fp_pD;
18    // parameter pIndex descr: panel index
19    int32_t pIndex = pParams[4].fp_integer;
20
21    if ((lIndex < 0) || (lIndex >= plv->num)) {
22        ((TAPIInterpreterCPU *)pIPcpu)->EF =
23            -204;
24        return;
25    }
26    switch (UUID) {
27        case 31:
28            plv->pL[lIndex].port3mask = BIT6
29                ;
30            break;
31        case 32:
32            plv->pL[lIndex].port3mask = BIT7
33                ;
34            break;
35        default:
36            ((TAPIInterpreterCPU *)pIPcpu)->
37                EF = -204;
38            return;
39    }
40 }
```

```

33         ((TAPInterpreterCPU *)pIPcpu)->pIP++;
34     };

```

3.3.49 uiCheckButtonPressed (MSP430-169STK)

Informations:

function HAL id:	210
variable type name:	uiCheckButtonPressed
group:	MSP430-169STK
description:	checks a button at the MSP430-169STK eval board

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1  // =====
2  // function implementation for uiCheckButtonPressed(var
   id = 210)
3  // description: if the button was pressed the CF is set
4  // =====
5
6  void HALfunc_ID210_uiCheckButtonPressed(void * pIPcpu,
   TuAPIInterpreterFunctionParameter * pParams) {
7  // parameter b descr: button
8      TMSP430_buttonVec * pbv = (TMSP430_buttonVec *)
   pParams[0].fp_pD;
9  // parameter bIndex descr: button index
10     int32_t bIndex = pParams[1].fp_integer;
11
12     if ((bIndex < 0) || (bIndex >= pbv->num)){
13         ((TAPInterpreterCPU *)pIPcpu)->EF =
   -210;
14         return;
15     }
16     unsigned char r = P1IN & pbv->pB[bIndex].
   port1Mask;
17     ((TAPInterpreterCPU *)pIPcpu)->CF = r ? 1 : 0;
18     ((TAPInterpreterCPU *)pIPcpu)->pIP++;
19 };

```

3.3.50 uiSetLED (MSP430-169STK)

Informations:

function HAL id:	211
variable type name:	uiSetLED
group:	MSP430-169STK
description:	sets a LED at the MSP430-169STK eval board

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```
1 // =====
2 // function implementation for uiSetLED(var id = 211)
3 // description: set the LED state
4 // =====
5
6 void HALfunc_ID211_uiSetLED(void * pIPcpu,
7     TuAPIInterpreterFunctionParameter * pParams) {
8     // parameter 1 descr: LED
9     TMSP430_LEDvec * plv = pParams[0].fp_pD;
10    // parameter 1Index descr: LED index
11    int32_t lIndex = pParams[1].fp_integer;
12    // parameter onFlag descr: if the flag is not zero the
13    // LED is turned on
14    int32_t onFlag = pParams[2].fp_integer;
15
16    if ((lIndex < 0) || (lIndex >= plv->num)) {
17        ((TAPIInterpreterCPU *)pIPcpu)->EF =
18            -211;
19        return;
20    }
21
22    if (onFlag) {
23        P3OUT &= ~(plv->pL[lIndex].port3mask);
24    } else {
25        P3OUT |= plv->pL[lIndex].port3mask;
26    }
27    ((TAPIInterpreterCPU *)pIPcpu)->pIP++;
28 }
```

3.3.51 uiSetDisplay (MSP430-169STK)

Informations:

function HAL id:	212
variable type name:	uiSetDisplay
group:	MSP430-169STK
description:	sets a display text at the MSP430-169STK eval board

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====
2 // function implementation for uiSetDisplay(var id =
   // 212)
3 // description: set the text of a display
4 // =====
5
6 void HALfunc_ID212_uiSetDisplay(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {
7 // parameter d descr: display
8     T MSP430_display * pd = pParams[0].fp_pD;
9 // parameter dIndex descr: display index
10    int32_t dIndex = pParams[1].fp_integer;
11 // parameter s descr: the string
12    TAPstringVector * pSV = pParams[2].fp_pD;
13 // parameter iString descr: index of the string at the
   array
14    int iString = (int) pParams[3].fp_integer;
15
16    msp430_LCD_print(0,0,pSV->sv[iString].szTxt);
17    ((TAPInterpreterCPU *)pIPcpu)->pIP++;
18 };

```

3.3.52 uiSetDim (gtk+ for Windows)

Informations:

function HAL id:	200
variable type name:	uiSetDim
group:	gtk+ for Windows
description:	sets the dimension of the UI

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====
2 // function implementation for uiSetDim(var id = 200)
3 // description: sets the dimension of a UI
4 // =====
5
6 void HALfunc_ID200_uiSetDim(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {

```

```

7 // parameter xPos descr: x position in pixel
8     int32_t xPos = pParams[0].fp_integer;
9 // parameter yPos descr: y position in pixel
10    int32_t yPos = pParams[1].fp_integer;
11 // parameter xLen descr: x width
12    int32_t xLen = pParams[2].fp_integer;
13 // parameter yLen descr: y height
14    int32_t yLen = pParams[3].fp_integer;
15 // parameter ui descr: UI element
16    TAPgtkUIvector * puiVec = (TAPgtkUIvector *)
17        pParams[4].fp_pV;
18 // parameter uiIndex descr: UI element index
19    int32_t uiIndex = pParams[5].fp_integer;
20
21    APgtkUI_setCoordinates (
22        &(puiVec->pUI[uiIndex]),
23        xPos,
24        yPos,
25        xLen,
26        yLen
27    );
28    ((TAPInterpreterCPU *)pIPcpu)->pIP++;
29 };

```

3.3.53 uiInitPanel (gtk+ for Windows)

Informations:

function HAL id:	201
variable type name:	uiInitPanel
group:	gtk+ for Windows
description:	inits a panel

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====
2 // function implementation for uiInitPanel(var id = 201)
3 // description: inits a panel
4 // =====
5
6 void HALfunc_ID201_uiInitPanel(void * pIPcpu,
7     TuAPIInterpreterFunctionParameter * pParams) {
8     // parameter uiUUID descr: uuid of the pannel
9     int32_t uiUUID = pParams[0].fp_integer;
10    // parameter p descr: pannel
11    TAPgtkUIvector * puiVec = (TAPgtkUIvector *)
12        pParams[1].fp_pD;
13    // parameter pIndex descr: pannel index

```



```

12     int32_t pIndex = pParams[2].fp_integer;
13
14     if (APgtkUI_createUI (
15         &(puiVec->pUI[pIndex]),
16         NULL,
17         uiUUID,
18         eAPgtkUItype_panel
19     )
20     ) {
21         ((TAPInterpreterCPU *)pIPcpu)->EF = -10;
22     }
23
24     ((TAPInterpreterCPU *)pIPcpu)->pIP++;
25 };

```

3.3.54 uiInitButton (gtk+ for Windows)

Informations:

function HAL id:	202
variable type name:	uiInitButton
group:	gtk+ for Windows
description:	inits a button

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1  // =====
2  // function implementation for uiInitButton(var id =
   // 202)
3  // description: inits a button
4  // =====
5
6  void HALfunc_ID202_uiInitButton(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {
7  // parameter uiUUID descr: uuid of the pannel
   int32_t uiUUID = pParams[0].fp_integer;
8  // parameter b descr: button
   TAPgtkUIvector * puiB = (TAPgtkUIvector *)
9  // parameter bIndex descr: button index
   pParams[1].fp_pD;
10 // parameter p descr: panel
   int32_t bIndex = pParams[2].fp_integer;
11 // parameter pIndex descr: panel index
   TAPgtkUIvector * puiP = (TAPgtkUIvector *)
12 // parameter pIndex descr: panel index
   pParams[3].fp_pD;
13 // parameter pIndex descr: panel index
   int32_t pIndex = pParams[4].fp_integer;
14
15     if (APgtkUI_createUI (
16
17
18

```

```

19         &(puiB->pUI[bIndex]),
20         &(puiP->pUI[pIndex]),
21         uiUUID,
22         eAPgtkUItype_button
23     )
24     {
25         ((TAPInterpreterCPU *)pIPcpu)->EF = -10;
26     }
27
28     ((TAPInterpreterCPU *)pIPcpu)->pIP++;
29 };

```

3.3.55 uiInitDisplay (gtk+ for Windows)

Informations:

function HAL id:	203
variable type name:	uiInitDisplay
group:	gtk+ for Windows
description:	inits a display

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1  // =====
2  // function implementation for uiInitDisplay(var id =
3  // 203)
4  // description: inits a display
5  // =====
6  void HALfunc_ID203_uiInitDisplay(void * pIPcpu,
7  TuAPInterpreterFunctionParameter * pParams) {
8  // parameter uiUUID descr: uuid of the panel
9  int32_t uiUUID = pParams[0].fp_integer;
10 // parameter b descr: display
11 TAPgtkUIvector * puiD = (TAPgtkUIvector *)
12 pParams[1].fp_pD;
13 // parameter bIndex descr: display index
14 int32_t dIndex = pParams[2].fp_integer;
15 // parameter p descr: panel
16 TAPgtkUIvector * puiP = (TAPgtkUIvector *)
17 pParams[3].fp_pD;
18 // parameter pIndex descr: panel index
19 int32_t pIndex = pParams[4].fp_integer;
20
21     if (APgtkUI_createUI (
22         &(puiD->pUI[dIndex]),
23         &(puiP->pUI[pIndex]),
24         uiUUID,

```

```

22         eAPgtkUItype_display
23     )
24     {
25         ((TAPInterpreterCPU *)pIPcpu)->EF = -10;
26     }
27     ((TAPInterpreterCPU *)pIPcpu)->pIP++;
28 };
29

```

3.3.56 uiInitLED (gtk+ for Windows)

Informations:

function HAL id:	204
variable type name:	uiInitLED
group:	gtk+ for Windows
description:	inits a LED

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1  // =====
2  // function implementation for uiInitLED(var id = 204)
3  // description: inits a LED
4  // =====
5
6  void HALfunc_ID204_uiInitLED(void * pIPcpu,
7      TuAPIInterpreterFunctionParameter * pParams) {
8      // parameter uiUUID descr: uuid of the LED
9      int32_t uiUUID = pParams[0].fp_integer;
10     // parameter l descr: LED
11     TAPgtkUIvector * puiL = (TAPgtkUIvector *)
12         pParams[1].fp_pD;
13     // parameter lIndex descr: LED index
14     int32_t lIndex = pParams[2].fp_integer;
15     // parameter p descr: panel
16     TAPgtkUIvector * puiP = (TAPgtkUIvector *)
17         pParams[3].fp_pD;
18     // parameter pIndex descr: panel index
19     int32_t pIndex = pParams[4].fp_integer;
20
21     if (APgtkUI_createUI (
22         &(puiL->pUI[lIndex]),
23         &(puiP->pUI[pIndex]),
24         uiUUID,
25         eAPgtkUItype_LED
26     )
27     ) {
28         ((TAPInterpreterCPU *)pIPcpu)->EF = -10;
29     }
30 }

```

```

26     }
27
28     ((TAPInterpreterCPU *)pIPcpu)->pIP++;
29 };

```

3.3.57 uiCheckButtonPressed (gtk+ for Windows)

Informations:

function HAL id:	210
variable type name:	uiCheckButtonPressed
group:	gtk+ for Windows
description:	check button state

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1  // =====
2  // function implementation for uiCheckButtonPressed(var
   id = 210)
3  // description: if the button was pressed the CF is set
4  // =====
5
6  void HALfunc_ID210_uiCheckButtonPressed(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {
7  // parameter b descr: button
8      TAPgtkUIvector * puiB = (TAPgtkUIvector *)
   pParams[0].fp_pD;
9  // parameter bIndex descr: button index
10     int32_t bIndex = pParams[1].fp_integer;
11 // parameter p descr: panel
12
13     if (puiB->pUI[bIndex].ui.button.pressCounter) {
14         puiB->pUI[bIndex].ui.button.pressCounter
           --;
15         ((TAPInterpreterCPU *)pIPcpu)->CF = 1;
16     } else {
17         ((TAPInterpreterCPU *)pIPcpu)->CF = 0;
18     }
19     ((TAPInterpreterCPU *)pIPcpu)->pIP++;
20 };

```

3.3.58 uiSetLED (gtk+ for Windows)

Informations:

function HAL id:	211
variable type name:	uiSetLED
group:	gtk+ for Windows
description:	set LED state

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```
1 // =====
2 // function implementation for uiSetLED(var id = 211)
3 // description: set the LED state
4 // =====
5
6 void HALfunc_ID211_uiSetLED(void * pIPcpu,
7     TuAPIInterpreterFunctionParameter * pParams) {
8     // parameter l descr: LED
9     TAPgtkUIvector * puiL = (TAPgtkUIvector *)
10         pParams[0].fp_pD;
11     // parameter lIndex descr: LED index
12     int32_t lIndex = pParams[1].fp_integer;
13     // parameter onFlag descr: if the flag is not zero the
14     // LED is turned on
15     int32_t onFlag = pParams[2].fp_integer;
16
17     puiL->pUI[lIndex].ui.led.onFlag = (!onFlag) ? 0
18         : 1;
19     APgtkUI_redrawUI(&(puiL->pUI[lIndex]));
20     ((TAPIInterpreterCPU *)pIPcpu)->pIP++;
21 }
```

3.3.59 uiSetDisplay (gtk+ for Windows)

Informations:

function HAL id:	212
variable type name:	uiSetDisplay
group:	gtk+ for Windows
description:	set display text

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====
2 // function implementation for uiSetDisplay(var id =
   212)
3 // description: set the text of a display
4 // =====
5
6 void HALfunc_ID212_uiSetDisplay(void * pIPcpu,
   TuAPIInterpreterFunctionParameter * pParams) {
7 // parameter d descr: display
8     TAPgtkUIvector * puiD = (TAPgtkUIvector *)
       pParams[0].fp_pD;
9 // parameter dIndex descr: display index
10    int32_t dIndex = pParams[1].fp_integer;
11 // parameter s descr: the string
12    TAPstringVector * pSV = (TAPstringVector *)
       pParams[2].fp_pD;
13 // parameter iString descr: index of the string at the
   array
14    int32_t iString = pParams[3].fp_integer;
15
16    gtkAP_DisplaySetText (
17        &(puiD->pUI[dIndex].ui.display),
18        pSV->sv[iString].szTxt
19    );
20    ((TAPIInterpreterCPU *)pIPcpu)->pIP++;
21 };

```

3.3.60 setStringSize (ANSI C strings)

Informations:

function HAL id:	5
variable type name:	setStringSize
group:	ANSI C strings
description:	setup string

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====
2 // function implementation for setStringSize(var id = 5)
3 // description: resets the size of a string
4 // =====
5
6 void HALfunc_ID5_setStringSize(void * pIPcpu,
   TuAPIInterpreterFunctionParameter * pParams) {
7 // parameter s descr: the string

```

```

8      TAPstringVector * pSV = (TAPstringVector *)
        pParams[0].fp_pV;
9  // parameter i descr: index of the string at the array
10     int32_t i = pParams[1].fp_integer;
11  // parameter length descr: length of the string
12     int32_t length = pParams[2].fp_integer;
13
14     if (APstringVector_resize(pSV,(int) i, length))
15     {
16         ((TAPInterpreterCPU *)pIPcpu)->EF = -1;
17     } else {
18         ((TAPInterpreterCPU *)pIPcpu)->pIP++;
19     }
};

```

3.3.61 setStringValues (ANSI C strings)

Informations:

function HAL id:	6
variable type name:	setStringValues
group:	ANSI C strings
description:	set string chars

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1  // =====
2  // function implementation for setStringValues(var id =
   // 6)
3  // description: set the string
4  // =====
5
6  void HALfunc_ID6_setStringValues(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {
7  // parameter s descr: the string
8      TAPstringVector * pSV = (TAPstringVector *)
        pParams[0].fp_pV;
9  // parameter i descr: index of the string at the array
10     int i = (int) pParams[1].fp_integer;
11  // parameter p descr: position at the string where to
   start from
12     int p = (int) pParams[2].fp_integer;
13
14     char * pSrc =(char *) &pParams[3].fp_raw;
15
16     int pe = p + (dAPInterpreterFuncMaxParams-3) *
        sizeof(int32_t);
17

```

```

18     APstringVector_fill(pSV,i,p,pe,pSrc);
19
20     ((TAPInterpreterCPU *)pIPcpu)->pIP++;
21 };

```

3.3.62 concatStrings (ANSI C strings)

Informations:

function HAL id:	7
variable type name:	concatStrings
group:	ANSI C strings
description:	concat two strings s1 & s2 -> s1

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1  // =====
2  // function implementation for concatStrings(var id = 7)
3  // description: concat two strings
4  // =====
5  void HALfunc_ID7_concatStrings(void * pIPcpu,
6      TuAPIInterpreterFunctionParameter * pParams) {
7      TAPstringVector * s1 = (TAPstringVector *)
8          pParams[0].fp_pD;
9      // parameter i1 descr: index of the string 1
10     int i1 = (int) pParams[1].fp_integer;
11     // parameter s2 descr: the concat string
12     TAPstringVector * s2 = (TAPstringVector *)
13         pParams[2].fp_pD;
14     // parameter i2 descr: index of the string 2
15     int i2 = (int) pParams[3].fp_integer;
16
17     if (APstringVector_concat(s1,i1,s2,i2)) {
18         ((TAPInterpreterCPU *)pIPcpu)->EF = -1;
19     } else {
20         ((TAPInterpreterCPU *)pIPcpu)->pIP++;
21     }
22 };

```


3.3.63 rationalToString (ANSI C strings)

Informations:

function HAL id:	8
variable type name:	rationalToString
group:	ANSI C strings
description:	converts a rational to a string(size of the string is kept untouched)

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```
1 // =====
2 // function implementation for rationalToString(var id =
3 // 8)
4 // description: converts a rational to a string
5 // =====
6 void HALfunc_ID8_rationalToString(void * pIPcpu,
7     TuAPIInterpreterFunctionParameter * pParams) {
8     // parameter s descr: the string
9     TAPstringVector * s = (TAPstringVector *)
10         pParams[0].fp_pD;
11     // parameter sIndex descr: index of the string at the
12     // array
13     int sIndex = (int) pParams[1].fp_integer;
14     // parameter r descr: rational vector
15     TAPgenericRationalVector * r = (
16         TAPgenericRationalVector *) pParams[2].fp_pD;
17     // parameter rIndx descr: rational vector index
18     int rIndx = (int) pParams[3].fp_integer;
19
20     APstringVector_printFloat(s, sIndex, r->pVal[
21         rIndx]);
22
23     ((TAPIInterpreterCPU *)pIPcpu)->pIP++;
24 }
```

3.3.64 integerToString (ANSI C strings)

Informations:

function HAL id:	9
variable type name:	integerToString
group:	ANSI C strings
description:	converts a integer to a string(size of the string is kept untouched)

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====
2 // function implementation for integerToString(var id =
  9)
3 // description: converts an integer to a string
4 // =====
5 void HALfunc_ID9_integerToString(void * pIPcpu,
  TuAPIInterpreterFunctionParameter * pParams) {
6 // parameter s descr: the string
7     TAPstringVector * s = (TAPstringVector *)
      pParams[0].fp_pD;
8 // parameter sIndex descr: index of the string at the
  array
9     int sIndex = (int) pParams[1].fp_integer;
10 // parameter i descr: integer vector
11     TAPgenericIntegerVector * i = (
      TAPgenericIntegerVector *) pParams[2].fp_pD;
12 // parameter iIndex descr: integer vector indx
13     int iIndex = (int) pParams[3].fp_integer;
14
15     APstringVector_printInt(s, sIndex, i->pVal[
      iIndex]);
16
17     ((TAPIInterpreterCPU *)pIPcpu)->pIP++;
18 };

```

3.3.65 assignString (ANSI C strings)

Informations:

function HAL id:	10
variable type name:	assignString
group:	ANSI C strings
description:	assigns a string to an other

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====
2 // function implementation for assignString(var id = 10)
3 // description: assigns a string to an other
4 // =====

```

```

5 void HALfunc_ID10_assignString(void * pIPcpu,
  TuAPIInterpreterFunctionParameter * pParams) {
6 // parameter s1 descr: the string at its end the other
  string is concated
7     TAPstringVector * s1 = (TAPstringVector *)
      pParams[0].fp_pD;
8 // parameter i1 descr: index of the string 1
9     int i1 = (int) pParams[1].fp_integer;
10 // parameter s2 descr: the concat string
11     TAPstringVector * s2 = (TAPstringVector *)
      pParams[2].fp_pD;
12 // parameter i2 descr: index of the string 2
13     int i2 = (int) pParams[3].fp_integer;
14
15     if (APstringVector_assign(s1,i1,s2,i2)) {
16         ((TAPIInterpreterCPU *)pIPcpu)->EF = -1;
17     } else {
18         ((TAPIInterpreterCPU *)pIPcpu)->pIP++;
19     }
20 }
21 };

```

3.3.66 initBiquadAsHP (biquad filters (generic))

Informations:

function HAL id:	100
variable type name:	initBiquadAsHP
group:	biquad filters (generic)
description:	inits a biquad at the cascade as high pass filter

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====
2 // function implementation for initBiquadAsHP(var id =
  100)
3 // description: inits a biquad filter as an high pass
  filter
4 // =====
5
6 void HALfunc_ID100_initBiquadAsHP(void * pIPcpu,
  TuAPIInterpreterFunctionParameter * pParams) {
7 // parameter bq descr: biquad cascade
8     TBQF_BiquadCascade * pbqc = ((
      TAPIInterpreterVariable *) pParams[0].fp_pV)->
      pData;
9 // parameter index descr: index at the cascade
10    int32_t index = pParams[1].fp_integer;

```

```

11 // parameter fs descr: sample frequency
12     float fs = *((TAPgenericRationalVector *)
13         pParams[2].fp_pD)->pVal;
14 // parameter fc descr: cut off frequency
15     float fc = *((TAPgenericRationalVector *)
16         pParams[3].fp_pD)->pVal;
17
18     if (BQF_BQFcascadeInitHP(pbqc, (unsigned int)
19         index, fs, fc)) {
20         ((TAPInterpreterCPU *)pIPcpu)->EF =
21             -100;
22     } else {
23         ((TAPInterpreterCPU *)pIPcpu)->pIP++;
24     }
25 };

```

3.3.67 initBiquadAsLP (biquad filters (generic))

Informations:

function HAL id:	101
variable type name:	initBiquadAsLP
group:	biquad filters (generic)
description:	inits a biquad at the cascade as low pass filter

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====
2 // function implementation for initBiquadAsLP(var id =
3 // 101)
4 // description: inits a biquad filter as a low pass
5 // filter
6 // =====
7 void HALfunc_ID101_initBiquadAsLP(void * pIPcpu,
8     TuAPInterpreterFunctionParameter * pParams) {
9     // parameter bq descr: biquad cascade
10     TBQF_BiquadCascade * pbqc = ((
11         TAPInterpreterVariable *) pParams[0].fp_pV)->
12         pData;
13 // parameter index descr: index at the cascade
14     int32_t index = pParams[1].fp_integer;
15 // parameter fs descr: sample frequency
16     float fs = *((TAPgenericRationalVector *)
17         pParams[2].fp_pD)->pVal;
18 // parameter fc descr: cut off frequency
19     float fc = *((TAPgenericRationalVector *)
20         pParams[3].fp_pD)->pVal;

```

```

15         if (BQF_BQFcascadeInitLP(pbqc, (unsigned int)
16             index, fs, fc)) {
17             ((TAPInterpreterCPU *)pIPcpu)->EF =
18                 -101;
19         } else {
20             ((TAPInterpreterCPU *)pIPcpu)->pIP++;
        }
    };

```

3.3.68 initBiquadAsPeakFilter (biquad filters (generic))

Informations:

function HAL id:	102
variable type name:	initBiquadAsPeakFilter
group:	biquad filters (generic)
description:	inits a biquad at the cascade as boost/cut peak filter

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1  // =====
2  // function implementation for initBiquadAsPeakFilter(
3  //     var id = 102)
4  // description: inits a biquad filter as peak filter
5  // =====
6  void HALfunc_ID102_initBiquadAsPeakFilter(void * pIPcpu,
7      TuAPInterpreterFunctionParameter * pParams) {
8  // parameter bq descr: biquad cascade
9      TBQF_BiquadCascade * pbqc = ((
10         TAPInterpreterVariable *) pParams[0].fp_pV)->
11         pData;
12 // parameter index descr: index at the cascade
13     int32_t index = pParams[1].fp_integer;
14 // parameter fs descr: sample frequency
15     float fs = *((TAPgenericRationalVector *)
16         pParams[2].fp_pD)->pVal;
17 // parameter fc descr: center frequency
18     float fc = *((TAPgenericRationalVector *)
19         pParams[3].fp_pD)->pVal;
20 // parameter q descr: quality
21     float q = *((TAPgenericRationalVector *) pParams
22         [4].fp_pD)->pVal;
23 // parameter g descr: gain (not in dB)
24     float g = *((TAPgenericRationalVector *) pParams
25         [5].fp_pD)->pVal;
26
27     if (BQF_BQFcascadeInitPeak(pbqc, (unsigned int)
28         index, fs, fc, q, g)) {

```

```

20         ((TAPInterpreterCPU *)pIPcpu)->EF =
           -102;
21     } else {
22         ((TAPInterpreterCPU *)pIPcpu)->pIP++;
23     }
24 };

```

3.3.69 initBiquadAsLowFreqShelvFilter (biquad filters (generic))

Informations:

function HAL id:	103
variable type name:	initBiquadAsLowFreqShelvFilter
group:	biquad filters (generic)
description:	inits a biquad at the cascade as low frequency boost/cut shelving filter

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1  // =====
2  // function implementation for
   initBiquadAsLowFreqShelvFilter(var id = 103)
3  // description: inits a biquad filter as low frequency
   shelving filter
4  // =====
5  void HALfunc_ID103_initBiquadAsLowFreqShelvFilter(void *
   pIPcpu, TuAPInterpreterFunctionParameter * pParams)
   {
6  // parameter bq descr: biquad cascade
7  //     TBQF_BiquadCascade * pbqc = ((
   TAPInterpreterVariable *) pParams[0].fp_pV)->
   pData;
8  // parameter index descr: index at the cascade
9  //     int32_t index = pParams[1].fp_integer;
10 // parameter fs descr: sample frequency
11 //     float fs = *((TAPgenericRationalVector *)
   pParams[2].fp_pD)->pVal;
12 // parameter f descr: cut/boost frequency
13 //     float f = *((TAPgenericRationalVector *) pParams
   [3].fp_pD)->pVal;
14 // parameter q descr: quality
15 //     float q = *((TAPgenericRationalVector *) pParams
   [4].fp_pD)->pVal;
16 // parameter g descr: gain (not in dB)
17 //     float g = *((TAPgenericRationalVector *) pParams
   [5].fp_pD)->pVal;

```

```

18
19         if (BQF_BQFcascadeInitLowFreqShelving(pbqc, (
20             unsigned int) index, fs, f, q, g)) {
21             ((TAPInterpreterCPU *)pIPcpu)->EF =
22                 -103;
23         } else {
24             ((TAPInterpreterCPU *)pIPcpu)->pIP++;
25         }
26     };

```

3.3.70 initBiquadAsHighFreqShelvFilter (biquad filters (generic))

Informations:

function HAL id:	104
variable type name:	initBiquadAsHighFreqShelvFilter
group:	biquad filters (generic)
description:	inits a biquad at the cascade as low frequency boost/cut shelving filter

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====
2 // function implementation for
3 //   initBiquadAsHighFreqShelvFilter(var id = 104)
4 // description: inits a biquad filter as high frequency
5 // shelving filter
6 // =====
7 void HALfunc_ID104_initBiquadAsHighFreqShelvFilter(void
8     * pIPcpu, TuAPInterpreterFunctionParameter * pParams)
9 {
10     // parameter bq descr: biquad cascade
11     TBQF_BiquadCascade * pbqc = ((
12         TAPInterpreterVariable *) pParams[0].fp_pV)->
13         pData;
14     // parameter index descr: index at the cascade
15     int32_t index = pParams[1].fp_integer;
16     // parameter fs descr: sample frequency
17     float fs = *((TAPgenericRationalVector *)
18         pParams[2].fp_pD)->pVal;
19     // parameter f descr: cut/boost frequency
20     float f = *((TAPgenericRationalVector *) pParams
21         [3].fp_pD)->pVal;
22     // parameter q descr: quality
23     float q = *((TAPgenericRationalVector *) pParams
24         [4].fp_pD)->pVal;

```

```

16 // parameter g descr: gain (not in dB)
17 float g = *((TAPgenericRationalVector *) pParams
           [5].fp_pD)->pVal;
18
19 if (BQF_BQFcascadeInitHighFreqShelving(pbqc, (
           unsigned int) index, fs, f, q, g)) {
20     ((TAPInterpreterCPU *)pIPcpu)->EF =
           -104;
21 } else {
22     ((TAPInterpreterCPU *)pIPcpu)->pIP++;
23 }
24 };

```

3.3.71 convoluteBiquad (biquad filters (generic))

Informations:

function HAL id:	110
variable type name:	convoluteBiquad
group:	biquad filters (generic)
description:	convolutes a vector of samples with a biquad cascade

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====
2 // function implementation for convoluteBiquad(var id =
   110)
3 // description: convolute biquad with an input and
   generate an output
4 // =====
5 void HALfunc_ID110_convoluteBiquad(void * pIPcpu,
   TuAPIInterpreterFunctionParameter * pParams) {
6 // parameter x descr: input
7     TAPgenericRationalVector * pXvec = (
           TAPgenericRationalVector *) pParams[0].fp_pD;
8 // parameter bqa descr: biquad cascade
9     TBQF_BiquadCascade * pbqc = (TBQF_BiquadCascade
           *) pParams[1].fp_pD;
10 // parameter y descr: output
11     TAPgenericRationalVector * pYvec = (
           TAPgenericRationalVector *) pParams[2].fp_pD;
12
13     BQF_BQFcascadeConvolute(pbqc, pXvec->pVal, pXvec
           ->num, pYvec->pVal);
14
15     ((TAPInterpreterCPU *)pIPcpu)->pIP++;
16 };

```


3.3.72 initNoisegate (audio dynamic processing (generic))

Informations:

function HAL id:	111
variable type name:	initNoisegate
group:	audio dynamic processing (generic)
description:	inits a noisegate

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```
1 // =====
2 // function implementation for initNoisegate(var id =
  111)
3 // description: initialize a noisegate
4 // =====
5 void HALfunc_ID111_initNoisegate(void * pIPcpu,
  TuAPIInterpreterFunctionParameter * pParams) {
6 // parameter ng descr: noisegate
7     TDynProc_Noisegate * pNG = (TDynProc_Noisegate
  *) pParams[0].fp_pD;
8 // parameter rmsTAV descr: time average value for the
  rms
9     float rmsTAV = pParams[1].fp_rational;
10 // parameter AT descr: attack value for the smoothing
11     float AT = pParams[2].fp_rational;
12 // parameter RT descr: release value for the smoothing
13     float RT = pParams[3].fp_rational;
14 // parameter NT descr: noise cut off threshold
15     float NT = pParams[4].fp_rational;
16 // parameter NS descr: slope
17     float NS = pParams[5].fp_rational;
18
19     DynProc_InitNoisegate(pNG, rmsTAV, AT, RT, NT,
  NS);
20     ((TAPIInterpreterCPU *)pIPcpu)->pIP++;
21 };
```

3.3.73 initExpander (audio dynamic processing (generic))

Informations:

function HAL id:	112
variable type name:	initExpander
group:	audio dynamic processing (generic)
description:	inits a expander

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====
2 // function implementation for initExpander(var id =
  112)
3 // description: initialize a expander
4 // =====
5 void HALfunc_ID112_initExpander(void * pIPcpu,
  TuAPIInterpreterFunctionParameter * pParams) {
6 // parameter exp descr: expander
7     TDynProc_Expander * pExp = (TDynProc_Expander *)
      pParams[0].fp_pD;
8 // parameter rmsTAV descr: time average value for the
  rms
9     float rmsTAV = pParams[1].fp_rational;
10 // parameter AT descr: attack value for the smoothing
11     float AT = pParams[2].fp_rational;
12 // parameter RT descr: release value for the smoothing
13     float RT = pParams[3].fp_rational;
14 // parameter ET descr: expander threshold
15     float ET = pParams[4].fp_rational;
16 // parameter ES descr: slope
17     float ES = pParams[5].fp_rational;
18
19     DynProc_InitExpander(pExp, rmsTAV, AT, RT, ET,
      ES);
20     ((TAPIInterpreterCPU *)pIPcpu)->pIP++;
21 };

```

3.3.74 initCompressor (audio dynamic processing (generic))

Informations:

function HAL id:	113
variable type name:	initCompressor
group:	audio dynamic processing (generic)
description:	inits a compressor

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====
2 // function implementation for initCompressor(var id =
  113)

```

```

3 // description: initialize a compressor
4 // =====
5 void HALfunc_ID113_initCompressor(void * pIPcpu,
6     TuAPIInterpreterFunctionParameter * pParams) {
7     // parameter comp descr: compressor
8     TDynProc_Compressor * pComp = (
9         TDynProc_Compressor *) pParams[0].fp_pD;
10    // parameter rmsTAV descr: time average value for the
11    rms
12    float rmsTAV = pParams[1].fp_rational;
13    // parameter AT descr: attack value for the smoothing
14    float AT = pParams[2].fp_rational;
15    // parameter RT descr: release value for the smoothing
16    float RT = pParams[3].fp_rational;
17    // parameter CT descr: compressor threshold
18    float CT = pParams[4].fp_rational;
19    // parameter CS descr: slope
20    float CS = pParams[5].fp_rational;
21
22    DynProc_InitCompressor(pComp, rmsTAV, AT, RT, CT
23        , CS);
24    ((TAPIInterpreterCPU *)pIPcpu)->pIP++;
25 };

```

3.3.75 initLimiter (audio dynamic processing (generic))

Informations:

function HAL id:	114
variable type name:	initLimiter
group:	audio dynamic processing (generic)
description:	inits a limiter

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====
2 // function implementation for initLimiter(var id = 114)
3 // description: initialize a limiter
4 // =====
5 void HALfunc_ID114_initLimiter(void * pIPcpu,
6     TuAPIInterpreterFunctionParameter * pParams) {
7     // parameter lim descr: limiter
8     TDynProc_Limiter * pLim = (TDynProc_Limiter *)
9         pParams[0].fp_pD;
10    // parameter ATpeak descr: attack value for the peak
11    detection
12    float ATpeak = pParams[1].fp_rational;

```

```

10 // parameter RTpeak descr: release value for the peak
    detection
11     float RTpeak = pParams[2].fp_rational;
12 // parameter ATsmooth descr: attack value for the
    smoothing
13     float ATsmooth = pParams[3].fp_rational;
14 // parameter RTsmooth descr: release value for the
    smoothing
15     float RTsmooth = pParams[4].fp_rational;
16 // parameter LT descr: limiter threshold
17     float LT = pParams[5].fp_rational;
18 // parameter LS descr: slope
19     float LS = pParams[6].fp_rational;
20
21     DynProc_InitLimiter(pLim, ATpeak, RTpeak,
        ATsmooth, RTsmooth, LT, LS);
22     ((TAPInterpreterCPU *)pIPcpu)->pIP++;
23 };

```

3.3.76 calcNoisegate (audio dynamic processing (generic))

Informations:

function HAL id:	115
variable type name:	calcNoisegate
group:	audio dynamic processing (generic)
description:	stream samples through a noisegate

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====
2 // function implementation for calcNoisegate(var id =
    115)
3 // description: sends a stream of samples through a
    noisegate
4 // =====
5 void HALfunc_ID115_calcNoisegate(void * pIPcpu,
    TuAPIInterpreterFunctionParameter * pParams) {
6 // parameter x descr: input
7     TAPgenericRationalVector * x = (
        TAPgenericRationalVector *) pParams[0].fp_pD;
8 // parameter ng descr: noisegate
9     TDynProc_Noisegate * pNG = (TDynProc_Noisegate
        *) pParams[1].fp_pD;
10 // parameter y descr: output
11     TAPgenericRationalVector * y = (
        TAPgenericRationalVector *) pParams[2].fp_pD;
12

```

```

13     int i;
14     for (i = 0; i < x->num; i++) {
15         y->pVal[i] = DynProc_calcNoisegate(pNG,
16             x->pVal[i]);
17     }
18     ((TAPInterpreterCPU *)pIPcpu)->pIP++;
};

```

3.3.77 calcExpander (audio dynamic processing (generic))

Informations:

function HAL id:	116
variable type name:	calcExpander
group:	audio dynamic processing (generic)
description:	stream samples through a expander

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1  // =====
2  // function implementation for calcExpander(var id =
3  // 116)
4  // description: sends a stream of samples through a
5  // expander
6  // =====
7  void HALfunc_ID116_calcExpander(void * pIPcpu,
8  TuAPIInterpreterFunctionParameter * pParams) {
9  // parameter x descr: input
10     TAPgenericRationalVector * x = (
11         TAPgenericRationalVector *) pParams[0].fp_pD;
12 // parameter exp descr: expander
13     TDynProc_Expander * pExp = (TDynProc_Expander *)
14         pParams[1].fp_pD;
15 // parameter y descr: output
16     TAPgenericRationalVector * y = (
17         TAPgenericRationalVector *) pParams[2].fp_pD;
18
19     int i;
20     for (i = 0; i < x->num; i++) {
21         y->pVal[i] = DynProc_calcExpander(pExp,
22             x->pVal[i]);
23     }
24     ((TAPInterpreterCPU *)pIPcpu)->pIP++;
25 };

```

3.3.78 calcCompressor (audio dynamic processing (generic))

Informations:

function HAL id:	117
variable type name:	calcCompressor
group:	audio dynamic processing (generic)
description:	stream samples through a compressor

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```
1 // =====
2 // function implementation for calcCompressor(var id =
3 // 117)
4 // description: sends a stream of samples through a
5 // compressor
6 // =====
7 void HALfunc_ID117_calcCompressor(void * pIPcpu,
8     TuAPIInterpreterFunctionParameter * pParams) {
9     // parameter x descr: input
10    TAPgenericRationalVector * x = (
11        TAPgenericRationalVector *) pParams[0].fp_pD;
12    // parameter comp descr: compressor
13    TDynProc_Compressor * pComp = (
14        TDynProc_Compressor *) pParams[1].fp_pD;
15    // parameter y descr: output
16    TAPgenericRationalVector * y = (
17        TAPgenericRationalVector *) pParams[2].fp_pD;
18
19    int i;
20    for (i = 0; i < x->num; i++) {
21        y->pVal[i] = DynProc_calcCompressor(
22            pComp, x->pVal[i]);
23    }
24    ((TAPIInterpreterCPU *)pIPcpu)->pIP++;
25 }
```

3.3.79 calcLimiter (audio dynamic processing (generic))

Informations:

function HAL id:	118
variable type name:	calcLimiter
group:	audio dynamic processing (generic)
description:	stream samples through a limiter

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====
2 // function implementation for calcLimiter(var id = 118)
3 // description: sends a stream of samples through a
  limiter
4 // =====
5 void HALfunc_ID118_calcLimiter(void * pIPcpu,
  TuAPIInterpreterFunctionParameter * pParams) {
6 // parameter x descr: input
7     TAPgenericRationalVector * x = (
      TAPgenericRationalVector *) pParams[0].fp_pD;
8 // parameter lim descr: limiter
9     TDynProc_Limiter * pLim = (TDynProc_Limiter *)
      pParams[1].fp_pD;
10 // parameter y descr: output
11     TAPgenericRationalVector * y = (
      TAPgenericRationalVector *) pParams[2].fp_pD;
12
13     int i;
14     for (i = 0; i < x->num; i++) {
15         y->pVal[i] = DynProc_calcLimiter(pLim, x
          ->pVal[i]);
16     }
17     ((TAPIInterpreterCPU *)pIPcpu)->pIP++;
18 };

```

3.3.80 calcDelay (generic delay)

Informations:

function HAL id:	150
variable type name:	calcDelay
group:	generic delay
description:	sends a stream of values into the delay and reads them out of it

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====
2 // function implementation for calcDelay(var id = 150)
3 // description: shifts the content of the delay
4 // =====

```

```

5 void HALfunc_ID150_calcDelay(void * pIPcpu,
  TuAPIInterpreterFunctionParameter * pParams) {
6 // parameter x descr: input
7     TAPgenericRationalVector * x = (
      TAPgenericRationalVector *) pParams[0].fp_pD;
8 // parameter delay descr: the delay
9     TgenDelay * delay = (TgenDelay *) pParams[1].
      fp_pD;
10 // parameter y descr: output
11     TAPgenericRationalVector * y = (
      TAPgenericRationalVector *) pParams[2].fp_pD;
12
13     genDelay_readWrite(delay, x->pVal, y->pVal, y->
      num);
14     ((TAPIInterpreterCPU *)pIPcpu)->pIP++;
15 };

```

3.3.81 initDelay (generic delay)

Informations:

function HAL id:	151
variable type name:	initDelay
group:	generic delay
description:	inits the delay

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====
2 // function implementation for initDelay(var id = 151)
3 // description: inits the delay
4 // =====
5 void HALfunc_ID151_initDelay(void * pIPcpu,
  TuAPIInterpreterFunctionParameter * pParams) {
6 // parameter d descr: the delay
7     TgenDelay * d = (TgenDelay *) pParams[0].fp_pD;
8 // parameter N descr: number of samples which correspond
  to the delay time (Tdelay = N * Ta)
9     TAPgenericRationalVector * N = pParams[1].fp_pD;
10 // parameter Nindex descr: the index at the vector of N
11     int Nindex = (int) pParams[2].fp_integer;
12 // parameter readToWriteOffset descr: the distance in
  Samples between the read and write position
13     int readToWriteOffset = (int) pParams[3].
      fp_integer;
14
15     // resize delay
16     if (genDelay_resize(d, N->pVal[Nindex])) {

```



```

17         ((TAPInterpreterCPU *)pIPcpu)->EF = -1;
18         return;
19     }
20     // set pointer
21     genDelay_shuffle(d,EgenDelayRWflag_readPointer,
22                     readToWriteOffset);
23     // inc IP
24     ((TAPInterpreterCPU *)pIPcpu)->pIP++;
25 };

```

3.4 message system drivers

3.4.1 driver 1 (ADSP 21369 blockbased, ADC in, DAC3 & DAC4 out, UART @9600,n,8,1)

Informations:

description:	ADSPuartDRV	
includes:		
c-Include	c-Library	system lib
AP.h		no

code:

```

1  // =====
2  // driver uuid =2
3  // =====
4
5  // INFO
6  // the ADSP internal dataformat is BIG endian
7  // the UART send function sends data in little endian
8  // thats why we use the BIG endian function
9
10 // -----
11 // drv own functions
12 // -----
13
14 // struct for the drv data
15 typedef struct SADSPuartDrv {
16     TAPMsgHeader                                h;
17     TAPReceiveStateMachine rxSM;
18     //!< receive state machine
19 } TADSPuartDrv;
20
21 TADSPuartDrv gADSPuartDrv;
22
23 // -----
24 // drv own functions
25 // -----

```

```

25
26 void drv_2_sendHeader (
27     uint32_t msgId,
28     uint32_t recv,
29     uint32_t num,
30     uint32_t length
31 ) {
32     gADSPuartDrv.h[eAPMsgHeaderPosition_msgTypeID] =
33         msgId;
34     gADSPuartDrv.h[eAPMsgHeaderPosition_msgNumber] =
35         num;
36     gADSPuartDrv.h[eAPMsgHeaderPosition_receiver] =
37         recv;
38     gADSPuartDrv.h[eAPMsgHeaderPosition_length] =
39         length;
40     sendUARTuintVectorBigEndian(gADSPuartDrv.h,
41         sizeof(TAPMsgHeader)/sizeof(uint32_t));
42 }
43
44 // call back function of the uart driver
45 int drv_2_cbAPClient (unsigned int d) {
46     return gADSPuartDrv.rxSM.state (&(gADSPuartDrv.
47         rxSM), &d, 1);
48 }
49
50 //open an existing driver and bind it to the AP
51 int drv_2_open (void * pAP, struct SAPMsgDrv *pDrv) {
52     // init header
53     gADSPuartDrv.h[eAPMsgHeaderPosition_endian] =
54         gAPendianFlag;
55     gADSPuartDrv.h[eAPMsgHeaderPosition_sender] = ((
56         TAP *) pAP)->nodeID;
57     gADSPuartDrv.h[eAPMsgHeaderPosition_receiver] =
58         dAPNodeID_ALL;
59     gADSPuartDrv.h[eAPMsgHeaderPosition_msgTypeID] =
60         0;
61     gADSPuartDrv.h[eAPMsgHeaderPosition_msgNumber] =
62         0;
63     gADSPuartDrv.h[eAPMsgHeaderPosition_length] = 0;
64
65     // save driver data
66     pDrv->pDrvData = &gADSPuartDrv;
67
68     // init receive state machine
69     APInitReceiveStateMachine(
70         &gADSPuartDrv.rxSM,
71         ((TAP *)pAP)->MS,
72         pDrv
73     );
74     return 0;
75 }
76
77 //close the driver
78 int drv_2_close (struct SAPMsgDrv *pDrv) {
79     return 0;
80 }

```

```

69 }
70 //destroys the driver
71 int drv_2_destroy (struct SAPMsgDrv *pDrv) {
72     return 0;
73 }
74
75 //send raw data
76 int drv_2_sendRaw (struct SAPMsgDrv *pDrv, int amount,
77     uint32_t *pData) {
78     sendUARTraw4ByteVector(pData,amount);
79     return 0;
80 }
81 //sends integer data in int32_t chunks in the sequence (
82     LB0,HB0,LB1,HB1,LB2,HB2,LB3,HB3)
83 int drv_2_sendInteger32 (struct SAPMsgDrv *pDrv, int
84     amount, int32_t * pData) {
85     sendUARTintVectorBigEndian(pData,amount);
86     return 0;
87 }
88 //sends float data in float chunks
89 int drv_2_sendFloat32 (struct SAPMsgDrv *pDrv, int
90     amount, float * pData) {
91     sendUARTFloatVector(pData,amount);
92     return 0;
93 }
94
95 //acknowledge
96 int drv_2_ACK (void *pDrvData, uint32_t receiver,
97     uint32_t mNum) {
98     drv_2_sendHeader(eAPMsgTypes_ACK, receiver, mNum
99         , 0);
100     return 0;
101 }
102 //not acknowledge
103 int drv_2_NACK (void *pDrvData, uint32_t receiver,
104     uint32_t mNum) {
105     drv_2_sendHeader(eAPMsgTypes_NACK, receiver,
106         mNum, 0);
107     return 0;
108 }
109 //start sending a program
110 int drv_2_startPrg (void *pDrvData, uint32_t receiver,
111     uint32_t mNum, int32_t globalVariableNumber, int32_t
112     localVariableNumber, int32_t instructionNumber) {
113     drv_2_sendHeader(eAPMsgTypes_startPrg, receiver,
114         mNum, 0);
115     sendUARTintVectorBigEndian(globalVariableNumber
116         ,1);
117     sendUARTintVectorBigEndian(localVariableNumber
118         ,1);
119     sendUARTintVectorBigEndian(instructionNumber,1);
120     return 0;

```

```

111 }
112
113 //sends a variable
114 int drv_2_sendVariable (void *pDrvData, uint32_t
    receiver, uint32_t mNum, int32_t index, int32_t
    varTypeID, int32_t num) {
115     drv_2_sendHeader(eAPMsgTypes_sendVariable,
        receiver, mNum, 3);
116     sendUARTintVectorBigEndian(&index,1);
117     sendUARTintVectorBigEndian(&varTypeID,1);
118     sendUARTintVectorBigEndian(&num,1);
119     return 0;
120 }
121 //sends an instruction
122 int drv_2_sendInstruction (void *pDrvData, uint32_t
    receiver, uint32_t mNum, int32_t index, uint32_t *
    fbc) {
123     drv_2_sendHeader(eAPMsgTypes_sendInstruction,
        receiver, mNum, dAPIInterpreterFuncMaxParams +
        1);
124     sendUARTintVectorBigEndian(&index,1);
125     sendUARTraw4ByteVector(fbc,sizeof(int32_t)*
        dAPIInterpreterFuncMaxParams);
126     return 0;
127 }
128 //sings that the program transmission has completed
129 int drv_2_endPrg (void *pDrvData, uint32_t receiver,
    uint32_t mNum) {
130     drv_2_sendHeader(eAPMsgTypes_endPrg, receiver,
        mNum, 0);
131     return 0;
132 }
133 //stops the AP
134 int drv_2_stop (void *pDrvData, uint32_t receiver,
    uint32_t mNum) {
135     drv_2_sendHeader(eAPMsgTypes_stop, receiver,
        mNum, 0);
136     return 0;
137 }
138 //the AP executes one instruction
139 int drv_2_step (void *pDrvData, uint32_t receiver,
    uint32_t mNum) {
140     drv_2_sendHeader(eAPMsgTypes_step, receiver,
        mNum, 0);
141     return 0;
142 }
143 //the AP runs the program
144 int drv_2_run (void *pDrvData, uint32_t receiver,
    uint32_t mNum) {
145     drv_2_sendHeader(eAPMsgTypes_run, receiver, mNum
        , 0);
146     return 0;
147 }
148 //a variable going to be updated
149 int drv_2_updateVariable (void *pDrvData, uint32_t

```

```

150     receiver, uint32_t mNum, int32_t gIndex, int32_t
        dataElements) {
151         drv_2_sendHeader(eAPMsgTypes_updateVariable,
            receiver, mNum, dataElements + 1);
152         sendUARTIntVectorBigEndian(&gIndex,1);
153         return 0;
154     }
155     //a AP is going to be logged in to the system
156     int drv_2_login (void *pDrvData, uint32_t receiver,
        uint32_t mNum) {
157         drv_2_sendHeader(eAPMsgTypes_login, receiver,
            mNum, 0);
158         return 0;
159     }
160     //a AP is going to be logged out of the system
161     int drv_2_logout (void *pDrvData, uint32_t receiver,
        uint32_t mNum) {
162         drv_2_sendHeader(eAPMsgTypes_logout, receiver,
            mNum, 0);
163         return 0;
164     }

```

3.4.2 driver 2 (MSP430-169STK)

Informations:

description:	MSP430uartDRV	
includes:		
c-Include	c-Library	system lib
AP.h		no

code:

```

1  // =====
2  // driver uuid =3
3  // =====
4
5  // struct for the drv data
6  typedef struct SMSP430uartDrv {
7      TAPMsgHeader                                     h;
8      TAPReceiveStateMachine                          rxSM;
9      //!< transmit header template
10     //!< receive state machine
11 } TMSP430uartDrv;
12
13 TMSP430uartDrv gMSP430uartDrv;
14
15 // -----
16 // drv own functions
17 // -----
18 void drv_3_sendHeader (

```

```

18         uint32_t msgId,
19         uint32_t recv,
20         uint32_t num,
21         uint32_t length
22     ) {
23         gMSP430uartDrv.h[eAPMsgHeaderPosition_msgTypeID]
24             = msgId;
25         gMSP430uartDrv.h[eAPMsgHeaderPosition_msgNumber]
26             = num;
27         gMSP430uartDrv.h[eAPMsgHeaderPosition_receiver]
28             = recv;
29         gMSP430uartDrv.h[eAPMsgHeaderPosition_length] =
30             length;
31         msp430_UART_send((unsigned char *)gMSP430uartDrv
32             .h,sizeof(TAPMsgHeader));
33     }
34
35     // function for feeding the recv state machine
36     void drv_3_feedRecvStateM () {
37         uint32_t d;
38         while (!stjFIFO_readElement(&gMsp430_uartFIFO ,&
39             d)) {
40             gMSP430uartDrv.rxSM.state (&
41                 gMSP430uartDrv.rxSM), &d, 1);
42         }
43     }
44
45     //open an existing driver and bind it to the AP
46     int drv_3_open (void * pAP, struct SAPMsgDrv *pDrv) {
47         // init header
48         gMSP430uartDrv.h[eAPMsgHeaderPosition_endian] =
49             gAPendianFlag;
50         gMSP430uartDrv.h[eAPMsgHeaderPosition_sender] =
51             ((TAP *) pAP)->nodeID;
52         gMSP430uartDrv.h[eAPMsgHeaderPosition_receiver]
53             = dAPNodeID_ALL;
54         gMSP430uartDrv.h[eAPMsgHeaderPosition_msgTypeID]
55             = 0;
56         gMSP430uartDrv.h[eAPMsgHeaderPosition_msgNumber]
57             = 0;
58         gMSP430uartDrv.h[eAPMsgHeaderPosition_length] =
59             0;
60
61         // save driver data
62         pDrv->pDrvData = &gMSP430uartDrv;
63
64         // init receive state machine
65         APInitReceiveStateMachine(
66             &gMSP430uartDrv.rxSM,
67             ((TAP *)pAP)->MS,
68             pDrv
69         );
70         return 0;
71     }
72 }
73
74
75
76
77
78
79

```

```

60 //close the driver
61 int drv_3_close (struct SAPMsgDrv *pDrv) {
62     return 0;
63 }
64 //destroys the driver
65 int drv_3_destroy (struct SAPMsgDrv *pDrv) {
66     return 0;
67 }
68
69 //sends raw data in uint32_t chunks
70 int drv_3_sendRaw (struct SAPMsgDrv *pDrv, int amount,
71     uint32_t * pData) {
72     msp430_UART_send((unsigned char *)pData, amount*
73         sizeof(uint32_t));
74     return 0;
75 }
76 //sends integer data in int32_t chunks in the sequence (
77     LB0,HB0,LB1,HB1,LB2,HB2,LB3,HB3)
78 int drv_3_sendInteger32 (struct SAPMsgDrv *pDrv, int
79     amount, int32_t * pData) {
80     msp430_UART_send((unsigned char *)pData, amount*
81         sizeof(uint32_t));
82     return 0;
83 }
84 //sends float data in float chunks
85 int drv_3_sendFloat32 (struct SAPMsgDrv *pDrv, int
86     amount, float * pData) {
87     msp430_UART_send((unsigned char *)pData, amount*
88         sizeof(uint32_t));
89     return 0;
90 }
91 //acknowledge
92 int drv_3_ACK (void *pDrvData, uint32_t receiver,
93     uint32_t mNum) {
94     drv_3_sendHeader(eAPMsgTypes_ACK, receiver, mNum
95         , 0);
96     return 0;
97 }
98 //not acknowledge
99 int drv_3_NACK (void *pDrvData, uint32_t receiver,
100     uint32_t mNum) {
101     drv_3_sendHeader(eAPMsgTypes_NACK, receiver,
102         mNum, 0);
103     return 0;
104 }
105 //start sending a program
106 int drv_3_startPrg (void *pDrvData, uint32_t receiver,
107     uint32_t mNum, int32_t globalVariableNumber, int32_t
108     localVariableNumber, int32_t instructionNumber) {
109     drv_3_sendHeader(eAPMsgTypes_startPrg, receiver,
110         mNum, 0);
111     msp430_UART_send((unsigned char *)&
112         globalVariableNumber, sizeof(int32_t));
113     msp430_UART_send((unsigned char *)&

```

```

        localVariableNumber, sizeof(int32_t));
100     msp430_UART_send((unsigned char *)&
        instructionNumber, sizeof(int32_t));
101     return 0;
102 }
103
104 //sends a variable
105 int drv_3_sendVariable (void *pDrvData, uint32_t
    receiver, uint32_t mNum, int32_t index, int32_t
    varTypeID, int32_t num) {
106     drv_3_sendHeader(eAPMsgTypes_sendVariable,
        receiver, mNum, 3);
107     msp430_UART_send((unsigned char *)&index, sizeof(
        int32_t));
108     msp430_UART_send((unsigned char *)&varTypeID,
        sizeof(int32_t));
109     msp430_UART_send((unsigned char *)&num, sizeof(
        int32_t));
110     return 0;
111 }
112 //sends an instruction
113 int drv_3_sendInstruction (void *pDrvData, uint32_t
    receiver, uint32_t mNum, int32_t index, uint32_t *
    fbc) {
114     drv_3_sendHeader(eAPMsgTypes_sendInstruction,
        receiver, mNum, dAPIInterpreterFuncMaxParams +
        1);
115     msp430_UART_send((unsigned char *)&index, sizeof(
        int32_t));
116     msp430_UART_send((unsigned char *)fbc, sizeof(
        int32_t) * (dAPIInterpreterFuncMaxParams+1));
117     return 0;
118 }
119 //sings that the program transmission has completed
120 int drv_3_endPrg (void *pDrvData, uint32_t receiver,
    uint32_t mNum) {
121     drv_3_sendHeader(eAPMsgTypes_endPrg, receiver,
        mNum, 0);
122     return 0;
123 }
124 //stops the AP
125 int drv_3_stop (void *pDrvData, uint32_t receiver,
    uint32_t mNum) {
126     drv_3_sendHeader(eAPMsgTypes_stop, receiver,
        mNum, 0);
127     return 0;
128 }
129 //the AP executes one instruction
130 int drv_3_step (void *pDrvData, uint32_t receiver,
    uint32_t mNum) {
131     drv_3_sendHeader(eAPMsgTypes_step, receiver,
        mNum, 0);
132     return 0;
133 }
134 //the AP runs the program

```



```

135 int drv_3_run (void *pDrvData, uint32_t receiver,
136               uint32_t mNum) {
137     drv_3_sendHeader(eAPMsgTypes_run, receiver, mNum
138                     , 0);
139     return 0;
140 }
141 //a variable going to be updated
142 int drv_3_updateVariable (void *pDrvData, uint32_t
143 receiver, uint32_t mNum, int32_t gIndex, int32_t
144 dataElements) {
145     drv_3_sendHeader(eAPMsgTypes_updateVariable,
146 receiver, mNum, dataElements + 1);
147     msp430_UART_send((unsigned char *)&gIndex, sizeof
148 (int32_t));
149     return 0;
150 }
151 //a AP is going to be logged in to the system
152 int drv_3_login (void *pDrvData, uint32_t receiver,
153 uint32_t mNum) {
154     drv_3_sendHeader(eAPMsgTypes_login, receiver,
155 mNum, 0);
156     return 0;
157 }
158 //a AP is going to be logged out of the system
159 int drv_3_logout (void *pDrvData, uint32_t receiver,
160 uint32_t mNum) {
161     drv_3_sendHeader(eAPMsgTypes_logout, receiver,
162 mNum, 0);
163     return 0;
164 }
165 }

```

3.4.3 driver 3 (AP client interface useing stjSocket and APclient functions)

Informations:

description:	winAPdrv
--------------	----------

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====
2 // driver uuid =1
3 // =====
4 // -----
5 // drv own functions
6 // -----
7
8 #define dAPClientServerAdminPort (50000)

```

```

9  #define dAPClientInitBuffer (1024)
10
11 // struct for the drv data
12 typedef struct SAPTCPIPdrv {
13     TAPClient
14         cl;                                //!< tcp/ip client
15     TAPMsgHeader
16         ;                                //!< transmit header template
17     TAPReceiveStateMachine
18         rxSM;                            //!< receive state machine
19 } TAPTCPIPdrv;
20
21 // -----
22 // drv own functions
23 // -----
24
25 int drv_1_sendHeader (
26     TAPClient * pCl,
27     uint32_t * pH,
28     uint32_t msgId,
29     uint32_t recv,
30     uint32_t num,
31     uint32_t length
32 ) {
33     pH[eAPMsgHeaderPosition_msgTypeID] = msgId;
34     pH[eAPMsgHeaderPosition_msgNumber] = num;
35     pH[eAPMsgHeaderPosition_receiver] = recv;
36     pH[eAPMsgHeaderPosition_length] = length;
37     if (APclient_send(pCl, sizeof(TAPMsgHeader), (
38         uint8_t *)pH)) return -1;
39     return 0;
40 }
41
42 // call back function of the TCP/IP driver
43 int drv_1_cbAPClient (void *pvDC, uint16_t number,
44     uint8_t * pData) {
45     TAPReceiveStateMachine * pSM = pvDC;
46     return pSM->state (pSM, (uint32_t *) pData,
47         number / sizeof(uint32_t));
48 }
49
50 //open an existing driver and bind it to the AP
51 int drv_1_open (void * pAP, struct SAPMsgDrv *pDrv) {
52     TAPTCPIPdrv * pDC = NULL;
53
54     pDC = malloc(sizeof(TAPTCPIPdrv));
55     if (!pDC) return -1;
56     if (APclient_create(&(pDC->cl),
57         dAPClientServerAdminPort, dAPClientInitBuffer,
58         &(pDC->rxSM), drv_1_cbAPClient)) return -2;
59     // init header
60     pDC->txHeader[eAPMsgHeaderPosition_endian] = (
61         uint32_t) gAPendianFlag;
62     pDC->txHeader[eAPMsgHeaderPosition_sender] = ((
63         TAP *) pAP)->nodeID;

```

```

54     pDC->txHeader[eAPMsgHeaderPosition_receiver] =
        dAPNodeID_ALL;
55     pDC->txHeader[eAPMsgHeaderPosition_msgTypeID] =
        0;
56     pDC->txHeader[eAPMsgHeaderPosition_msgNumber] =
        0;
57     pDC->txHeader[eAPMsgHeaderPosition_length] = 0;
58
59     // save driver data
60     pDrv->pDrvData = pDC;
61
62     // init receive state machine
63     APInitReceiveStateMachine(
64         &pDC->rxSM,
65         ((TAP *)pAP)->MS,
66         pDrv
67     );
68     return 0;
69 }
70 //close the driver
71 int drv_1_close (struct SAPMsgDrv *pDrv) {
72     TAPTCPIPdrv * pDC = (TAPTCPIPdrv *) (pDrv->
        pDrvData);
73     if (pDC) {
74         APclient_close(&pDC->cl);
75     }
76     return 0;
77 }
78 //destroys the driver
79 int drv_1_destroy (struct SAPMsgDrv *pDrv) {
80     TAPTCPIPdrv * pDC = (TAPTCPIPdrv *) (pDrv->
        pDrvData);
81     if (pDC) {
82         free(pDC);
83         pDrv->pDrvData = NULL;
84     }
85     return 0;
86 }
87
88 //sends raw data
89 int drv_1_sendRaw (struct SAPMsgDrv *pDrv, int amount,
        uint32_t * pData) {
90     TAPTCPIPdrv * pDC = (TAPTCPIPdrv *) (pDrv->
        pDrvData);
91     return APclient_send(&(pDC->cl), amount*sizeof(
        uint32_t), (uint8_t *)pData);
92 }
93
94 //a AP is going to be logged out of the system
95 int drv_1_sendInteger32 (struct SAPMsgDrv *pDrv, int
        amount, int32_t * pData) {
96     TAPTCPIPdrv * pDC = (TAPTCPIPdrv *) (pDrv->
        pDrvData);
97
98     if (gAPendianFlag == eAP_littleEndian) {

```

```

99         return APclient_send(&(pDC->cl), amount*
        sizeof(uint32_t), (uint8_t *)pData);
100     } else {
101         int i;
102         int32_t v;
103         for (i = 0; i < amount; i++) {
104             v = *pData;
105             ApendianConversation32Bit((
                uint32_t *)&v,
                eAP_littleEndian);
106             if (APclient_send(&(pDC->cl),
                sizeof(int32_t), (uint8_t *)&v
            )) {
                return -1;
            }
            pData++;
        }
110     }
111     return 0;
112 }
113 }
114
115 //a AP is going to be logged out of the system
116 int drv_1_sendFloat32 (struct SAPMsgDrv *pDrv, int
    amount, float * pData) {
117     TAPTCPIPdrv * pDC = (TAPTCPIPdrv *) (pDrv->
        pDrvData);
118
119     return APclient_send(&(pDC->cl), amount*sizeof(
        float), (uint8_t *)pData);
120 }
121
122
123
124 //acknowledge
125 int drv_1_ACK (void *pDrvData, uint32_t receiver,
    uint32_t mNum) {
126     TAPClient * pCl = &((TAPTCPIPdrv *) (pDrvData))->
        cl;
127     uint32_t * pH = ((TAPTCPIPdrv *) (pDrvData))->
        txHeader;
128
129     return drv_1_sendHeader(pCl, pH, eAPMsgTypes_ACK
        , receiver, mNum, 0);
130 }
131 //not acknowledge
132 int drv_1_NACK (void *pDrvData, uint32_t receiver,
    uint32_t mNum) {
133     TAPClient * pCl = &((TAPTCPIPdrv *) (pDrvData))->
        cl;
134     uint32_t * pH = ((TAPTCPIPdrv *) (pDrvData))->
        txHeader;
135
136     return drv_1_sendHeader(pCl, pH,
        eAPMsgTypes_NACK, receiver, mNum, 0);
137 }

```

```

138 //start sending a program
139 int drv_1_startPrg (void *pDrvData, uint32_t receiver,
    uint32_t mNum, int32_t globalVariableNumber, int32_t
    localVariableNumber, int32_t instructionNumber) {
140     TAPClient * pCl = &((TAPTCPIPdrv *) (pDrvData))->
        cl;
141     uint32_t * pH = ((TAPTCPIPdrv *) (pDrvData))->
        txHeader;
142     if (
143         drv_1_sendHeader(pCl, pH,
            eAPMsgTypes_startPrg,
            receiver, mNum, 0) ||
144         APclient_send(pCl, sizeof(int32_t) &
            globalVariableNumber) ||
145         APclient_send(pCl, sizeof(int32_t) &
            localVariableNumber) ||
146         APclient_send(pCl, sizeof(int32_t) &
            instructionNumber)
147     ) return -10;
148     return 0;
149 }
150
151 //sends a variable
152 int drv_1_sendVariable (void *pDrvData, uint32_t
    receiver, uint32_t mNum, int32_t index, int32_t
    varTypeID, int32_t num) {
153     TAPClient * pCl = &((TAPTCPIPdrv *) (pDrvData))->
        cl;
154     uint32_t * pH = ((TAPTCPIPdrv *) (pDrvData))->
        txHeader;
155     if (
156         drv_1_sendHeader(pCl, pH,
            eAPMsgTypes_sendVariable,
            receiver, mNum, 3) ||
157         APclient_send(pCl, sizeof(int32_t) &
            index) ||
158         APclient_send(pCl, sizeof(int32_t) &
            varTypeID) ||
159         APclient_send(pCl, sizeof(int32_t) &
            num)
160     ) return -10;
161     return 0;
162 }
163 //sends an instruction
164 int drv_1_sendInstruction (void *pDrvData, uint32_t
    receiver, uint32_t mNum, int32_t index, uint32_t *
    fbc) {
165     TAPClient * pCl = &((TAPTCPIPdrv *) (pDrvData))->
        cl;
166     uint32_t * pH = ((TAPTCPIPdrv *) (pDrvData))->
        txHeader;
167     if (

```

```

168         drv_1_sendHeader(pCl, pH,
                           eAPMsgTypes_sendInstruction,
                           receiver, mNum,
                           dAPIInterpreterFuncMaxParams +
                           1) ||
169         APclient_send(pCl, sizeof(int32_t)
                        ),(uint8_t *)&index) ||
170         APclient_send(pCl, sizeof(int32_t)
                        ) * dAPIInterpreterFuncMaxParams
                           ,(uint8_t *)fbc)
171     ) return -10;
172     return 0;
173 }
174 //sings that the program transmission has completed
175 int drv_1_endPrg (void *pDrvData, uint32_t receiver,
                   uint32_t mNum) {
176     TAPClient * pCl = &((TAPTCPIPdrv *) (pDrvData))->
        cl;
177     uint32_t * pH = ((TAPTCPIPdrv *) (pDrvData))->
        txHeader;
178
179     return drv_1_sendHeader(pCl, pH,
                             eAPMsgTypes_endPrg, receiver, mNum, 0);
180 }
181 //stops the AP
182 int drv_1_stop (void *pDrvData, uint32_t receiver,
                 uint32_t mNum) {
183     TAPClient * pCl = &((TAPTCPIPdrv *) (pDrvData))->
        cl;
184     uint32_t * pH = ((TAPTCPIPdrv *) (pDrvData))->
        txHeader;
185
186     return drv_1_sendHeader(pCl, pH,
                             eAPMsgTypes_stop, receiver, mNum, 0);
187 }
188 //the AP executes one instruction
189 int drv_1_step (void *pDrvData, uint32_t receiver,
                 uint32_t mNum) {
190     TAPClient * pCl = &((TAPTCPIPdrv *) (pDrvData))->
        cl;
191     uint32_t * pH = ((TAPTCPIPdrv *) (pDrvData))->
        txHeader;
192
193     return drv_1_sendHeader(pCl, pH,
                             eAPMsgTypes_step, receiver, mNum, 0);
194 }
195 //the AP runs the program
196 int drv_1_run (void *pDrvData, uint32_t receiver,
                uint32_t mNum) {
197     TAPClient * pCl = &((TAPTCPIPdrv *) (pDrvData))->
        cl;
198     uint32_t * pH = ((TAPTCPIPdrv *) (pDrvData))->
        txHeader;
199
200     return drv_1_sendHeader(pCl, pH, eAPMsgTypes_run

```

```

201     , receiver, mNum, 0);
202 }
203 //a variable going to be updated
204 int drv_1_updateVariable (void *pDrvData, uint32_t
receiver, uint32_t mNum, int32_t gIndex, int32_t
dataElements) {
205     TAPClient * pCl = &((TAPTCPIPdrv *) (pDrvData))->
cl;
206     uint32_t * pH = ((TAPTCPIPdrv *) (pDrvData))->
txHeader;
207     if (
                drv_1_sendHeader(pCl, pH,
eAPMsgTypes_updateVariable,
receiver, mNum, dataElements
+ 1) ||
                APclient_send(pCl, sizeof(int32_t
), (uint8_t *)&gIndex)
                ) return -10;
208     return 0;
209 }
210 //a AP is going to be logged in to the system
211 int drv_1_login (void *pDrvData, uint32_t receiver,
uint32_t mNum) {
212     TAPClient * pCl = &((TAPTCPIPdrv *) (pDrvData))->
cl;
213     uint32_t * pH = ((TAPTCPIPdrv *) (pDrvData))->
txHeader;
214     return drv_1_sendHeader(pCl, pH,
eAPMsgTypes_login, receiver, mNum, 0);
215 }
216 //a AP is going to be logged out of the system
217 int drv_1_logout (void *pDrvData, uint32_t receiver,
uint32_t mNum) {
218     TAPClient * pCl = &((TAPTCPIPdrv *) (pDrvData))->
cl;
219     uint32_t * pH = ((TAPTCPIPdrv *) (pDrvData))->
txHeader;
220     return drv_1_sendHeader(pCl, pH,
eAPMsgTypes_logout, receiver, mNum, 0);
221 }
222
223
224
225

```

3.5 audio processor blueprints

3.5.1 audio processor blueprint 1 (libsndfile sample based)

Informations:

description:	a wavfile processing AP
--------------	-------------------------

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====
2 // AP uuid = 6
3 // =====
4
5 #define WAVsampleCache (65536)
6
7 // the wav modul global var
8 TStjWAVmodule gWAVModule;
9
10
11 // inits the AP
12 int APinit (
13
14             TAP *
15             pAP,
16             TAPNodeID
17             nodeID,
18             const TAPMsgDrv *      pDrvList
19             ,
20             const int
21             driverNumber ,
22             size_t
23             messagePoolSize ,
24             int
25             sysEndian
26
27 )
28 {
29     pAP->nodeID = nodeID;
30     pAP->pNodeList = NULL;
31     pAP->pDrvList = pDrvList;
32     pAP->sysEndian = sysEndian;
33     pAP->driverNumber = driverNumber;
34     pAP->msgSysMMU = AP_MMU_create(messagePoolSize);
35     pAP->IP = APInterpreterCreate(pAP);
36     pAP->MS = APMScreate (pAP->msgSysMMU,sysEndian);
37     pAP->msgNumber = 0;
38     pAP->APstate = eAPstate_idle;
39
40     if(
41         (!pAP->msgSysMMU) ||
42         (!pAP->IP) ||
43         (!pAP->MS)
44     ) return -1;
45     // login the ap to the message system
46
47     // init wav module
48     TStjWAVOpenInfo wavIOs[] = {
49         {1,"input.wav",1,44100,0,
50          WAVsampleCache},
51         {2,"bypass.wav",0,44100,0,

```



```

43         WAVsampleCache},
44         {3,"subbass.wav",0,44100,0,
45         WAVsampleCache},
46         {4,"lowerVoice.wav",0,44100,0,
47         WAVsampleCache},
48         {5,"upperVoice.wav",0,44100,0,
49         WAVsampleCache},
50         {6,"harmonics.wav",0,44100,0,
51         WAVsampleCache},
52     };
53
54     if (WAVmoduleInit (sizeof(wavIOs) / sizeof(
55         TStjWAVOpenInfo),wavIOs,&gWAVModule)) {
56         return -2;
57     }
58
59     return TX_login(pAP);
60 }
61
62 // deletes the AP
63 void APdelete (TAP * pAP)
64 {
65     // close wav module
66     WAVmoduleExit(&gWAVModule);
67
68     APMSdelete (pAP->MS);
69     APInterpreterDelete(pAP->IP);
70     AP_MMU_delete(pAP->msgSysMMU);
71 }
72
73 // find a node at the list
74 TAPNode * APfindNode(TAP * pAP, TAPNodeID nodeID) {
75     TAPNode * pN = pAP->pNodeList;
76     while (pN) {
77         if (pN->nodeID == nodeID) return pN;
78         pN = pN->pNext;
79     };
80     return NULL;
81 }
82
83 // adds a new node to the node list
84 int APaddNode(TAP * pAP, TAPNodeID newNodeID, const
85     TAPMsgDrv * pDrv) {
86     if (APfindNode(pAP,newNodeID)) return 1;
87     TAPNode * pN = (TAPNode *) malloc(sizeof(TAPNode
88     ));
89     if (!pN) return -1;
90     pN->nodeID = newNodeID;
91     pN->pDrv = pDrv;
92     pN->pNext = pAP->pNodeList;
93     pAP->pNodeList = pN;
94     return 0;
95 }
96
97 // removes a node from the node list

```

```

90 void APremoveNode(TAP * pAP, TAPNodeID nodeID){
91     TAPNode * pAntN = pAP->pNodeList; // antecessor
        node
92     TAPNode * pActN = pAP->pNodeList; // actual node
93
94     while (pActN) {
95         // compare node id's
96         if (pActN->nodeID == nodeID) {
97             // unchain
98
99             // check if we at the first
            position at the list
100         if (pAP->pNodeList == pAntN) {
101             // reset the pointer
102             pAP->pNodeList = pActN->
                pNext;
103         } else {
104             // set the antecessor
105             pAntN->pNext = pActN->
                pNext;
106         }
107         // free node
108         free(pActN);
109         // and abort
110         return;
111     }
112     // the actual element becomes the
        precessor element
113     pAntN = pActN;
114     pActN = pActN->pNext;
115 }
116
117 // get a new message number
118 unsigned int APgetNewMessageNumber (TAP *pAP) {
119     pAP->msgNumber++;
120     return pAP->msgNumber;
121 }
122
123 // find the driver associated with der nodeID
124 const TAPMsgDrv * APfindDrvBySenderId (TAP * pAP,
125     TAPNodeID node) {
126     TAPNode * pN = pAP->pNodeList;
127     while (pN) {
128         if (pN->nodeID == node) {
129             return pN->pDrv;
130         }
131         pN = pN->pNext;
132     }
133     return NULL;
134 }
135
136 // runs the AP
137 int APrun(TAP *pAP) {
138     pAP->APstate = eAPstate_run;

```

```

139         return 0;
140     }
141
142
143     typedef struct SAPrealMMUMemory {
144         int *
145             pData;                // the data
146             size_t
147             count;                // amount of data
148             elements
149         struct SAPrealMMUMemory *    pNext;
150             // next element
151         struct SAPrealMMUMemory *    pPrev;
152             // previous element
153     } TAPrealMMUMemory;
154
155     //the mmu type
156     typedef struct SAPrealMMU {
157         int *
158             memory;
159             // the memory bolck
160         TAPrealMMUMemory *    pStart;
161             // first element
162         TAPrealMMUMemory *    pEnd;
163             // second element
164         TAPrealMMUMemory *    pUnusedList;
165             // list with the unused elements
166         int *
167             pUnusedData;          // pointer to the
168             unused memory
169         size_t
170             elementsAvailable;    // amount of elements
171             witch are available without using the
172             garbage collector
173         size_t
174             totalAvailable;       // total amount of
175             free bytes
176     } TAPrealMMU;
177
178     // =====
179     // memory entry functions
180     // =====
181
182     // a little macro for unchaining an element
183     #define DMemoryEntryUnchain(pM) \
184         if (pM->pNext) pM->pNext->pPrev = pM->pPrev; \
185         if (pM->pPrev) pM->pPrev->pNext = pM->pNext
186
187     //creates an memory entry
188     TAPrealMMUMemory * MemoryEntry_create () {
189         TAPrealMMUMemory * pM = NULL;
190
191         pM = (TAPrealMMUMemory *) malloc(sizeof(
192             TAPrealMMUMemory));

```

```

177         if (!pM) return NULL;
178         pM->pData = NULL;
179         pM->count = 0;
180         pM->pNext = NULL;
181         pM->pPrev = NULL;
182
183         return pM;
184     }
185
186     //deletes an memory Entry
187     void MemoryEntry_delete (
188         TAPrealMMUMemory * pM    // the memory to
189                                   delete
190     )
191     {
192         // put the entry out of the chain
193         DMemoryEntryUnchain(pM);
194         // now we delete it
195         free(pM);
196     }
197
198     // =====
199     // mmu helper
200     // =====
201
202     //alloc if needed a new memory entry
203     TAPrealMMUMemory * MMU_helper_createMemoryEntry (
204         TAPrealMMU * pMMU    // MMU
205         structure to init
206     )
207     {
208         // check if we have to alloc a new memory entry
209         if (!pMMU->pUnusedList) return
210             MemoryEntry_create();
211         // no there is some left at the list
212         TAPrealMMUMemory * pM;
213         // take the first one
214         pM = pMMU->pUnusedList;
215         // reset the list
216         pMMU->pUnusedList = pM->pNext;
217         // now unchain the element (for sure)
218         DMemoryEntryUnchain(pM);
219         // set the element pointers
220         pM->pNext = NULL;
221         pM->pPrev = NULL;
222         return pM;
223     }
224
225     //the garbage collector
226     void MMU_helper_garbageCollector (
227         TAPrealMMU * pMMU    // MMU
228         structure to init
229     )
230     {
231         TAPrealMMUMemory * pM = pMMU->pStart;

```

```

228     int * pD = pMMU->memory;
229     while (pM) {
230         // check if we have to move the data
231         if (pD != pM->pData) {
232             // move the data
233             memmove(pD, pM->pData, pM->count *
                sizeof(int));
234         }
235         // reset the destination pointer
236         pD += pM->count;
237         pM = pM->pNext;
238     }
239     // compressing memory finished
240     // set the mmu vars new
241     pMMU->elementsAvailable = pMMU->totalAvailable;
242     pMMU->pUnusedData = pD;
243 }
244
245
246 // create a mmu
247 TAPMMU AP_MMU_create (size_t elementsNumber) {
248     TAPrealMMU * pMMU;
249
250     pMMU = (TAPrealMMU *) malloc (sizeof(TAPrealMMU)
251 );
252     if (!pMMU) return NULL;
253
254
255     // setup lists
256     pMMU->pStart = NULL;
257     pMMU->pEnd = NULL;
258     pMMU->pUnusedList = NULL;
259
260     pMMU->elementsAvailable = elementsNumber;
261     pMMU->pUnusedData = (int *) malloc (pMMU->
262         elementsAvailable);
263     pMMU->totalAvailable = pMMU->elementsAvailable;
264     return pMMU;
265 }
266
267 // destroying the mmu
268 void AP_MMU_delete (TAPMMU mmu) {
269     TAPrealMMU * pMMU = (TAPrealMMU *) mmu;
270
271     TAPrealMMUMemory * pM;
272     TAPrealMMUMemory * pMnext;
273
274     // 1. delete al mmu entry's
275     // 1.1 unused entry
276     pM = pMMU->pUnusedList;
277     while (pM) {
278         pMnext = pM->pNext;
279         MemoryEntry_delete(pM);

```

```

280         pM = pMnext;
281     }
282     pMMU->pUnusedList = NULL;
283     // 1.2 used blocks
284     pM = pMMU->pStart;
285     while (pM) {
286         pMnext = pM->pNext;
287         MemoryEntry_delete(pM);
288         pM = pMnext;
289     }
290     pMMU->pStart = NULL;
291     pMMU->pEnd = NULL;
292     // 2. delete mmu memory
293     free (pMMU->memory);
294
295
296 }
297 // getting memmory from the mmu
298 TAPMMUmemory AP_MMU_get (TAPMMU mmu, size_t elements) {
299     TAPrealMMU * pMMU = (TAPrealMMU *) mmu;
300
301     // check if there is enough space
302     if (pMMU->totalAvailable < elements) goto error;
303     // check if we have to use the garbage collector
304     if (pMMU->elementsAvailable < elements) {
305         // start garbage collector
306         MMU_helper_garbageCollector(pMMU);
307     }
308     // we have enough memory so let's allocate some
309
310     // get a new entry
311     TAPrealMMUmemory * pM;
312     pM = MMU_helper_createMemoryEntry(pMMU);
313     if (!pM) return NULL;
314     // get some memory
315     pM->pData = pMMU->pUnusedData;
316     pM->count = elements;
317     // refresh data
318     pMMU->pUnusedData += elements;
319     pMMU->totalAvailable -= elements;
320     pMMU->elementsAvailable -= elements;
321     // insert memory element at the end of the list
322     // and update last element
323     pM->pPrev = pMMU->pEnd;
324     if (pMMU->pEnd) pMMU->pEnd->pNext = pM;
325     if (!pMMU->pStart) pMMU->pStart = pM;
326     pMMU->pEnd = pM;
327
328     return pM;
329
330 error:
331
332     return NULL;
333 }
334 // free memmory from the mmu

```

```

334 void AP_MMU_free (TAPMMU mmu, TAPMMUmemory memory) {
335     TAPrealMMU * pMMU = (TAPrealMMU *) mmu;
336     TAPrealMMUMemory * pM = (TAPrealMMUMemory *)
        memory;
337
338     if (!pM) return;
339     // set mmu settings
340     if (pMMU->pStart == pM) pMMU->pStart = pM->pNext
        ;
341     if (pMMU->pEnd == pM) pMMU->pEnd = pM->pPrev;
342     // unchain element
343     DMemoryEntryUnchain (pM);
344     // and put it to the chain of unused
345     pM->pNext = pMMU->pUnusedList;
346     pM->pPrev = NULL;
347     if (pMMU->pUnusedList) {
348         pMMU->pUnusedList->pPrev = pM;
349     }
350     pMMU->pUnusedList = pM;
351     // now set the mmu data new
352     pMMU->totalAvailable += pM->count;
353
354 }
355
356 // getting access to the MMU data
357 void * AP_MMU_getData (TAPMMUmemory memory) {
358     TAPrealMMUMemory * pM = (TAPrealMMUMemory *)
        memory;
359     return pM->pData;
360 }
361
362 // the real interpreter
363 typedef struct SAPrealInterpreter {
364     int
        state;
        // state of the IP
365     int
        nextState;
        // the next state of the IP
366     TAPInterpreterCPU
        cpu;
        // the IP
        core
367     TAPInterpreterFuncCall *
        code;
        // the code
368     int32_t
        instructionCount;
        // number of
        instructions at the code
369     TAPInterpreterVariable *
        variables;
        // the variables
370     int32_t
        variableCount;
        // number of
        the variables
371     int
        sysEndian;
        // endian of the system

```

```

372 } TAPrealInterpreter;
373
374 // create a new interpreter
375 TAPInterpreter APIInterpreterCreate (void * pAP) {
376     TAPrealInterpreter * pIP = NULL;
377     pIP = (TAPrealInterpreter *) malloc (sizeof(
378         TAPrealInterpreter));
379     if (!pIP) return NULL;
380
381     pIP->state = eAPIInterpreterState_idle;
382     pIP->nextState = eAPIInterpreterState_idle;
383     pIP->sysEndian = ((TAP *)pAP)->sysEndian;
384     pIP->cpu.CF = 0;
385     pIP->cpu.EF = 0;
386     pIP->cpu.pCodeStart = NULL;
387     pIP->cpu.pCodeEnd = NULL;
388     pIP->cpu.pIP = NULL;
389
390     pIP->code = NULL;
391     pIP->instructionCount = 0;
392
393     pIP->variables = NULL;
394     pIP->variableCount = 0;
395
396     return pIP;
397 }
398
399 // cleans the interpreter
400 void APIInterpreterClean (TAPInterpreter IP) {
401     TAPrealInterpreter * pIP = (TAPrealInterpreter
402         *) IP;
403
404     // clean code
405     if (pIP->code) {
406         free (pIP->code);
407         pIP->code = NULL;
408     }
409     pIP->instructionCount = 0;
410
411     // clean variables
412     TAPInterpreterVariable * pV = pIP->variables;
413     int i;
414     for (i = 0; i < pIP->variableCount; i++) {
415         if (pV->pVI) pV->pVI->pFkt_delete(pV->
416             pData);
417         pV++;
418     }
419     if (pIP->variables) {
420         free (pIP->variables);
421         pIP->variables = NULL;
422     }
423     pIP->variableCount = 0;
424 }

```



```

424 // deletes the interpreter
425 void APIInterpreterDelete (TAPIInterpreter IP) {
426     TAPrealInterpreter * pIP = (TAPrealInterpreter
427         *) IP;
428     APIInterpreterClean(IP);
429     free (pIP);
430 }
431 int APIInterpreterStateRun(TAPIInterpreter IP) {
432     TAPrealInterpreter * pIP = (TAPrealInterpreter
433         *) IP;
434     TAPIInterpreterFuncCall * pFC;
435     // setup cpu
436     pIP->cpu.CF = 0;
437     pIP->cpu.EF = 0;
438     pIP->cpu.pIP = pIP->code;
439     pIP->cpu.pCodeStart = pIP->code;
440     pIP->cpu.pCodeEnd = pIP->code + pIP->
441         instructionCount;
442     // run code
443     while (eAPIInterpreterState_run == pIP->state) {
444         pFC = pIP->cpu.pIP;
445         // check if we reached the end of the
446         // code
447         if (pFC > pIP->cpu.pCodeEnd) {
448             return 0;
449         }
450         // execute command
451         pFC->pHALFkt (&(pIP->cpu), pFC->param);
452         // check error flags
453         if (pIP->cpu.EF) {
454             return -1;
455         }
456     }
457     return 1;
458 }
459 // process the actual state
460 int APIInterpreterProcessState(TAPIInterpreter IP){
461     TAPrealInterpreter * pIP = (TAPrealInterpreter
462         *) IP;
463     pIP->state = pIP->nextState;
464     int rc = 0;
465     switch (pIP->state) {
466         case eAPIInterpreterState_idle:
467             break;
468         case eAPIInterpreterState_loadProgramm:
469             break;
470         case eAPIInterpreterState_run:
471             rc = APIInterpreterStateRun(IP);
472             if (rc >= 0) pIP->state =
473 
```

```

474         eAPIInterpreterState_idle;
475         break;
476     case eAPIInterpreterState_oneStep:
477         break;
478     case eAPIInterpreterState_halt:
479         break;
480     default:
481         return -10;
482 }
483 return rc;
484 }
485
486 // set interpreter state
487 int APIInterpreterSetState (TAPIInterpreter IP, int
488     msgEndian, int32_t state) {
489     TAPrealInterpreter * pIP = (TAPrealInterpreter
490         *) IP;
491     if (msgEndian != pIP->sysEndian) {
492         APendianConversation32Bit((uint32_t *)&
493             state);
494     }
495     pIP->nextState = (int) state;
496     return 0;
497 }
498
499 // setup the interpreter for a new program
500 int APIInterpreterInitNewProgramm (TAPIInterpreter IP, int
501     msgEndian, int32_t instructionsNumber, int32_t
502     VariableNumber) {
503     TAPrealInterpreter * pIP = (TAPrealInterpreter
504         *) IP;
505     int i;
506
507     APIInterpreterClean (IP);
508
509     if (msgEndian != pIP->sysEndian) {
510         APendianConversation32Bit((uint32_t *)&
511             instructionsNumber);
512         APendianConversation32Bit((uint32_t *)&
513             VariableNumber);
514     }
515
516     pIP->code = (TAPIInterpreterFuncCall *) malloc(
517         sizeof(TAPIInterpreterFuncCall)*
518         instructionsNumber);
519     pIP->instructionCount = instructionsNumber;
520
521     pIP->variables = (TAPIInterpreterVariable *)
522         malloc(sizeof(TAPIInterpreterVariable) * (
523             VariableNumber));
524     for (i = 0; i < VariableNumber; i++) {
525         pIP->variables[i].pData = NULL;
526         pIP->variables[i].pVI = NULL;
527     }
528 }

```

```

516         pIP->variableCount = VariableNumber;
517
518         return 0;
519     }
520
521     // load a variable/~array to an index
522     int APIInterpreterLoadVar (TAPIInterpreter IP, int
        msgEndian, int32_t index, int32_t varTypeID, int32_t
        numberOfElements)
523     {
524         TAPrealInterpreter * pIP = (TAPrealInterpreter
            *) IP;
525         if (msgEndian != pIP->sysEndian) {
526             APendianConversation32Bit((uint32_t *)&
                index);
527             APendianConversation32Bit((uint32_t *)&
                varTypeID);
528         }
529
530         if ((index < 0) || (index > pIP->variableCount))
            return -1;
531
532         // set pointer to the runtime variable
533         TAPIInterpreterVariable * pRTV = pIP->variables +
            index;
534         THAL_Variable const * pV = HALfindVar(varTypeID)
            ;
535         if (!pV) return -2;
536
537         pRTV->pData = pV->pFkt_create((unsigned int)
            numberOfElements);
538         //if (!pRTV->pData) return -3;
539
540         pRTV->pVI = pV;
541         return 0;
542     }
543
544     // load a single Instruction to an index
545     int APIInterpreterLoadInstr (TAPIInterpreter IP, int
        msgEndian, int32_t index, int32_t * pRawInstr)
546     {
547         TAPrealInterpreter * pIP = (TAPrealInterpreter
            *) IP;
548         if (msgEndian != pIP->sysEndian) {
549             APendianConversation32Bit((uint32_t *)&
                index);
550         }
551         if ((index < 0) || (index > pIP->
            instructionCount)) return -1;
552         TAPIInterpreterFuncCall * pIFC = pIP->code +
            index;
553         memset (pIFC, 0, sizeof(TAPIInterpreterFuncCall))
            ;
554
555         // get function

```

```

556     int32_t fid = *pRawInstr;
557     if (msgEndian != pIP->sysEndian) {
558         APendianConversation32Bit((uint32_t *)&
                                   fid);
559     }
560     THALFunction const * pF = HALfindFunction(fid);
561     if (!pF) return -2;
562     pIFC->pHALFkt = pF->pfktHAL;
563
564     // convert parameters
565     pRawInstr++; // set to the first parameter
566     int i;
567     THALFunctionParam const * pP = pF->paramList.pL;
568     TuAPIInterpreterFunctionParameter * pIFP = pIFC->
        param;
569     for (i = 0; i < pF->paramList.number; i++) {
570         if (APconvertRawParamData (msgEndian,pIP
                                   ->sysEndian,pRawInstr,pP,pIFP,pIP->
                                   variables)) return -3;
571         pP++;
572         pRawInstr++;
573         pIFP++;
574     }
575     return 0;
576 }
577
578 typedef struct SAPrealMsgSystem {
579     TAPMsg * pOldRXMsg;
580         // pointer to the oldest received
581         messages
582     TAPMsg * pNewRXMsg;
583         // pointer to the newest received
584         messages
585     TAPMMU mmu;
586         // the mmu
587     int sysEndianness; // the system endianness
588     int messagecounter; // a counter for checkin if a
589         new message has been received
590
591 } TAPrealMsgSystem;
592
593 int SInitial (
594     void * pVoidSM,
595         // pointer to the
596         statemachine
597     uint32_t * pD,
598         // pointer to the
599         data
600     int number
601         // the number of data
602         elements

```

```

593     );
594
595     int SMdata (
596         void *                pVoidSM,
597         // pointer to the
598         statemachine
599         uint32_t *            pD,
600         // pointer to the
601         data
602         int                    number
603         // the number of data
604         elements
605     );
606
607     int SMmessageFinished (
608         void *                pVoidSM
609         // pointer to the statemachine
610     );
611
612     // create AP message system
613     TAPMsgSystem APMScreate (
614         TAPMMU                mmu,
615         // the mmu
616         int                    sysEndianness // the system
617         endianness
618     ) {
619         TAPrealMsgSystem * pMS = (TAPrealMsgSystem *)
620         malloc (sizeof(TAPrealMsgSystem));
621         if (!pMS) return NULL;
622         pMS->mmu = mmu;
623         pMS->sysEndianness = sysEndianness;
624         pMS->pOldRXMsg = NULL;
625         pMS->pNewRXMsg = NULL;
626         pMS->messagecounter = 0;
627
628         return pMS;
629     }
630
631     void APMSdelete (
632         TAPMsgSystem ms
633     ) {
634         TAPrealMsgSystem * pMS = (TAPrealMsgSystem *) ms
635         ;
636         free (pMS);
637     }
638
639     // frees a message from the message system
640     void APMSdeleteMsg (
641         TAPMsgSystem ms,
642         TAPMsg *      pM

```

```

636     ) {
637         TAPrealMsgSystem * pMS = (TAPrealMsgSystem *) ms
        ;
638         AP_MMU_free(pMS->mmu, pM->memory);
639     }
640
641     // get memory for a new message
642     TAPMsg * APMSgetNewMsg (
643         TAPrealMsgSystem *      pMsgSys,
644         int
        dataElementsNumber,
645         const TAPMsgDrv *      pDrv
        ) {
646         TAPMMUmemory m = AP_MMU_get (pMsgSys->mmu,
        sizeof(TAPMsg)/sizeof(int) +
        dataElementsNumber*sizeof(uint32_t)/sizeof(
        int));
647         if (!m) return NULL;
648
649         int * pRD = (int *) AP_MMU_getData(m);
650         TAPMsg * pM = (TAPMsg *) pRD;
651         pM->memory = m;
652         pM->extraData.pDrv = pDrv;
653         pM->pH = (TAPMsgHeader *)((int *) pRD + sizeof(
        TAPMsg)/sizeof(int));
654         pM->pData = (int *)pM->pH + sizeof(TAPMsgHeader)
        /sizeof(int);
655         pM->pNext = NULL;
656         return pM;
657     }
658
659     // insert a new message into the message queue
660     void APMSInsertMsg (
661         TAPrealMsgSystem *      pMS,
662         TAPMsg *                pM
        ) {
663
664         if (pMS->pNewRXMsg) {
665             pMS->pNewRXMsg->pNext = pM;
666         }
667         pMS->pNewRXMsg = pM;
668         if (!pMS->pOldRXMsg) {
669             pMS->pOldRXMsg = pM;
670         }
671         pMS->messagecounter++;
672     }
673
674     // get oldest message
675     TAPMsg * APMSgetMsg (
676         TAPMsgSystem            ms,
677         // the message system
678         TAPMessageID            msgID,

```

```

683         // if 0 all messages are allowed
        TAPNodeID sender,
        // if 0 all senders are
        allowed
684        uint32_t mNumber
        // if 0 all numbers are
        allowed
685    ) {
686
687        TAPrealMsgSystem * pMS = (TAPrealMsgSystem *) ms
        ;
688
689        TAPMsg * res = NULL;
690        // search msg list
691
692        TAPMsg * pM = pMS->pOldRXMsg;
693        TAPMsg * pAntecessorM = NULL;
694        // flags
695        int senderOK;
696        int msgIDok;
697        int numberOK;
698
699        while (pM) {
700            senderOK = 0;
701            msgIDok = 0;
702            numberOK = 0;
703
704            if (!sender) {
705                senderOK = 1;
706            } else {
707                if (*(pM->pH)[
                    eAPMsgHeaderPosition_sender]
                    == sender) senderOK = 1;
708            }
709            if (!msgID) {
710                msgIDok = 1;
711            } else {
712                if (*(pM->pH)[
                    eAPMsgHeaderPosition_msgTypeID
                    ] == msgID) msgIDok = 1;
713            }
714            if (!mNumber) {
715                numberOK = 1;
716            } else {
717                if (*(pM->pH)[
                    eAPMsgHeaderPosition_msgNumber
                    ] == mNumber) numberOK = 1;
718            }
719
720            if ((senderOK) && (msgIDok) && (numberOK
                )) {
721                res = pM;
722                goto exit;
723            }
724            // the id was wrong but the number was

```

```

725         ok >> so the action failed
726         if (numberOK) goto fail;
727         pAntecessorM = pM;
728         pM = pM->pNext;
729     }
730 fail:
731     return NULL;
732 exit:
733     if (pAntecessorM) {
734         pAntecessorM->pNext = pM->pNext;
735     } else {
736         pMS->pOldRXMsg = pM->pNext;
737     }
738     if (pM == pMS->pNewRXMsg) {
739         pMS->pNewRXMsg = NULL;
740     }
741     return res;
742 }
743
744 // wait till a new message has been received
745 void APMSwaitForNewMessage (TAPrealMsgSystem * pMS)
746 {
747     volatile int mc;
748
749     mc = pMS->messagecounter;
750
751     while (mc == pMS->messagecounter) {
752     }
753 }
754
755 // returns 0 if a new message is available
756 inline int APMSisMessageAvalble (TAPrealMsgSystem * pMS
757     ) {
758     return (!pMS->pOldRXMsg) ? 0 : -1;
759 }
760
761 // =====
762 // the receive state machine
763 // =====
764
765 // the receive state machine state function for
766 // receiving the msg header
767 int SMinitial (
768     void *                pVoidSM,
769     // pointer to the
770     statemachine
771     uint32_t *            pD,
772     // pointer to the
773     data
774     int                    number
775     // the number of data
776     elements
777 ) {

```



```

771     TAPReceiveStateMachine *
772     pSM = (TAPReceiveStateMachine *) pVoidSM;
773     TAPrealMsgSystem *
774     pMS = (TAPrealMsgSystem *) pSM->pMS;
775
776     int
777     copyAmount = number;
778
779     int
780     i;
781
782     // 1. try to copy the data to the header
783     if (pSM->elementsLeft < copyAmount) copyAmount =
784     pSM->elementsLeft;
785     // copy
786     for (i = 0; i < copyAmount; i++) {
787         *pSM->pD = *pD;
788         pSM->pD++;
789         pD++;
790     }
791     pSM->elementsLeft -= copyAmount;
792
793     // check if we have to change the statemachine
794     // because we received the header
795     if (pSM->elementsLeft) return 0;
796
797     // yes! alloc msg buffer and (opt.) transfer
798     // data
799
800     // 2. convert endian
801     int msgEndian = pSM->header[
802     eAPMsgHeaderPosition_endian];
803     if (pMS->sysEndianness != msgEndian) {
804         for (i = 1; i <
805             eAPMsgHeaderPosition_headerElementNumber
806             ; i++) {
807             ApendianConversation32Bit(&pSM->
808             header[i]);
809         }
810     }
811
812     // 3. now alloc message
813     // 3.1 get length
814     int msgElementNumber = (int) pSM->header[
815     eAPMsgHeaderPosition_length];
816     // 3.2. get memory
817     pSM->pMsg = APMSgetNewMsg (pMS, msgElementNumber,
818     pSM->pDrv);
819     if (!pSM->pMsg) return -100;
820
821     // 3.3 check getMemory result
822     if (!pSM->pMsg) return -1;
823     // copy message header
824     pSM->pD = (uint32_t *) pSM->pMsg->pH;
825     for (i = 0; i <
826         eAPMsgHeaderPosition_headerElementNumber; i++)
827     {
828         *pSM->pD = pSM->header[i];

```

```

811         pSM->pD++;
812     }
813     pSM->elementsLeft = pSM->header[
814         eAPMsgHeaderPosition_length];
815     // set up the data
816     // 1. check if there is an data element
817     if (!pSM->elementsLeft) {
818         // no! now finish the message
819         return SMmessageFinished(pVoidSM);
820     }
821     // 2. yes
822     // 2.1 setup the sm for the data receiving
823     pSM->state = SMdata;
824     // 2.2 now check if we have to copy some data
825     number -= copyAmount;
826     if (number) {
827         // set the data pointer
828         pD += copyAmount;
829         // and copy the data
830         return SMdata (pVoidSM,pD,number);
831     }
832     return 0;
833 }
834
835 // the receive state machine state function for
836 // receiving the data
837 int SMdata (
838     void *                pVoidSM,
839     // pointer to the
840     statemachine
841     uint32_t *            pD,
842     // pointer to the
843     data
844     int                    number
845     // the number of data
846     elements
847 ) {
848     TAPReceiveStateMachine *
849     pSM = (TAPReceiveStateMachine *) pVoidSM;
850     int
851     copyAmount = number;
852     int
853     i;
854     // 1. transfer the data
855     // do some clipping
856     if (pSM->elementsLeft < copyAmount) copyAmount =
857         pSM->elementsLeft;
858     // copy
859     for (i = 0; i < copyAmount;i++) {
860         *pSM->pD = *pD;
861         pSM->pD++;
862         pD++;
863     }
864     // set statemachine work data

```

```

854     pSM->elementsLeft -= copyAmount;
855     // check if we have to change the statemachine
856     if (pSM->elementsLeft) return 0;
857     int res = SMmessageFinished (pVoidSM);
858     if (res) return res;
859
860     // check if there some bytes left to copy
861     number -= copyAmount;
862     if (number) {
863         // set the data pointer
864         pD += copyAmount;
865         // and copy the data
866         return pSM->state (pVoidSM,pD,number);
867     }
868     return 0;
869 }
870
871 // this function is called when all data have been
872 // received
873 int SMmessageFinished (
874     void *                pVoidSM
875     // pointer to the statemachine
876 ) {
877     TAPReceiveStateMachine *
878     pSM = (TAPReceiveStateMachine *) pVoidSM;
879     TAPMsg *
880     pM;
881
882     // 1. reset SM
883     // set the helper
884     pSM->elementsLeft =
885     eAPMsgHeaderPosition_headerElementNumber;
886     pSM->pD = pSM->header;
887
888     // data
889     pM = pSM->pMsg; // save msg info for inserting
890     pSM->pMsg = NULL;
891
892     // right state function
893     pSM->state = SInitial; // the state
894
895     // 2. insert message at the message system
896     APMSInsertMsg ((TAPrealMsgSystem *)pSM->pMS,pM);
897     return 0;
898 }
899
900 // inits the state machine
901 void APInitReceiveStateMachine (
902     TAPReceiveStateMachine *
903     pSM, // pointer to the
904     state machine
905     TAPMsgSystem
906     pMS, // pointer to
907     the message system

```

```

900         const TAPMsgDrv *
                                pDrv // the driver
                                associated with the statemachine
901     ) {
902         pSM->state = SInitial;
903         pSM->pMS = pMS;
904         pSM->pDrv = pDrv;
905         // set the helper
906         pSM->elementsLeft =
            eAPMsgHeaderPosition_headerElementNumber;
907         pSM->pD = pSM->header;
908
909         // data
910         pSM->pMsg = NULL;
911     }
912
913     int APHandleMsg (
914         TAP *                pAP,
915         TAPMsg *             pM
916     ) {
917
918         TAPMessageID
                                msgID;
919         const THALMsgProcessMessageAssociation *
            pMsgIDandFuncAsso;
920         int
                                i;
921
922         // get message id
923         msgID = (*(pM->pH))[
            eAPMsgHeaderPosition_msgTypeID];
924         // search handler
925         pMsgIDandFuncAsso = gHALMsgProcessRXHandlers.pL
            ;
926         for (i = 0; i < gHALMsgProcessRXHandlers.number;
            i++) {
927             if (((TAPMessageID)pMsgIDandFuncAsso->
                msgID) == msgID) {
928                 return pMsgIDandFuncAsso->
                    pfktHandle(pAP, pM);
929             }
930             pMsgIDandFuncAsso++;
931         }
932         return -1;
933     }
934
935     void APMessageProcessingThread (TAP * pAP) {
936
937         TAPrealMsgSystem *                pMS = (
            TAPrealMsgSystem *) pAP->MS;
938         TAPMsg *                          pM;
939         TAPNodeID                          recv;
940         while (1) {
941             // wait for a message
942             APMSwaitForNewMessage(pMS);

```

```

943         // get the message
944         pM = APMSgetMsg (pMS,0,0,0);
945         // search the message handler
946         recv = (*(pM->pH))[
                eAPMsgHeaderPosition_receiver];
947         if ((recv == dAPNodeID_ALL) || (recv ==
948             pAP->nodeID)) {
                if(APHandleMsg (pAP,pM)) goto
                    exit;
949             }
950         // free memory
951         AP_MMU_free(pMS->mmu, pM->memory);
952     }
953 exit:
954     AP_MMU_free(pMS->mmu, pM->memory);
955 }

```

3.5.2 audio processor blueprint 2 (libsndfile overlapped frame based)

Informations:

description:	a overlapped frame based wavfile processing AP	
includes:		
c-Include	c-Library	system lib
AP.h		no

code:

```

1  // =====
2  // AP uuid = 4
3  // =====
4
5  // the global var for the Frame WAV modul
6  TStjFrameWAVmodule gFrameWAVModule;
7
8
9  // inits the AP
10 int APinit (
11
12         TAP *
13         pAP,
14         TAPNodeID
15         nodeID,
16         const TAPMsgDrv *          pDrvList
17         ,
18         const int
19         driverNumber ,
20         size_t
21         messagePoolSize ,
22         int
23         sysEndian

```

```

17         )
18     {
19         pAP->nodeID = nodeID;
20         pAP->pNodeList = NULL;
21         pAP->pDrvList = pDrvList;
22         pAP->sysEndian = sysEndian;
23         pAP->driverNumber = driverNumber;
24         pAP->msgSysMMU = AP_MMU_create(messagePoolSize);
25         pAP->IP = APInterpreterCreate(pAP);
26         pAP->MS = APMScreate (pAP->msgSysMMU,sysEndian);
27         pAP->msgNumber = 0;
28         pAP->APstate = eAPstate_idle;
29
30         if(
31             (!pAP->msgSysMMU)||
32             (!pAP->IP)||
33             (!pAP->MS)
34         ) return -1;
35         // setup the wav modul
36
37         TStjFrameWAVOpenInfo WI[] = {
38             {1,"input.wav"
39              ,1,44100,1,1024,256},
40             {2,"output.wav"
41              ,0,44100,1,1024,256}
42         };
43
44         if (FrameWAVmoduleInit(2,WI,&gFrameWAVModule))
45             return -2;
46
47         // login the ap to the message system
48         return TX_login(pAP);
49     }
50
51     // deletes the AP
52     void APdelete (TAP * pAP)
53     {
54         APMSdelete (pAP->MS);
55         APInterpreterDelete(pAP->IP);
56         AP_MMU_delete(pAP->msgSysMMU);
57         // closes the wav frame modul
58         FrameWAVmoduleExit(&gFrameWAVModule);
59     }
60
61     // find a node at the list
62     TAPNode * APfindNode(TAP * pAP, TAPNodeID nodeID) {
63         TAPNode * pN = pAP->pNodeList;
64         while (pN) {
65             if (pN->nodeID == nodeID) return pN;
66             pN = pN->pNext;
67         }
68         return NULL;
69     }

```

```

69 // adds a new node to the node list
70 int APaddNode(TAP * pAP, TAPNodeID newNodeID, const
    TAPMsgDrv * pDrv) {
71     if (APfindNode(pAP,newNodeID)) return 1;
72     TAPNode * pN = (TAPNode *) malloc(sizeof(TAPNode
        ));
73     if (!pN) return -1;
74     pN->nodeID = newNodeID;
75     pN->pDrv = pDrv;
76     pN->pNext = pAP->pNodeList;
77     pAP->pNodeList = pN;
78     return 0;
79 }
80
81 // removes a node from the node list
82 void APremoveNode(TAP * pAP, TAPNodeID nodeID){
83     TAPNode * pAntN = pAP->pNodeList; // antecessor
        node
84     TAPNode * pActN = pAP->pNodeList; // actual node
85
86     while (pActN) {
87         // compare node id's
88         if (pActN->nodeID == nodeID) {
89             // unchain
90
91             // check if we at the first
                position at the list
92             if (pAP->pNodeList == pAntN) {
93                 // reset the pointer
94                 pAP->pNodeList = pActN->
                    pNext;
95             } else {
96                 // set the antecessor
97                 pAntN->pNext = pActN->
                    pNext;
98             }
99             // free node
100             free(pActN);
101             // and abort
102             return;
103         }
104         // the actual element becomes the
            precessor element
105         pAntN = pActN;
106         pActN = pActN->pNext;
107     }
108 }
109
110 // get a new message number
111 unsigned int APgetNewMessageNumber (TAP *pAP) {
112     pAP->msgNumber++;
113     return pAP->msgNumber;
114 }
115
116 // find the driver associated with der nodeID

```

```

117 const TAPMsgDrv * APfindDrvBySenderID (TAP * pAP,
    TAPNodeID node) {
118     TAPNode * pN = pAP->pNodeList;
119     while (pN) {
120         if (pN->nodeID == node) {
121             return pN->pDrv;
122         }
123         pN = pN->pNext;
124     }
125     return NULL;
126 }
127
128 // runs the AP
129 int APrun(TAP *pAP) {
130     pAP->APstate = eAPstate_run;
131 }
132
133
134 typedef struct SAPrealMMUMemory {
135     int *
        pData; // the data
136     size_t
        count; // amount of data
        elements
137     struct SAPrealMMUMemory * pNext;
        // next element
138     struct SAPrealMMUMemory * pPrev;
        // previous element
139 } TAPrealMMUMemory;
140
141 //the mmu type
142 typedef struct SAPrealMMU {
143     int *
        // the memory bolck
        TAPrealMMUMemory * pStart;
        // first element
144     TAPrealMMUMemory * pEnd;
        // second element
145     TAPrealMMUMemory * pUnusedList;
        // list with the unused elements
146     int *
        pUnusedData; // pointer to the
        unused memory
147     size_t
        elementsAvailable; // amount of elements
        witch are available without using the
        garbage collector
148     size_t
        totalAvailable; // total amount of
        free bytes
149
150 } TAPrealMMU;
151
152 // =====
153 // memory entry functions

```



```

155 // =====
156
157 // a little macro for unchaining an element
158 #define DMemoryEntryUnchain(pM) \
159     if (pM->pNext) pM->pNext->pPrev = pM->pPrev; \
160     if (pM->pPrev) pM->pPrev->pNext = pM->pNext
161
162
163 //creates an memory entry
164 TAPrealMMUMemory * MemoryEntry_create () {
165     TAPrealMMUMemory * pM = NULL;
166
167     pM = (TAPrealMMUMemory *) malloc(sizeof(
168         TAPrealMMUMemory));
169     if (!pM) return NULL;
170     pM->pData = NULL;
171     pM->count = 0;
172     pM->pNext = NULL;
173     pM->pPrev = NULL;
174
175     return pM;
176 }
177
178 //deletes an memory Entry
179 void MemoryEntry_delete (
180     TAPrealMMUMemory * pM // the memory to
181     delete
182 )
183 {
184     // put the entry out of the chain
185     DMemoryEntryUnchain(pM);
186     // now we delete it
187     free(pM);
188 }
189
190 // =====
191 // mmu helper
192 // =====
193
194 //alloc if needed a new memory entry
195 TAPrealMMUMemory * MMU_helper_createMemoryEntry (
196     TAPrealMMU * pMMU // MMU
197     structure to init
198 )
199 {
200     // check if we have to alloc a new memory entry
201     if (!pMMU->pUnusedList) return
202     MemoryEntry_create();
203     // no there is some left at the list
204     TAPrealMMUMemory * pM;
205     // take the first one
206     pM = pMMU->pUnusedList;
207     // reset the list
208     pMMU->pUnusedList = pM->pNext;
209     // now unchain the element (for sure)

```

```

206     DMemoryEntryUnchain(pM);
207     // set the element pointers
208     pM->pNext = NULL;
209     pM->pPrev = NULL;
210     return pM;
211 }
212
213 //the garbage collector
214 void MMU_helper_garbageCollector (
215     TAPrealMMU * pMMU           // MMU
216     structure to init
217 )
218 {
219     TAPrealMMUMemory * pM = pMMU->pStart;
220     int * pD = pMMU->memory;
221     while (pM) {
222         // check if we have to move the data
223         if (pD != pM->pData) {
224             // move the data
225             memmove(pD, pM->pData, pM->count *
226                 sizeof(int));
227         }
228         // reset the destination pointer
229         pD += pM->count;
230         pM = pM->pNext;
231     }
232     // compressing memory finished
233     // set the mmu vars new
234     pMMU->elementsAvailable = pMMU->totalAvailable;
235     pMMU->pUnusedData = pD;
236 }
237
238 // create a mmu
239 TAPMMU AP_MMU_create (size_t elementsNumber) {
240     TAPrealMMU * pMMU;
241
242     pMMU = (TAPrealMMU *) malloc (sizeof(TAPrealMMU)
243         );
244     if (!pMMU) return NULL;
245
246     // setup lists
247     pMMU->pStart = NULL;
248     pMMU->pEnd = NULL;
249     pMMU->pUnusedList = NULL;
250
251     pMMU->elementsAvailable = elementsNumber;
252     pMMU->pUnusedData = (int *) malloc (pMMU->
253         elementsAvailable);
254     pMMU->totalAvailable = pMMU->elementsAvailable;
255     return pMMU;
256 }

```

```

257
258 // destroying the mmu
259 void AP_MMU_delete (TAPMMU mmu) {
260     TAPrealMMU * pMMU = (TAPrealMMU *) mmu;
261
262     TAPrealMMUMemory * pM;
263     TAPrealMMUMemory * pMnext;
264
265     // 1. delete al mmu entry's
266     // 1.1 unused entry
267     pM = pMMU->pUnusedList;
268     while (pM) {
269         pMnext = pM->pNext;
270         MemoryEntry_delete(pM);
271         pM = pMnext;
272     }
273     pMMU->pUnusedList = NULL;
274     // 1.2 used blocks
275     pM = pMMU->pStart;
276     while (pM) {
277         pMnext = pM->pNext;
278         MemoryEntry_delete(pM);
279         pM = pMnext;
280     }
281     pMMU->pStart = NULL;
282     pMMU->pEnd = NULL;
283     // 2. delete mmu memory
284     free (pMMU->memory);
285
286 }
287
288 // getting memmory from the mmu
289 TAPMMUmemory AP_MMU_get (TAPMMU mmu, size_t elements) {
290     TAPrealMMU * pMMU = (TAPrealMMU *) mmu;
291
292     // check if there is enough space
293     if (pMMU->totalAvailable < elements) goto error;
294     // check if we have to use the garbage collector
295     if (pMMU->elementsAvailable < elements) {
296         // start garbage collector
297         MMU_helper_garbageCollector(pMMU);
298     }
299     // we have enough memory so let's allocate some
300
301     // get a new entry
302     TAPrealMMUMemory * pM;
303     pM = MMU_helper_createMemoryEntry(pMMU);
304     if (!pM) return NULL;
305     // get some memory
306     pM->pData = pMMU->pUnusedData;
307     pM->count = elements;
308     // refresh data
309     pMMU->pUnusedData += elements;
310     pMMU->totalAvailable -= elements;
311     pMMU->elementsAvailable -= elements;

```

```

312         // insert memory element at the end of the list
           and update last element
313     pM->pPrev = pMMU->pEnd;
314     if (pMMU->pEnd) pMMU->pEnd->pNext = pM;
315     if (!pMMU->pStart) pMMU->pStart = pM;
316     pMMU->pEnd = pM;
317
318     return pM;
319 error:
320
321     return NULL;
322 }
323
324 // free memmory from the mmu
325 void AP_MMU_free (TAPMMU mmu, TAPMMUememory memory) {
326     TAPrealMMU * pMMU = (TAPrealMMU *) mmu;
327     TAPrealMMUMemory * pM = (TAPrealMMUMemory *)
           memory;
328
329     if (!pM) return;
330     // set mmu settings
331     if (pMMU->pStart == pM) pMMU->pStart = pM->pNext
           ;
332     if (pMMU->pEnd == pM) pMMU->pEnd = pM->pPrev;
333     // unchain element
334     DMemoryEntryUnchain (pM);
335     // and put it to the chain of unused
336     pM->pNext = pMMU->pUnusedList;
337     pM->pPrev = NULL;
338     if (pMMU->pUnusedList) {
339         pMMU->pUnusedList->pPrev = pM;
340     }
341     pMMU->pUnusedList = pM;
342     // now set the mmu data new
343     pMMU->totalAvailable += pM->count;
344 }
345
346 // getting access to the MMU data
347 void * AP_MMU_getData (TAPMMUememory memory) {
348     TAPrealMMUMemory * pM = (TAPrealMMUMemory *)
           memory;
349     return pM->pData;
350 }
351
352 // the real interpreter
353 typedef struct SAPrealInterpreter {
354     int
           state;
           // state of the IP
355
           int
           nextState;
           // the next state of the IP
356     TAPInterpreterCPU
           cpu;
           // the IP

```

```

358         core
TAPInterpreterFuncCall *          code;
                                   // the code
359     int32_t
        instructionCount;          // number of
        instructions at the code
360     TAPInterpreterVariable *      variables;
                                   // the variables
361     int32_t
        variableCount;             // number of
        the variables
362     int
        sysEndian;
        // endian of the system
363 } TAPrealInterpreter;
364
365 // create a new interpreter
366 TAPInterpreter APIInterpreterCreate (void * pAP) {
367     TAPrealInterpreter * pIP = NULL;
368     pIP = (TAPrealInterpreter *) malloc (sizeof(
        TAPrealInterpreter));
369     if (!pIP) return NULL;
370
371     pIP->state = eAPIInterpreterState_idle;
372     pIP->nextState = eAPIInterpreterState_idle;
373     pIP->sysEndian = ((TAP *)pAP)->sysEndian;
374     pIP->cpu.CF = 0;
375     pIP->cpu.EF = 0;
376     pIP->cpu.pCodeStart = NULL;
377     pIP->cpu.pCodeEnd = NULL;
378     pIP->cpu.pIP = NULL;
379
380     pIP->code = NULL;
381     pIP->instructionCount = 0;
382
383     pIP->variables = NULL;
384     pIP->variableCount = 0;
385
386     return pIP;
387 }
388
389 // cleans the interpreter
390 void APIInterpreterClean (TAPInterpreter IP) {
391     TAPrealInterpreter * pIP = (TAPrealInterpreter
        *) IP;
392
393     // clean code
394     if (pIP->code) {
395         free (pIP->code);
396         pIP->code = NULL;
397     }
398     pIP->instructionCount = 0;
399
400     // clean variables
401     TAPInterpreterVariable * pV = pIP->variables;

```

```

402     int i;
403     for (i = 0; i < pIP->variableCount; i++) {
404         if (pV->pVI) pV->pVI->pFkt_delete(pV->
405             pData);
406         pV++;
407     }
408     if (pIP->variables) {
409         free (pIP->variables);
410         pIP->variables = NULL;
411     }
412     pIP->variableCount = 0;
413 }
414
415 // deletes the interpreter
416 void APIInterpreterDelete (TAPIInterpreter IP) {
417     TAPrealInterpreter * pIP = (TAPrealInterpreter
418         *) IP;
419     APIInterpreterClean(IP);
420     free (pIP);
421 }
422
423 int APIInterpreterStateRun(TAPIInterpreter IP) {
424     TAPrealInterpreter * pIP = (TAPrealInterpreter
425         *) IP;
426     TAPIInterpreterFuncCall * pFC;
427
428     // setup cpu
429     pIP->cpu.CF = 0;
430     pIP->cpu.EF = 0;
431     pIP->cpu.pIP = pIP->code;
432     pIP->cpu.pCodeStart = pIP->code;
433     pIP->cpu.pCodeEnd = pIP->code + pIP->
434         instructionCount;
435
436     // run code
437     while (eAPIInterpreterState_run == pIP->state) {
438         pFC = pIP->cpu.pIP;
439         // check if we reached the end of the
440         // code
441         if (pFC > pIP->cpu.pCodeEnd) {
442             return 0;
443         }
444         // execute command
445         pFC->pHALFkt (&(pIP->cpu), pFC->param);
446         // check error flags
447         if (pIP->cpu.EF) {
448             return -1;
449         }
450     }
451     return 1;
452 }
453
454 // process the actual state

```

```

452 int APInterpreterProcessState(TAPInterpreter IP){
453     TAPrealInterpreter * pIP = (TAPrealInterpreter
        *) IP;
454     pIP->state = pIP->nextState;
455     int rc = 0;
456
457     switch (pIP->state) {
458         case eAPInterpreterState_idle:
459             break;
460         case eAPInterpreterState_loadProgramm:
461             break;
462         case eAPInterpreterState_run:
463             rc = APInterpreterStateRun(IP);
464             if (rc >= 0) pIP->state =
                eAPInterpreterState_idle;
465             break;
466         case eAPInterpreterState_oneStep:
467             break;
468         case eAPInterpreterState_halt:
469             break;
470         default:
471             return -10;
472     }
473     return rc;
474 }
475
476 // set interpreter state
477 int APInterpreterSetState (TAPInterpreter IP, int
478     msgEndian, int32_t state) {
479     TAPrealInterpreter * pIP = (TAPrealInterpreter
        *) IP;
480     if (msgEndian != pIP->sysEndian) {
481         APendianConversation32Bit((uint32_t *)&
            state);
482     }
483     pIP->nextState = (int) state;
484     return 0;
485 }
486
487 // setup the interpreter for a new program
488 int APInterpreterInitNewProgramm (TAPInterpreter IP, int
489     msgEndian, int32_t instructionsNumber, int32_t
490     VariableNumber) {
491     TAPrealInterpreter * pIP = (TAPrealInterpreter
        *) IP;
492     int i;
493
494     APInterpreterClean (IP);
495
496     if (msgEndian != pIP->sysEndian) {
497         APendianConversation32Bit((uint32_t *)&
            instructionsNumber);
498         APendianConversation32Bit((uint32_t *)&
            VariableNumber);

```

```

497     }
498
499     pIP->code = (TAPInterpreterFuncCall *) malloc(
        sizeof(TAPInterpreterFuncCall)*
        instructionsNumber);
500     pIP->instructionCount = instructionsNumber;
501
502     pIP->variables = (TAPInterpreterVariable *)
        malloc(sizeof(TAPInterpreterVariable) * (
        VariableNumber));
503     for (i = 0; i < VariableNumber;i++) {
504         pIP->variables[i].pData = NULL;
505         pIP->variables[i].pVI = NULL;
506     }
507     pIP->variableCount = VariableNumber;
508
509     return 0;
510 }
511
512 // load a variable/~array to an index
513 int APInterpreterLoadVar (TAPInterpreter IP, int
    msgEndian, int32_t index, int32_t varTypeID, int32_t
    numberOfElements)
514 {
515     TAPrealInterpreter * pIP = (TAPrealInterpreter
        *) IP;
516     if (msgEndian != pIP->sysEndian) {
517         APendianConversation32Bit((uint32_t *)&
            index);
518         APendianConversation32Bit((uint32_t *)&
            varTypeID);
519     }
520
521     if ((index < 0) || (index > pIP->variableCount))
        return -1;
522
523     // set pointer to the runtime variable
524     TAPInterpreterVariable * pRTV = pIP->variables +
        index;
525     const THAL_Variable * pV = HALfindVar(varTypeID)
        ;
526     if (!pV) return -2;
527
528     pRTV->pData = pV->pFkt_create((unsigned int)
        numberOfElements);
529     //if (!pRTV->pData) return -3;
530
531     pRTV->pVI = pV;
532     return 0;
533 }
534
535 // load a single Instruction to an index
536 int APInterpreterLoadInstr (TAPInterpreter IP,int
    msgEndian, int32_t index, int32_t * pRawInstr)
537 {

```



```

538     TAPrealInterpreter * pIP = (TAPrealInterpreter
539         *) IP;
540     if (msgEndian != pIP->sysEndian) {
541         APendianConversation32Bit((uint32_t *)&
542             index);
543     }
544     if ((index < 0) || (index > pIP->
545         instructionCount)) return -1;
546     TAPInterpreterFuncCall * pIFC = pIP->code +
547         index;
548     memset (pIFC, 0, sizeof(TAPInterpreterFuncCall))
549     ;
550
551     // get function
552     int32_t fid = *pRawInstr;
553     if (msgEndian != pIP->sysEndian) {
554         APendianConversation32Bit((uint32_t *)&
555             fid);
556     }
557     const THALFunction * pF = HALfindFunction(fid);
558     if (!pF) return -2;
559     pIFC->pHALFkt = pF->ppktHAL;
560
561     // convert parameters
562     pRawInstr++; // set to the first parameter
563     int i;
564     const THALFunctionParam * pP = pF->paramList.pL;
565     TuAPIInterpreterFunctionParameter * pIFP = pIFC->
566         param;
567     for (i = 0; i < pF->paramList.number; i++) {
568         if (APconvertRawParamData (msgEndian,pIP
569             ->sysEndian,pRawInstr,pP,pIFP,pIP->
570             variables)) return -3;
571         pP++;
572         pRawInstr++;
573         pIFP++;
574     }
575     return 0;
576 }
577
578 typedef struct SAPrealMsgSystem {
579     TAPMsg *
580         // pointer to the oldest received
581         messages
582     TAPMsg *
583         // pointer to the newest received
584         messages
585     TAPMMU
586         mmu;
587     // the mmu
588     int
589         sysEndianness; // the system endianness
590     int
591         messagecounter; // a counter for checkin if a
592         new message has been received

```

```

576 } TAPrealMsgSystem;
577
578
579
580 int SMinitial (
581     void *                pVoidSM,
582     // pointer to the
583     statemachine
584     uint32_t *            pD,
585     // pointer to the
586     data
587     int                    number
588     // the number of data
589     elements
590 );
591
592 int SMdata (
593     void *                pVoidSM,
594     // pointer to the
595     statemachine
596     uint32_t *            pD,
597     // pointer to the
598     data
599     int                    number
600     // the number of data
601     elements
602 );
603
604 int SMmessageFinished (
605     void *                pVoidSM
606     // pointer to the statemachine
607 );
608
609
610
611 // create AP message system
612 TAPMsgSystem APMScreate (
613     TAPMMU                mmu,
614     // the mmu
615     int                    sysEndianness // the system
616     endianness
617 ) {
618     TAPrealMsgSystem * pMS = (TAPrealMsgSystem *)
619         malloc (sizeof(TAPrealMsgSystem));
620     if (!pMS) return NULL;
621     pMS->mmu = mmu;
622     pMS->sysEndianness = sysEndianness;
623     pMS->pOldRXMsg = NULL;
624     pMS->pNewRXMsg = NULL;
625     pMS->messagecounter = 0;
626
627     return pMS;
628 }
629
630

```

```

614
615 void APMSdelete (
616     TAPMsgSystem ms
617 ) {
618     TAPrealMsgSystem * pMS = (TAPrealMsgSystem *) ms
        ;
619     free (pMS);
620 }
621
622
623 // frees a message from the message system
624 void APMSdeleteMsg (
625     TAPMsgSystem      ms,
626     TAPMsg *          pM
627 ) {
628     TAPrealMsgSystem * pMS = (TAPrealMsgSystem *) ms
        ;
629     AP_MMU_free(pMS->mmu, pM->memory);
630 }
631
632 // get memory for a new message
633 TAPMsg * APMSgetNewMsg (
634     TAPrealMsgSystem *      pMsgSys,
635     int
        dataElementsNumber,
636     const TAPMsgDrv *      pDrv
637 ) {
638     TAPMMUmemory m = AP_MMU_get (pMsgSys->mmu,
        sizeof(TAPMsg)/sizeof(int) +
        dataElementsNumber*sizeof(uint32_t)/sizeof(
        int));
639     if (!m) return NULL;
640
641     int * pRD = (int *) AP_MMU_getData(m);
642     TAPMsg * pM = (TAPMsg *) pRD;
643     pM->memory = m;
644     pM->extraData.pDrv = pDrv;
645     pM->pH = (TAPMsgHeader *)((int *) pRD + sizeof(
        TAPMsg)/sizeof(int));
646     pM->pData = (int *)pM->pH + sizeof(TAPMsgHeader)
        /sizeof(int);
647     pM->pNext = NULL;
648     return pM;
649 }
650
651 // insert a new message into the message queue
652 void APMSinsertMsg (
653     TAPrealMsgSystem *      pMS,
654     TAPMsg *                pM
655 ) {
656
657     if (pMS->pNewRXMsg) {
658         pMS->pNewRXMsg->pNext = pM;
659     }
660     pMS->pNewRXMsg = pM;

```

```

661         if (!pMS->pOldRXMsg) {
662             pMS->pOldRXMsg = pM;
663         }
664         pMS->messagecounter++;
665
666
667     }
668 }
669
670 // get oldest message
671 TAPMsg * APMSgetMsg (
672     TAPMsgSystem          ms,
673     TAPMessageID          msgID,
674     TAPNodeID             sender,
675     uint32_t              mNumber
676     // if 0 all messages are allowed
677     // if 0 all senders are allowed
678     // if 0 all numbers are allowed
679 ) {
680     TAPrealMsgSystem * pMS = (TAPrealMsgSystem *) ms;
681
682     TAPMsg * res = NULL;
683     // search msg list
684
685     TAPMsg * pM = pMS->pOldRXMsg;
686     TAPMsg * pAntecessorM = NULL;
687     // flags
688     int senderOK;
689     int msgIDok;
690     int numberOK;
691
692     while (pM) {
693         senderOK = 0;
694         msgIDok = 0;
695         numberOK = 0;
696
697         if (!sender) {
698             senderOK = 1;
699         } else {
700             if (*(pM->pH)[
701                 eAPMsgHeaderPosition_sender]
702                 == sender) senderOK = 1;
703         }
704
705         if (!msgID) {
706             msgIDok = 1;
707         } else {
708             if (*(pM->pH)[
709                 eAPMsgHeaderPosition_msgTypeID
710                 ] == msgID) msgIDok = 1;
711         }
712
713         if (!number) {
714             numberOK = 1;
715         } else {
716             if (pM->mNumber == number) numberOK = 1;
717         }
718
719         if (senderOK & msgIDok & numberOK) {
720             res = pM;
721             pM = pAntecessorM;
722         } else {
723             pM = pM->pNext;
724         }
725     }
726 }

```

```

705         if (!mNumber) {
706             numberOK = 1;
707         } else {
708             if (*(pM->pH)[
                    eAPMsgHeaderPosition_msgNumber
                ] == mNumber) numberOK = 1;
709         }
710
711         if ((senderOK) && (msgIDok) && (numberOK
            )) {
712             res = pM;
713             goto exit;
714         }
715         // the id was wrong but the number was
            ok >> so the action failed
716         if (numberOK) goto fail;
717         pAntecessorM = pM;
718         pM = pM->pNext;
719     }
720 fail:
721     return NULL;
722 exit:
723     if (pAntecessorM) {
724         pAntecessorM->pNext = pM->pNext;
725     } else {
726         pMS->pOldRXMsg = pM->pNext;
727     }
728     if (pM == pMS->pNewRXMsg) {
729         pMS->pNewRXMsg = NULL;
730     }
731
732     return res;
733 }
734
735 // wait till a new message has been received
736 void APMSwaitForNewMessage (TAPrealMsgSystem * pMS)
737 {
738     volatile int mc;
739
740     mc = pMS->messagecounter;
741
742     while (mc == pMS->messagecounter) {
743
744     }
745 }
746
747 // returns 0 if a new message is available
748 inline int APMSisMessageAvailble (TAPrealMsgSystem * pMS
    ) {
749     return (!pMS->pOldRXMsg) ? 0 : -1;
750 }
751
752 // =====
753 // the receive state machine
754 // =====

```

```

755 // the receive state machine state function for
756 // receiving the msg header
757 int SInitial (
758     void *                pVoidSM,
759     // pointer to the
760     statemachine
761     uint32_t *            pD,
762     // pointer to the
763     data
764     int                    number
765     // the number of data
766     elements
767 ) {
768     TAPReceiveStateMachine *
769     pSM = (TAPReceiveStateMachine *) pVoidSM;
770     TAPrealMsgSystem *
771     pMS = (TAPrealMsgSystem *) pSM->pMS;
772     int
773     copyAmount = number;
774     int
775     i;
776     // 1. try to copy the data to the header
777     if (pSM->elementsLeft < copyAmount) copyAmount =
778     pSM->elementsLeft;
779     // copy
780     for (i = 0; i < copyAmount; i++) {
781         *pSM->pD = *pD;
782         pSM->pD++;
783         pD++;
784     }
785     pSM->elementsLeft -= copyAmount;
786     // check if we have to change the statemachine
787     // because we received the header
788     if (pSM->elementsLeft) return 0;
789     // yes! alloc msg buffer and (opt.) transfer
790     data
791     // 2. convert endian
792     int msgEndian = pSM->header[
793     eAPMsgHeaderPosition_endian];
794     if (pMS->sysEndianness != msgEndian) {
795         for (i = 1; i <
796             eAPMsgHeaderPosition_headerElementNumber
797             ; i++) {
798             APendianConversation32Bit(&pSM->
799             header[i]);
800         }
801     }
802     // 3. now alloc message
803     // 3.1 get length
804     int msgElementNumber = (int) pSM->header[

```

```

792         eAPMsgHeaderPosition_length];
793     // 3.2. get memory
794     pSM->pMsg = APMSgetNewMsg (pMS,msgElementNumber,
795                               pSM->pDrv);
796     if (!pSM->pMsg) return -100;
797
798     // 3.3 check getMemory result
799     if (!pSM->pMsg) return -1;
800     // copy message header
801     pSM->pD = (uint32_t *) pSM->pMsg->pH;
802     for (i = 0; i <
803          eAPMsgHeaderPosition_headerElementNumber;i++)
804     {
805         *pSM->pD = pSM->header[i];
806         pSM->pD++;
807     }
808     pSM->elementsLeft = pSM->header[
809         eAPMsgHeaderPosition_length];
810     // set up the data
811     // 1. check if there is an data element
812     if (!pSM->elementsLeft) {
813         // no! now finish the message
814         return SMmessageFinished(pVoidSM);
815     }
816     // 2. yes
817     // 2.1 setup the sm for the data receiving
818     pSM->state = SMdata;
819     // 2.2 now check if we have to copy some data
820     number -= copyAmount;
821     if (number) {
822         // set the data pointer
823         pD += copyAmount;
824         // and copy the data
825         return SMdata (pVoidSM,pD,number);
826     }
827     return 0;
828 }
829
830 // the receive state machine state function for
831 // receiving the data
832 int SMdata (
833     void *                pVoidSM,
834     // pointer to the
835     statemachine
836     uint32_t *            pD,
837     // pointer to the
838     data
839     int                    number
840     // the number of data
841     elements
842 ) {
843     TAPReceiveStateMachine *
844     pSM = (TAPReceiveStateMachine *) pVoidSM;
845     int

```

```

834         copyAmount = number;
835         int
836             i;
837         // 1. transfer the data
838         // do some clipping
839         if (pSM->elementsLeft < copyAmount) copyAmount =
840             pSM->elementsLeft;
841         // copy
842         for (i = 0; i < copyAmount; i++) {
843             *pSM->pD = *pD;
844             pSM->pD++;
845             pD++;
846         }
847         // set statemachine work data
848         pSM->elementsLeft -= copyAmount;
849         // check if we have to change the statemachine
850         if (pSM->elementsLeft) return 0;
851         int res = SMmessageFinished (pVoidSM);
852         if (res) return res;
853
854         // check if there some bytes left to copy
855         number -= copyAmount;
856         if (number) {
857             // set the data pointer
858             pD += copyAmount;
859             // and copy the data
860             return pSM->state (pVoidSM, pD, number);
861         }
862         return 0;
863     }
864 }
865
866 // this function is called when all data have been
867 // received
868 int SMmessageFinished (
869     void *                pVoidSM
870     // pointer to the statemachine
871 ) {
872     TAPReceiveStateMachine *
873     pSM = (TAPReceiveStateMachine *) pVoidSM;
874     TAPMsg *
875     pM;
876
877     // 1. reset SM
878     // set the helper
879     pSM->elementsLeft =
880         eAPMsgHeaderPosition_headerElementNumber;
881     pSM->pD = pSM->header;
882
883     // data
884     pM = pSM->pMsg; // save msg info for inserting
885     pSM->pMsg = NULL;
886
887     // right state function
888     pSM->state = SInitial; // the state

```



```

881 // 2. insert message at the message system
882 APMSInsertMsg ((TAPrealMsgSystem *)pSM->pMS,pM);
883 return 0;
884 }
885
886 // inits the state machine
887 void APInitReceiveStateMachine (
888     TAPReceiveStateMachine *
889         pSM, // pointer to the
            state machine
890     TAPMsgSystem
            pMS, // pointer to
            the message system
891     const TAPMsgDrv *
            pDrv // the driver
            associated with the statemachine
892 ) {
893     pSM->state = SInitial;
894     pSM->pMS = pMS;
895     pSM->pDrv = pDrv;
896     // set the helper
897     pSM->elementsLeft =
            eAPMsgHeaderPosition_headerElementNumber;
898     pSM->pD = pSM->header;
899
900     // data
901     pSM->pMsg = NULL;
902 }
903
904 int APHandleMsg (
905     TAP *                pAP,
906     TAPMsg *             pM
907 ) {
908
909     TAPMessageID
            msgID;
910     const THALMsgProcessMessageAssociation *
            pMsgIDandFunctAsso;
911     int
            i;
912
913     // get message id
914     msgID = (*(pM->pH))[
            eAPMsgHeaderPosition_msgTypeID];
915     // search handler
916     pMsgIDandFunctAsso = gHALMsgProcessRXHandlers.pL
            ;
917     for (i = 0; i < gHALMsgProcessRXHandlers.number;
            i++) {
918         if (((TAPMessageID)pMsgIDandFunctAsso->
            msgID) == msgID) {
919             return pMsgIDandFunctAsso->
            pfktHandle(pAP,pM);
920         }
921     }

```

```

921         pMsgIDandFunctAsso++;
922     }
923     return -1;
924 }
925
926 void APMessageProcessingThread (TAP * pAP) {
927     TAPrealMsgSystem *      pMS = (
928         TAPrealMsgSystem *) pAP->MS;
929     TAPMsg *                pM;
930     TAPNodeID               recv;
931     while (1) {
932         // wait for a message
933         APMSwaitForNewMessage(pMS);
934         // get the message
935         pM = APMSgetMsg (pMS,0,0,0);
936         // search the message handler
937         recv = (*(pM->pH))[
938             eAPMsgHeaderPosition_receiver];
939         if ((recv == dAPNodeID_ALL) || (recv ==
940             pAP->nodeID)) {
941             if (APHandleMsg (pAP,pM)) goto
942                 exit;
943         }
944         // free memory
945         AP_MMU_free(pMS->mmu, pM->memory);
946     }
947 exit:
948     AP_MMU_free(pMS->mmu, pM->memory);
949 }

```

3.5.3 audio processor blueprint 3 (ADSP 21369 block-based, ADC in, DAC3 & DAC4 out, UART @9600,n,8,1)

Informations:

description:	a ADSP AP
--------------	-----------

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====
2 // AP uuid = 7
3 // =====
4 // inits the AP
5 int APinit (
6             TAP *
              pAP,

```

```

7         TAPNodeID
            nodeID,
8         const TAPMsgDrv *      pDrvList
9         ,
10        const int
            driverNumber,
11        size_t
            messagePoolSize,
            int
                sysEndian
12    )
13 {
14     gAPendianFlag = sysEndian;
15
16     pAP->nodeID = nodeID;
17     pAP->pNodeList = NULL;
18     pAP->pDrvList = pDrvList;
19     pAP->driverNumber = driverNumber;
20     pAP->msgSysMMU = AP_MMU_create(messagePoolSize);
21     pAP->IP = APInterpreterCreate(pAP);
22     pAP->MS = APMScreate (pAP->msgSysMMU,sysEndian);
23     pAP->msgNumber = 0;
24     pAP->APstate = eAPstate_idle;
25
26     if(
27         (!pAP->msgSysMMU)||
28         (!pAP->IP)||
29         (!pAP->MS)
30     ) return -1;
31
32     initHW(drv_2_cbAPClient);
33
34     startHW();
35
36     // init drv
37     TAPMsgDrv * pDrv = (TAPMsgDrv *)pDrvList;
38     int i;
39     for (i = 0; i < driverNumber; i++) {
40         pDrv->pfkt_open(pAP, pDrv);
41         pDrv++;
42     }
43
44     // login the ap to the message system
45     return TX_login(pAP);
46 }
47
48 // deletes the AP
49 void APdelete (TAP * pAP)
50 {
51     TX_logout(pAP);
52     // close & destroy drv
53     TAPMsgDrv * pDrv = (TAPMsgDrv *)pAP->pDrvList;
54     int i;
55     for (i = 0; i < pAP->driverNumber; i++) {
56         pDrv->pfkt_close(pDrv);

```

```

57         pDrv->pfkt_destroy(pDrv);
58         pDrv++;
59     }
60
61     APMSdelete (pAP->MS);
62     APInterpreterDelete(pAP->IP);
63     AP_MMU_delete(pAP->msgSysMMU);
64 }
65
66 // find a node at the list
67 TAPNode * APfindNode(TAP * pAP, TAPNodeID nodeID) {
68     TAPNode * pN = pAP->pNodeList;
69     while (pN) {
70         if (pN->nodeID == nodeID) return pN;
71         pN = pN->pNext;
72     };
73     return NULL;
74 }
75
76 // adds a new node to the node list
77 int APaddNode(TAP * pAP, TAPNodeID newNodeID, const
78     TAPMsgDrv * pDrv) {
79     if (APfindNode(pAP,newNodeID)) return 1;
80     TAPNode * pN = (TAPNode *) malloc(sizeof(TAPNode
81         ));
82     if (!pN) return -1;
83     pN->nodeID = newNodeID;
84     pN->pDrv = pDrv;
85     pN->pNext = pAP->pNodeList;
86     pAP->pNodeList = pN;
87     return 0;
88 }
89
90 // removes a node from the node list
91 void APremoveNode(TAP * pAP, TAPNodeID nodeID){
92     TAPNode * pAntN = pAP->pNodeList; // antecessor
93     node
94     TAPNode * pActN = pAP->pNodeList; // actual node
95
96     while (pActN) {
97         // compare node id's
98         if (pActN->nodeID == nodeID) {
99             // unchain
100
101             // check if we at the first
102             position at the list
103             if (pAP->pNodeList == pAntN) {
104                 // reset the pointer
105                 pAP->pNodeList = pActN->
106                     pNext;
107             } else {
108                 // set the antecessor
109                 pAntN->pNext = pActN->
110                     pNext;
111             }
112         }
113     }
114 }

```

```

1106         // free node
1107         free(pActN);
1108         // and abort
1109         return;
1110     }
1111     // the actual element becomes the
1112     // predecessor element
1113     pAntN = pActN;
1114     pActN = pActN->pNext;
1115 }
1116
1117 // get a new message number
1118 unsigned int APgetNewMessageNumber (TAP *pAP) {
1119     pAP->msgNumber++;
1120     return pAP->msgNumber;
1121 }
1122
1123 // find the driver associated with der nodeID
1124 const TAPMsgDrv * APfindDrvBySenderID (TAP * pAP,
1125     TAPNodeID node) {
1126     TAPNode * pN = pAP->pNodeList;
1127     while (pN) {
1128         if (pN->nodeID == node) {
1129             return pN->pDrv;
1130         }
1131         pN = pN->pNext;
1132     }
1133     return NULL;
1134 }
1135
1136 // runs the AP
1137 int APrun(TAP *pAP) {
1138     pAP->APstate = eAPstate_run;
1139     return 0;
1140 }
1141
1142 typedef struct SAPrealMMUMemory {
1143     int *
1144         pData;                // the data
1145     size_t
1146         count;                // amount of data
1147         elements
1148     struct SAPrealMMUMemory *    pNext;
1149         // next element
1150     struct SAPrealMMUMemory *    pPrev;
1151         // previous element
1152 } TAPrealMMUMemory;
1153
1154 //the mmu type
1155 typedef struct SAPrealMMU {
1156     int *
1157         memory;
1158     TAPrealMMUMemory *
1159         // the memory bolck
1160         pStart;

```

```

153         // first element
TAPrealMMUMemory * pEnd;
154         // second element
TAPrealMMUMemory * pUnusedList;
155         // list with the unused elements
int *
    pUnusedData; // pointer to the
    unused memory
156 size_t
    elementsAvailable; // amount of elements
    witch are available without using the
    garbage collector
157 size_t
    totalAvailable; // total amount of
    free bytes
158
159 } TAPrealMMU;
160
161 // =====
162 // memory entry functions
163 // =====
164
165 // a little macro for unchaining an element
166 #define DMemoryEntryUnchain(pM) \
167     if (pM->pNext) pM->pNext->pPrev = pM->pPrev; \
168     if (pM->pPrev) pM->pPrev->pNext = pM->pNext
169
170
171 //creates an memory entry
172 TAPrealMMUMemory * MemoryEntry_create () {
173     TAPrealMMUMemory * pM = NULL;
174
175     pM = (TAPrealMMUMemory *) malloc(sizeof(
        TAPrealMMUMemory));
176     if (!pM) return NULL;
177     pM->pData = NULL;
178     pM->count = 0;
179     pM->pNext = NULL;
180     pM->pPrev = NULL;
181
182     return pM;
183 }
184
185 //deletes an memory Entry
186 void MemoryEntry_delete (
187     TAPrealMMUMemory * pM // the memory to
        delete
188 )
189 {
190     // put the entry out of the chain
191     DMemoryEntryUnchain(pM);
192     // now we delete it
193     free(pM);
194 }
195

```

```

196 // =====
197 // mmu helper
198 // =====
199
200 //alloc if needed a new memory entry
201 TAPrealMMUMemory * MMU_helper_createMemoryEntry (
202     TAPrealMMU *    pMMU           // MMU
203     structure to init
204 )
205 {
206     // check if we have to alloc a new memory entry
207     if (!pMMU->pUnusedList) return
208     MemoryEntry_create();
209     // no there is some left at the list
210     TAPrealMMUMemory * pM;
211     // take the first one
212     pM = pMMU->pUnusedList;
213     // reset the list
214     pMMU->pUnusedList = pM->pNext;
215     // now unchain the element (for sure)
216     DMemoryEntryUnchain(pM);
217     // set the element pointers
218     pM->pNext = NULL;
219     pM->pPrev = NULL;
220     return pM;
221 }
222
223 //the garbage collector
224 void MMU_helper_garbageCollector (
225     TAPrealMMU *    pMMU           // MMU
226     structure to init
227 )
228 {
229     TAPrealMMUMemory * pM = pMMU->pStart;
230     int * pD = pMMU->memory;
231     while (pM) {
232         // check if we have to move the data
233         if (pD != pM->pData) {
234             // move the data
235             memmove(pD, pM->pData, pM->count *
236                 sizeof(int));
237         }
238         // reset the destination pointer
239         pD += pM->count;
240         pM = pM->pNext;
241     }
242     // compressing memory finished
243     // set the mmu vars new
244     pMMU->elementsAvailable = pMMU->totalAvailable;
245     pMMU->pUnusedData = pD;
246 }
247
248 // create a mmu

```

```

247 TAPMMU AP_MMU_create (size_t elementsNumber) {
248     TAPrealMMU * pMMU;
249
250     pMMU = (TAPrealMMU *) malloc (sizeof(TAPrealMMU)
251 );
252     if (!pMMU) return NULL;
253
254     // setup lists
255     pMMU->pStart = NULL;
256     pMMU->pEnd = NULL;
257     pMMU->pUnusedList = NULL;
258
259     pMMU->elementsAvailable = elementsNumber;
260     pMMU->pUnusedData = (int *) malloc (pMMU->
261         elementsAvailable);
262     pMMU->totalAvailable = pMMU->elementsAvailable;
263     return pMMU;
264 }
265
266 // destroying the mmu
267 void AP_MMU_delete (TAPMMU mmu) {
268     TAPrealMMU * pMMU = (TAPrealMMU *) mmu;
269
270     TAPrealMMUMemory * pM;
271     TAPrealMMUMemory * pMnext;
272
273     // 1. delete al mmu entry's
274     // 1.1 unused entry
275     pM = pMMU->pUnusedList;
276     while (pM) {
277         pMnext = pM->pNext;
278         MemoryEntry_delete(pM);
279         pM = pMnext;
280     }
281     pMMU->pUnusedList = NULL;
282     // 1.2 used blocks
283     pM = pMMU->pStart;
284     while (pM) {
285         pMnext = pM->pNext;
286         MemoryEntry_delete(pM);
287         pM = pMnext;
288     }
289     pMMU->pStart = NULL;
290     pMMU->pEnd = NULL;
291     // 2. delete mmu memory
292     free (pMMU->memory);
293
294 }
295
296 // getting memmory from the mmu
297 TAPMMUmemory AP_MMU_get (TAPMMU mmu, size_t elements) {
298     TAPrealMMU * pMMU = (TAPrealMMU *) mmu;
299

```



```

300 // check if there is enough space
301 if (pMMU->totalAvailable < elements) goto error;
302 // check if we have to use the garbage collector
303 if (pMMU->elementsAvailable < elements) {
304     // start garbage collector
305     MMU_helper_garbageCollector(pMMU);
306 }
307 // we have enough memory so let's allocate some
308
309 // get a new entry
310 TAPrealMMUMemory * pM;
311 pM = MMU_helper_createMemoryEntry(pMMU);
312 if (!pM) return NULL;
313 // get some memory
314 pM->pData = pMMU->pUnusedData;
315 pM->count = elements;
316 // refresh data
317 pMMU->pUnusedData += elements;
318 pMMU->totalAvailable -= elements;
319 pMMU->elementsAvailable -= elements;
320 // insert memory element at the end of the list
    and update last element
321 pM->pPrev = pMMU->pEnd;
322 if (pMMU->pEnd) pMMU->pEnd->pNext = pM;
323 if (!pMMU->pStart) pMMU->pStart = pM;
324 pMMU->pEnd = pM;
325
326 return pM;
327 error:
328
329 return NULL;
330 }
331
332 // free memory from the mmu
333 void AP_MMU_free (TAPMMU mmu, TAPMMUmemory memory) {
334     TAPrealMMU * pMMU = (TAPrealMMU *) mmu;
335     TAPrealMMUMemory * pM = (TAPrealMMUMemory *)
        memory;
336
337     if (!pM) return;
338     // set mmu settings
339     if (pMMU->pStart == pM) pMMU->pStart = pM->pNext
        ;
340     if (pMMU->pEnd == pM) pMMU->pEnd = pM->pPrev;
341     // unchain element
342     DMemoryEntryUnchain (pM);
343     // and put it to the chain of unused
344     pM->pNext = pMMU->pUnusedList;
345     pM->pPrev = NULL;
346     if (pMMU->pUnusedList) {
347         pMMU->pUnusedList->pPrev = pM;
348     }
349     pMMU->pUnusedList = pM;
350     // now set the mmu data new
351     pMMU->totalAvailable += pM->count;

```

```

352 }
353 }
354
355 // getting access to the MMU data
356 void * AP_MMU_getData (TAPMMUmemory memory) {
357     TAPrealMMUMemory * pM = (TAPrealMMUMemory *)
        memory;
358     return pM->pData;
359 }
360
361 // the real interpreter
362 typedef struct SAPrealInterpreter {
363     TAP *
        pAP; //
        // pointer to the audio processor
364     int
        state; // state of the IP
365     int
        nextState; // the next state of the IP
366     TAPInterpreterCPU cpu; // the IP
        core
367     TAPInterpreterFuncCall * code; // the code
368     int32_t
        instructionCount; // number of
        instructions at the code
369     TAPInterpreterVariable * variables; // the variables
370     int32_t
        variableCount; // number of
        the variables
371 } TAPrealInterpreter;
372
373 // create a new interpreter
374 TAPInterpreter APIInterpreterCreate (void * pAP) {
375     TAPrealInterpreter * pIP = NULL;
376     pIP = (TAPrealInterpreter *) malloc (sizeof(
        TAPrealInterpreter));
377     if (!pIP) return NULL;
378
379     pIP->pAP = pAP;
380     pIP->state = eAPIInterpreterState_idle;
381     pIP->nextState = eAPIInterpreterState_idle;
382     pIP->cpu.IP = pIP;
383     pIP->cpu.CF = 0;
384     pIP->cpu.EF = 0;
385     pIP->cpu.pCodeStart = NULL;
386     pIP->cpu.pCodeEnd = NULL;
387     pIP->cpu.pIP = NULL;
388
389     pIP->code = NULL;
390     pIP->instructionCount = 0;

```

```

391         pIP->variables = NULL;
392         pIP->variableCount = 0;
393
394         return pIP;
395     }
396
397     // cleans the interpreter
398     void APIInterpreterClean (TAPIInterpreter IP) {
399         TAPrealInterpreter * pIP = (TAPrealInterpreter
400             *) IP;
401
402         // clean code
403         if (pIP->code) {
404             free (pIP->code);
405             pIP->code = NULL;
406         }
407         pIP->instructionCount = 0;
408
409         // clean variables
410         TAPIInterpreterVariable * pV = pIP->variables;
411         int i;
412         for (i = 0; i < pIP->variableCount; i++) {
413             if (pV->pVI) pV->pVI->pFkt_delete(pV->
414                 pData);
415             pV++;
416         }
417         if (pIP->variables) {
418             free (pIP->variables);
419             pIP->variables = NULL;
420         }
421         pIP->variableCount = 0;
422     }
423
424     // deletes the interpreter
425     void APIInterpreterDelete (TAPIInterpreter IP) {
426         TAPrealInterpreter * pIP = (TAPrealInterpreter
427             *) IP;
428         APIInterpreterClean(IP);
429         free (pIP);
430     }
431
432     int APIInterpreterStateRun(TAPIInterpreter IP) {
433         TAPrealInterpreter * pIP = (TAPrealInterpreter
434             *) IP;
435         TAPIInterpreterFuncCall * pFC;
436         int cc; // cycle counter
437
438         // setup cpu
439         pIP->cpu.CF = 0;
440         pIP->cpu.EF = 0;
441         pIP->cpu.pIP = pIP->code;
442         pIP->cpu.pCodeStart = pIP->code;
443         pIP->cpu.pCodeEnd = pIP->code + pIP->

```

```

442         instructionCount;
443     // run code
444     cc = 10;
445     while (eAPIInterpreterState_run == pIP->state) {
446         pFC = pIP->cpu.pIP;
447         // check if we reached the end of the
448         // code
449         if (pFC > pIP->cpu.pCodeEnd) {
450             return 0;
451         }
452         // execute command
453         pFC->pHALFkt (&(pIP->cpu), pFC->param);
454         // check error flags
455         if (pIP->cpu.EF) {
456             return -1;
457         }
458         // check message system after x cycles
459         if (!cc) {
460             APMesssageProcess(pIP->pAP);
461             cc = 100;
462         } else {
463             cc--;
464         }
465     }
466     return 1;
467 }
468
469 // process the actual state
470 int APIInterpreterProcessState(TAPIInterpreter IP){
471     TAPrealInterpreter * pIP = (TAPrealInterpreter
472 *) IP;
473     pIP->state = pIP->nextState;
474     int rc = 0;
475
476     switch (pIP->state) {
477         case eAPIInterpreterState_idle:
478             break;
479         case eAPIInterpreterState_loadProgramm:
480             break;
481         case eAPIInterpreterState_run:
482             rc = APIInterpreterStateRun(IP);
483             if (rc >= 0) pIP->state =
484                 eAPIInterpreterState_idle;
485             break;
486         case eAPIInterpreterState_oneStep:
487             break;
488         case eAPIInterpreterState_halt:
489             break;
490         default:
491             return -10;
492     }
493     return rc;

```

```

493 }
494
495
496 // set interpreter state
497 int APInterpreterSetState (TAPInterpreter IP, int
    msgEndian, int32_t state) {
498     TAPrealInterpreter * pIP = (TAPrealInterpreter
        *) IP;
499     if (msgEndian != gAPendianFlag) {
500         APendianConversation32Bit((uint32_t *)&
            state, msgEndian);
501     }
502     pIP->nextState = (int) state;
503     return 0;
504 }
505
506 // setup the interpreter for a new program
507 int APInterpreterInitNewProgramm (TAPInterpreter IP, int
    msgEndian, int32_t instructionsNumber, int32_t
    VariableNumber) {
508     TAPrealInterpreter * pIP = (TAPrealInterpreter
        *) IP;
509     int i;
510
511     APInterpreterClean (IP);
512
513     if (msgEndian != gAPendianFlag) {
514         APendianConversation32Bit((uint32_t *)&
            instructionsNumber, msgEndian);
515         APendianConversation32Bit((uint32_t *)&
            VariableNumber, msgEndian);
516     }
517
518     pIP->code = (TAPInterpreterFuncCall *) malloc(
        sizeof(TAPInterpreterFuncCall)*
            instructionsNumber);
519     pIP->instructionCount = instructionsNumber;
520
521     pIP->variables = (TAPInterpreterVariable *)
        malloc(sizeof(TAPInterpreterVariable) * (
            VariableNumber));
522     for (i = 0; i < VariableNumber; i++) {
523         pIP->variables[i].pData = NULL;
524         pIP->variables[i].pVI = NULL;
525     }
526     pIP->variableCount = VariableNumber;
527
528     return 0;
529 }
530
531 // load a variable/~array to an index
532 int APInterpreterLoadVar (TAPInterpreter IP, int
    msgEndian, int32_t index, int32_t varTypeID, int32_t
    numberOfElements)
533 {

```

```

534     TAPrealInterpreter * pIP = (TAPrealInterpreter
        *) IP;
535     if (msgEndian != gAPendianFlag) {
536         APendianConversation32Bit((uint32_t *)&
            index, msgEndian);
537         APendianConversation32Bit((uint32_t *)&
            varTypeID, msgEndian);
538         APendianConversation32Bit((uint32_t *)&
            numberOfElements, msgEndian);
539     }
540
541     if ((index < 0) || (index > pIP->variableCount))
        return -1;
542
543     // set pointer to the runtime variable
544     TAPInterpreterVariable * pRTV = pIP->variables +
        index;
545     THAL_Variable const * pV = HALfindVar(varTypeID)
        ;
546     if (!pV) return -2;
547
548     pRTV->pData = pV->pFkt_create((unsigned int)
        numberOfElements);
549     //if (!pRTV->pData) return -3;
550
551     pRTV->pVI = pV;
552     return 0;
553 }
554
555 // load a single Instruction to an index
556 int APInterpreterLoadInstr (TAPInterpreter IP, int
    msgEndian, int32_t index, int32_t * pRawInstr)
557 {
558     TAPrealInterpreter * pIP = (TAPrealInterpreter
        *) IP;
559     if (msgEndian != gAPendianFlag) {
560         APendianConversation32Bit((uint32_t *)&
            index, msgEndian);
561     }
562     if ((index < 0) || (index > pIP->
        instructionCount)) return -1;
563     TAPInterpreterFuncCall * pIFC = pIP->code +
        index;
564     memset (pIFC, 0, sizeof(TAPInterpreterFuncCall))
        ;
565
566     // get function
567     int32_t fid = *pRawInstr;
568     if (msgEndian != gAPendianFlag) {
569         APendianConversation32Bit((uint32_t *)&
            fid, msgEndian);
570     }
571     THALFunction const * pF = HALfindFunction(fid);
572     if (!pF) return -2;
573     pIFC->pHALFkt = pF->pFktHAL;

```

```

574         // convert parameters
575         pRawInstr++; // set to the first parameter
576         int i;
577         THALFunctionParam const * pP = pF->paramList.pL;
578         TuAPIInterpreterFunctionParameter * pIFP = pIFC->
579             param;
580         for (i = 0; i < pF->paramList.number; i++) {
581             if (APconvertRawParamData (msgEndian,
582                 pRawInstr, pP, pIFP, pIP->variables))
583                 return -3;
584             pP++;
585             pRawInstr++;
586             pIFP++;
587         }
588         return 0;
589     }
590     // gets the variable by it's index
591     TAPIInterpreterVariable * APIInterpreterGetVariableByIndex
592     (TAPIInterpreter IP, int index) {
593         return &(((TAPrealInterpreter *) IP)->variables[
594             index]);
595     }
596     // gets the AP from the IP
597     void * APIInterpreterGetAPfromIP (TAPIInterpreter IP) {
598         return ((TAPrealInterpreter *) IP)->pAP;
599     }
600     typedef struct SAPrealMsgSystem {
601         TAPMsg *          pOldRXMsg;
602             // pointer to the oldest received
603             messages
604         TAPMsg *          pNewRXMsg;
605             // pointer to the newest received
606             messages
607         TAPMMU            mmu;
608             // the mmu
609         int
610             sysEndianness; // the system endianness
611         int
612             messagecounter; // a counter for checkin if a
613             new message has been received
614     } TAPrealMsgSystem;
615
616     int SMinitial (
617         void *          pVoidSM,
618             // pointer to the
619             statemachine
620         uint32_t *      pD,

```

```

        data // pointer to the
        int number
614         // the number of data
        elements
        );
615
616 int SMdata (
617     void * pVoidSM,
618     // pointer to the
        statemachine
619     uint32_t * pD,
        // pointer to the
        data
620     int number
        // the number of data
        elements
        );
621
622 int SMmessageFinished (
623     void * pVoidSM
624     // pointer to the statemachine
        );
625
626
627
628 // create AP message system
629 TAPMsgSystem APMScreate (
630     TAPMMU mmu,
631     // the mmu
        int
632     sysEndianness // the system
        endianness
        ) {
633     TAPrealMsgSystem * pMS = (TAPrealMsgSystem *)
634         malloc (sizeof(TAPrealMsgSystem));
635     if (!pMS) return NULL;
636     pMS->mmu = mmu;
637     pMS->sysEndianness = sysEndianness;
638     pMS->pOldRXMsg = NULL;
639     pMS->pNewRXMsg = NULL;
640     pMS->messagecounter = 0;
641
642     return pMS;
643 }
644
645 void APMSdelete (
646     TAPMsgSystem ms
647 ) {
648     TAPrealMsgSystem * pMS = (TAPrealMsgSystem *) ms
649     ;
650     free (pMS);
651 }
652

```



```

653
654 // frees a message from the message system
655 void APMSdeleteMsg (
656     TAPMsgSystem      ms,
657     TAPMsg *           pM
658 ) {
659     TAPrealMsgSystem * pMS = (TAPrealMsgSystem *) ms
        ;
660     AP_MMU_free(pMS->mmu,pM->memory);
661 }
662
663 // get memory for a new message
664 TAPMsg * APMSgetNewMsg (
665     TAPrealMsgSystem * pMsgSys,
666     int
        dataElementsNumber,
667     const TAPMsgDrv *   pDrv
668 ) {
669     TAPMMUmemory m = AP_MMU_get (pMsgSys->mmu,
        sizeof(TAPMsg)/sizeof(int) +
        dataElementsNumber*sizeof(uint32_t)/sizeof(
        int));
670     if (!m) return NULL;
671
672     int * pRD = (int *) AP_MMU_getData(m);
673     TAPMsg * pM = (TAPMsg *) pRD;
674     pM->memory = m;
675     pM->extraData.pDrv = pDrv;
676     pM->pH = (TAPMsgHeader *)((int *) pRD + sizeof(
        TAPMsg)/sizeof(int));
677     pM->pData = (int *)pM->pH + sizeof(TAPMsgHeader)
        /sizeof(int);
678     pM->pNext = NULL;
679     return pM;
680 }
681
682 // insert a new message into the message queue
683 void APMSinsertMsg (
684     TAPrealMsgSystem * pMS,
685     TAPMsg *           pM
686 ) {
687
688     if (pMS->pNewRXMsg) {
689         pMS->pNewRXMsg->pNext = pM;
690     }
691     pMS->pNewRXMsg = pM;
692     if (!pMS->pOldRXMsg) {
693         pMS->pOldRXMsg = pM;
694     }
695     pMS->messagecounter++;
696 }
697
698
699 // unchains a received message
700 void APMSunchainMessage (

```

```

701         TAPrealMsgSystem *      pMS,
702         TAPMsg *                  pM,
703         TAPMsg *                  pAntecessorM
704     ) {
705         if (pAntecessorM) {
706             pAntecessorM->pNext = pM->pNext;
707         } else {
708             pMS->pOldRXMsg = pM->pNext;
709         }
710         if (pM == pMS->pNewRXMsg) {
711             pMS->pNewRXMsg = NULL;
712         }
713
714         // now there is one message less left
715         pMS->messagecounter--;
716     }
717
718     // get oldest message
719     TAPMsg * APMSgetMsg (
720         TAPMsgSystem          ms,
721         TAPMessageID          msgID,
722         TAPNodeID             sender,
723         uint32_t              mNumber,
724         int                   ackMsgAllowed
725     ) {
726
727         TAPrealMsgSystem * pMS = (TAPrealMsgSystem *) ms
728         ;
729         // flags
730         int senderOK;
731         int msgIDok;
732         int numberOK;
733         // temp vars
734         TAPMsg * pM;
735         TAPMsg * pAntecessorM;
736         uint32_t * pH;
737
738         // result var
739         TAPMsg * res = NULL;
740     checkMessages:
741
742         // search msg list
743         pM = pMS->pOldRXMsg;
744         pAntecessorM = NULL;
745

```

```

746         if (!pM) goto waitForMessage;
747
748
749         senderOK = 0;
750         msgIDok = 0;
751         numberOK = 0;
752
753         pH = *(pM->pH);
754
755         if (!sender) {
756             senderOK = 1;
757         } else {
758             if (pH[eAPMsgHeaderPosition_sender] ==
759                 sender) senderOK = 1;
760         }
761         if (!msgID) {
762             // filter ack/nack msg
763             if (ackMsgAllowed) {
764                 msgIDok = 1;
765             } else {
766                 if (
767
768                     (pH[
769                         eAPMsgHeaderPosition_msgTypeID
770                     ] !=
771                         eAPMsgTypes_ACK
772                     ) &&
773                     (pH[
774                         eAPMsgHeaderPosition_msgTypeID
775                     ] !=
776                         eAPMsgTypes_NACK
777                     )
778                 ) {
779                     msgIDok = 1;
780                 }
781             }
782         } else {
783             if (pH[eAPMsgHeaderPosition_msgTypeID]
784                 == msgID) msgIDok = 1;
785         }
786         if (!mNumber) {
787             numberOK = 1;
788         } else {
789             if (pH[eAPMsgHeaderPosition_msgNumber]
790                 == mNumber) numberOK = 1;
791         }
792         if ((senderOK) && (msgIDok) && (numberOK)) {
793             res = pM;
794             goto exit;
795         }
796         pAntecessorM = pM;
797         pM = pM->pNext;
798         if (pM) goto checkMessages;
799 waitForMessage:
800     goto checkMessages;
801

```

```

790 exit:
791     // unchain message
792     APMSunchainMessage(pMS,pM,pAntecessorM);
793     // now one thread is less waiting for a message
794     return res;
795 error:
796     return NULL;
797 }
798
799
800 // wait till a new message has been received
801 void APMSwaitForNewMessage (TAPrealMsgSystem * pMS)
802 {
803     volatile int mc;
804
805     mc = pMS->messagecounter;
806
807     while (mc == pMS->messagecounter) {
808
809     }
810 }
811
812 // returns 0 if a new message is available
813 inline int APMSisMessageAvailble (TAPrealMsgSystem * pMS
814     ) {
815     return (!pMS->pOldRXMsg) ? 0 : -1;
816 }
817
818 // =====
819 // the receive state machine
820 // =====
821
822 // the receive state machine state function for
823 // receiving the msg header
824 int SMinitial (
825     void *                pVoidSM,
826                         // pointer to the
827                         statemachine
828     uint32_t *            pD,
829                         // pointer to the
830                         data
831     int                   number
832                         // the number of data
833                         elements
834 ) {
835     TAPReceiveStateMachine *
836     pSM = (TAPReceiveStateMachine *) pVoidSM;
837     TAPrealMsgSystem *
838     pMS = (TAPrealMsgSystem *) pSM->pMS;
839
840     int
841
842     copyAmount = number;
843
844     int
845
846     i;
847
848     // 1. try to copy the data to the header

```

```

833     if (pSM->elementsLeft < copyAmount) copyAmount =
        pSM->elementsLeft;
834     // copy
835     for (i = 0; i < copyAmount; i++) {
836         *pSM->pD = *pD;
837         pSM->pD++;
838         pD++;
839     }
840     pSM->elementsLeft -= copyAmount;
841
842     // check if we have to change the statemachine
        because we received the header
843     if (pSM->elementsLeft) return 0;
844
845     // yes! alloc msg buffer and (opt.) transfer
        data
846
847     // 2. convert endian
848     int msgEndian = pSM->header[
        eAPMsgHeaderPosition_endian];
849     if (pMS->sysEndianness != msgEndian) {
850         for (i = 0; i <
            eAPMsgHeaderPosition_headerElementNumber
            ; i++) {
851             APendianConversation32Bit(&pSM->
                header[i], msgEndian);
852         }
853     }
854     // 3. now alloc message
855     // 3.1 get length
856     int msgElementNumber = (int) pSM->header[
        eAPMsgHeaderPosition_length];
857     // 3.2. get memory
858     pSM->pMsg = APMSgetNewMsg (pMS, msgElementNumber,
        pSM->pDrv);
859     if (!pSM->pMsg) return -100;
860
861     // 3.3 check getMemory result
862     if (!pSM->pMsg) return -1;
863     // copy message header
864     pSM->pD = (uint32_t *) pSM->pMsg->pH;
865     for (i = 0; i <
        eAPMsgHeaderPosition_headerElementNumber; i++)
        {
866         *pSM->pD = pSM->header[i];
867         pSM->pD++;
868     }
869     pSM->elementsLeft = pSM->header[
        eAPMsgHeaderPosition_length];
870     // set up the data
871     // 1. check if there is an data element
872     if (!pSM->elementsLeft) {
873         // no! now finish the message
874         return SMmessageFinished(pVoidSM);
875     }

```

```

876 // 2. yes
877 // 2.1 setup the sm for the data receiving
878 pSM->state = SMdata;
879 // 2.2 now check if we have to copy some data
880 number -= copyAmount;
881 if (number) {
882     // set the data pointer
883     pD += copyAmount;
884     // and copy the data
885     return SMdata (pVoidSM,pD,number);
886 }
887 return 0;
888 }
889
890
891 // the receive state machine state function for
892 // receiving the data
893 int SMdata (
894     void *                pVoidSM,
895     // pointer to the
896     statemachine
897     uint32_t *            pD,
898     // pointer to the
899     data
900     int                    number
901     // the number of data
902     elements
903 ) {
904     TAPReceiveStateMachine *
905     pSM = (TAPReceiveStateMachine *) pVoidSM;
906     int
907     copyAmount = number;
908     int
909     i;
910     // 1. transfer the data
911     // do some clipping
912     if (pSM->elementsLeft < copyAmount) copyAmount =
913     pSM->elementsLeft;
914     // copy
915     for (i = 0; i < copyAmount;i++) {
916         *pSM->pD = *pD;
917         pSM->pD++;
918         pD++;
919     }
920     // set statemachine work data
921     pSM->elementsLeft -= copyAmount;
922     // check if we have to change the statemachine
923     if (pSM->elementsLeft) return 0;
924     int res = SMmessageFinished (pVoidSM);
925     if (res) return res;
926
927     // check if there some bytes left to copy
928     number -= copyAmount;
929     if (number) {
930         // set the data pointer

```

```

920         pD += copyAmount;
921         // and copy the data
922         return pSM->state (pVoidSM,pD,number);
923     }
924     return 0;
925 }
926
927 // this function is called when all data have been
928 // received
929 int SMmessageFinished (
930     void * pVoidSM
931     // pointer to the statemachine
932 ) {
933     TAPReceiveStateMachine *
934     pSM = (TAPReceiveStateMachine *) pVoidSM;
935     TAPMsg *
936     pM;
937     // 1. reset SM
938     // set the helper
939     pSM->elementsLeft =
940     eAPMsgHeaderPosition_headerElementNumber;
941     pSM->pD = pSM->header;
942     // data
943     pM = pSM->pMsg; // save msg info for inserting
944     pSM->pMsg = NULL;
945     // right state function
946     pSM->state = SInitial; // the state
947     // 2. insert message at the message system
948     APMSInsertMsg ((TAPrealMsgSystem *)pSM->pMS,pM);
949     return 0;
950 }
951
952 // inits the state machine
953 void APInitReceiveStateMachine (
954     TAPReceiveStateMachine *
955     pSM, // pointer to the
956     state machine
957     TAPMsgSystem
958     pMS, // pointer to
959     the message system
960     const TAPMsgDrv *
961     pDrv // the driver
962     associated with the statemachine
963 ) {
964     pSM->state = SInitial;
965     pSM->pMS = pMS;
966     pSM->pDrv = pDrv;
967     // set the helper
968     pSM->elementsLeft =
969     eAPMsgHeaderPosition_headerElementNumber;

```

```

963     pSM->pD = pSM->header;
964
965     // data
966     pSM->pMsg = NULL;
967 }
968
969 int APHandleMsg (
970     TAP *                pAP,
971     TAPMsg *             pM
972 ) {
973
974     TAPMessageID
975                                     msgID;
976     const THALMsgProcessMessageAssociation *
977         pMsgIDandFunctAsso;
978     int
979                                     i;
980
981     // get message id
982     msgID = (*(pM->pH))[
983         eAPMsgHeaderPosition_msgTypeID];
984     // search handler
985     pMsgIDandFunctAsso = gHALMsgProcessRXHandlers.pL
986         ;
987     for (i = 0; i < gHALMsgProcessRXHandlers.number;
988         i++) {
989         if (((TAPMessageID)pMsgIDandFunctAsso->
990             msgID) == msgID) {
991             return pMsgIDandFunctAsso->
992                 pfktHandle(pAP,pM);
993         }
994         pMsgIDandFunctAsso++;
995     }
996     return -1;
997 }
998
999 void APMessageProcessingThread (TAP * pAP) {
1000     TAPPrealMsgSystem *
1001                                     pMS = (
1002         TAPPrealMsgSystem *) pAP->MS;
1003     TAPMsg *
1004                                     pM;
1005     TAPNodeID
1006                                     recv;
1007     while (1) {
1008         // get the message
1009         pM = APMSgetMsg (pMS,0,0,0,0);
1010         if (!pM) goto error;
1011         // search the message handler
1012         recv = (*(pM->pH))[
1013             eAPMsgHeaderPosition_receiver];
1014         if ((recv == dAPNodeID_ALL) || (recv ==
1015             pAP->nodeID)) {
1016             if(APHandleMsg (pAP,pM)) goto
1017                 exit;
1018         }
1019         // free memory
1020         APMSdeleteMsg (pAP->MS,pM);

```



```

1006     }
1007 exit:
1008     // free memory
1009     APMSdeleteMsg (pAP->MS,pM);
1010 error:
1011     return;
1012 }
1013
1014
1015 // if a message is in the queue available it will be
1016 // processed
1017 void APMessageProcess (TAP * pAP) {
1018     TAPrealMsgSystem * pMS = (
1019         TAPrealMsgSystem *) pAP->MS;
1020     TAPMsg * pM;
1021     TAPNodeID recvID;
1022
1023     // search msg list
1024     pM = pMS->pOldRXMsg;
1025
1026     // if there is no message we will return
1027     if (!pM) return;
1028
1029     // unchain first message
1030     APMSunchainMessage(pMS, pM, NULL);
1031
1032     // check header
1033     recvID = (*(pM->pH))[
1034         eAPMsgHeaderPosition_receiver];
1035     if ((recvID == dAPNodeID_ALL) || (recvID == pAP
1036         ->nodeID)) {
1037         APHandleMsg (pAP,pM);
1038     }
1039     // free memory
1040     APMSdeleteMsg (pAP->MS,pM);
1041 }

```

3.5.4 audio processor blueprint 4 (libsndfile frame based)

Informations:

description:	a block based wavfile processing AP, x.wav in, y.wav out; both @48000kHz
--------------	---

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====
2 // AP uuid = 8
3 // =====

```

```

4
5 // the global var for the Frame WAV modul
6 TStjFrameWAVmodule gFrameWAVModule;
7
8 // inits the AP
9 int APinit (
10
11         TAP *
12         pAP,
13         TAPNodeID
14         nodeID,
15         const TAPMsgDrv *      pDrvList
16         ,
17         const int
18         driverNumber,
19         size_t
20         messagePoolSize,
21         int
22         sysEndian
23     )
24 {
25     pAP->nodeID = nodeID;
26     pAP->pNodeList = NULL;
27     pAP->pDrvList = pDrvList;
28     pAP->sysEndian = sysEndian;
29     pAP->driverNumber = driverNumber;
30     pAP->msgSysMMU = AP_MMU_create(messagePoolSize);
31     pAP->IP = APInterpreterCreate(pAP);
32     pAP->MS = APMScreate (pAP->msgSysMMU,sysEndian);
33     pAP->msgNumber = 0;
34     pAP->APstate = eAPstate_idle;
35
36     if(
37         (!pAP->msgSysMMU) ||
38         (!pAP->IP) ||
39         (!pAP->MS)
40     ) return -1;
41
42     TStjFrameWAVOpenInfo WI[] = {
43         {1,"x.wav",1,48000,1},
44         {2,"y.wav",0,48000,1}
45     };
46
47     if (FrameWAVmoduleInit(2,WI,&gFrameWAVModule))
48         return -2;
49
50     // login the ap to the message system
51     return TX_login(pAP);
52 }
53
54 // deletes the AP
55 void APdelete (TAP * pAP)
56 {
57     APMSdelete (pAP->MS);
58     APInterpreterDelete(pAP->IP);
59 }

```

```

52     AP_MMU_delete(pAP->msgSysMMU);
53     // closes the wav frame modul
54     FrameWAVmoduleExit(&gFrameWAVModule);
55 }
56
57 // find a node at the list
58 TAPNode * APfindNode(TAP * pAP, TAPNodeID nodeID) {
59     TAPNode * pN = pAP->pNodeList;
60     while (pN) {
61         if (pN->nodeID == nodeID) return pN;
62         pN = pN->pNext;
63     };
64     return NULL;
65 }
66
67 // adds a new node to the node list
68 int APaddNode(TAP * pAP, TAPNodeID newNodeID, const
69     TAPMsgDrv * pDrv) {
70     if (APfindNode(pAP, newNodeID)) return 1;
71     TAPNode * pN = (TAPNode *) malloc(sizeof(TAPNode
72     ));
73     if (!pN) return -1;
74     pN->nodeID = newNodeID;
75     pN->pDrv = pDrv;
76     pN->pNext = pAP->pNodeList;
77     pAP->pNodeList = pN;
78     return 0;
79 }
80
81 // removes a node from the node list
82 void APremoveNode(TAP * pAP, TAPNodeID nodeID){
83     TAPNode * pAntN = pAP->pNodeList; // antecessor
84     node
85     TAPNode * pActN = pAP->pNodeList; // actual node
86
87     while (pActN) {
88         // compare node id's
89         if (pActN->nodeID == nodeID) {
90             // unchain
91
92             // check if we at the first
93             position at the list
94             if (pAP->pNodeList == pAntN) {
95                 // reset the pointer
96                 pAP->pNodeList = pActN->
97                 pNext;
98             } else {
99                 // set the antecessor
100                 pAntN->pNext = pActN->
101                 pNext;
102             }
103             // free node
104             free(pActN);
105             // and abort
106             return;

```

```

101         }
102         // the actual element becomes the
           predecessor element
103         pAntN = pActN;
104         pActN = pActN->pNext;
105     }
106 }
107
108 // get a new message number
109 unsigned int APgetNewMessageNumber (TAP *pAP) {
110     pAP->msgNumber++;
111     return pAP->msgNumber;
112 }
113
114 // find the driver associated with der nodeID
115 const TAPMsgDrv * APfindDrvBySenderID (TAP * pAP,
           TAPNodeID node) {
116     TAPNode * pN = pAP->pNodeList;
117     while (pN) {
118         if (pN->nodeID == node) {
119             return pN->pDrv;
120         }
121         pN = pN->pNext;
122     }
123     return NULL;
124 }
125
126 // runs the AP
127 int APrun(TAP *pAP) {
128     pAP->APstate = eAPstate_run;
129     return 0;
130 }
131
132
133 typedef struct SAPrealMMUMemory {
134     int *
           pData; // the data
135     size_t
           count; // amount of data
           elements
136     struct SAPrealMMUMemory * pNext;
           // next element
137     struct SAPrealMMUMemory * pPrev;
           // previous element
138 } TAPrealMMUMemory;
139
140 //the mmu type
141 typedef struct SAPrealMMU {
142     int *
           // the memory bolck
           memory;
143     TAPrealMMUMemory * pStart;
           // first element
144     TAPrealMMUMemory * pEnd;
           // second element
145     TAPrealMMUMemory * pUnusedList;

```

```

146         // list with the unused elements
147         int *
            pUnusedData;          // pointer to the
            unused memory
148         size_t
            elementsAvailable;    // amount of elements
            witch are available without using the
            garbage collector
149         size_t
            totalAvailable;       // total amount of
            free bytes
150     } TAPrealMMU;
151
152     // =====
153     // memory entry functions
154     // =====
155
156     // a little macro for unchaining an element
157     #define DMemoryEntryUnchain(pM) \
158         if (pM->pNext) pM->pNext->pPrev = pM->pPrev; \
159         if (pM->pPrev) pM->pPrev->pNext = pM->pNext
160
161
162     //creates an memory entry
163     TAPrealMMUMemory * MemoryEntry_create () {
164         TAPrealMMUMemory * pM = NULL;
165
166         pM = (TAPrealMMUMemory *) malloc(sizeof(
            TAPrealMMUMemory));
167         if (!pM) return NULL;
168         pM->pData = NULL;
169         pM->count = 0;
170         pM->pNext = NULL;
171         pM->pPrev = NULL;
172
173         return pM;
174     }
175
176     //deletes an memory Entry
177     void MemoryEntry_delete (
178         TAPrealMMUMemory * pM    // the memory to
            delete
179     )
180     {
181         // put the entry out of the chain
182         DMemoryEntryUnchain(pM);
183         // now we delete it
184         free(pM);
185     }
186
187     // =====
188     // mmu helper
189     // =====
190

```

```

191 //alloc if needed a new memory entry
192 TAPrealMMUMemory * MMU_helper_createMemoryEntry (
193     TAPrealMMU *    pMMU           // MMU
194     )
195 {
196     // check if we have to alloc a new memory entry
197     if (!pMMU->pUnusedList) return
198         MemoryEntry_create();
199     // no there is some left at the list
200     TAPrealMMUMemory * pM;
201     // take the first one
202     pM = pMMU->pUnusedList;
203     // reset the list
204     pMMU->pUnusedList = pM->pNext;
205     // now unchain the element (for sure)
206     DMemoryEntryUnchain(pM);
207     // set the element pointers
208     pM->pNext = NULL;
209     pM->pPrev = NULL;
210     return pM;
211 }
212 //the garbage collector
213 void MMU_helper_garbageCollector (
214     TAPrealMMU *    pMMU           // MMU
215     )
216 {
217     TAPrealMMUMemory * pM = pMMU->pStart;
218     int * pD = pMMU->memory;
219     while (pM) {
220         // check if we have to move the data
221         if (pD != pM->pData) {
222             // move the data
223             memmove(pD, pM->pData, pM->count*
224                 sizeof(int));
225         }
226         // reset the destination pointer
227         pD += pM->count;
228         pM = pM->pNext;
229     }
230     // compressing memory finished
231     // set the mmu vars new
232     pMMU->elementsAvailable = pMMU->totalAvailable;
233     pMMU->pUnusedData = pD;
234 }
235
236 // create a mmu
237 TAPMMU AP_MMU_create (size_t elementsNumber) {
238     TAPrealMMU * pMMU;
239
240     pMMU = (TAPrealMMU *) malloc (sizeof(TAPrealMMU))
241

```

```

242         );
243         if (!pMMU) return NULL;
244
245         // setup lists
246         pMMU->pStart = NULL;
247         pMMU->pEnd = NULL;
248         pMMU->pUnusedList = NULL;
249
250         pMMU->elementsAvailable = elementsNumber;
251         pMMU->pUnusedData = (int *) malloc (pMMU->
252             elementsAvailable);
253         pMMU->totalAvailable = pMMU->elementsAvailable;
254         return pMMU;
255     }
256
257     // destroying the mmu
258     void AP_MMU_delete (TAPMMU mmu) {
259         TAPrealMMU * pMMU = (TAPrealMMU *) mmu;
260
261         TAPrealMMUMemory * pM;
262         TAPrealMMUMemory * pMnext;
263
264         // 1. delete al mmu entry's
265         // 1.1 unused entry
266         pM = pMMU->pUnusedList;
267         while (pM) {
268             pMnext = pM->pNext;
269             MemoryEntry_delete(pM);
270             pM = pMnext;
271         }
272         pMMU->pUnusedList = NULL;
273         // 1.2 used blocks
274         pM = pMMU->pStart;
275         while (pM) {
276             pMnext = pM->pNext;
277             MemoryEntry_delete(pM);
278             pM = pMnext;
279         }
280         pMMU->pStart = NULL;
281         pMMU->pEnd = NULL;
282         // 2. delete mmu memory
283         free (pMMU->memory);
284
285     }
286
287     // getting memmory from the mmu
288     TAPMMUmemory AP_MMU_get (TAPMMU mmu, size_t elements) {
289         TAPrealMMU * pMMU = (TAPrealMMU *) mmu;
290
291         // check if there is enough space
292         if (pMMU->totalAvailable < elements) goto error;
293         // check if we have to use the garbage collector
294         if (pMMU->elementsAvailable < elements) {

```

```

295         // start garbage collector
296         MMU_helper_garbageCollector(pMMU);
297     }
298     // we have enough memory so let's allocate some
299
300     // get a new entry
301     TAPrealMMUMemory * pM;
302     pM = MMU_helper_createMemoryEntry(pMMU);
303     if (!pM) return NULL;
304     // get some memory
305     pM->pData = pMMU->pUnusedData;
306     pM->count = elements;
307     // refresh data
308     pMMU->pUnusedData += elements;
309     pMMU->totalAvailable -= elements;
310     pMMU->elementsAvailable -= elements;
311     // insert memory element at the end of the list
312     // and update last element
313     pM->pPrev = pMMU->pEnd;
314     if (pMMU->pEnd) pMMU->pEnd->pNext = pM;
315     if (!pMMU->pStart) pMMU->pStart = pM;
316     pMMU->pEnd = pM;
317
318     return pM;
319
320 error:
321     return NULL;
322 }
323
324 // free memory from the mmu
325 void AP_MMU_free (TAPMMU mmu, TAPMMUmemory memory) {
326     TAPrealMMU * pMMU = (TAPrealMMU *) mmu;
327     TAPrealMMUMemory * pM = (TAPrealMMUMemory *)
328         memory;
329
330     if (!pM) return;
331     // set mmu settings
332     if (pMMU->pStart == pM) pMMU->pStart = pM->pNext;
333     ;
334     if (pMMU->pEnd == pM) pMMU->pEnd = pM->pPrev;
335     // unchain element
336     DMemoryEntryUnchain (pM);
337     // and put it to the chain of unused
338     pM->pNext = pMMU->pUnusedList;
339     pM->pPrev = NULL;
340     if (pMMU->pUnusedList) {
341         pMMU->pUnusedList->pPrev = pM;
342     }
343     pMMU->pUnusedList = pM;
344     // now set the mmu data new
345     pMMU->totalAvailable += pM->count;
346 }
347
348 // getting access to the MMU data

```



```

347 void * AP_MMU_getData (TAPMMUmemory memory) {
348     TAPrealMMUMemory * pM = (TAPrealMMUMemory *)
        memory;
349     return pM->pData;
350 }
351
352 // the real interpreter
353 typedef struct SAPrealInterpreter {
354     int
        state;
        // state of the IP
355     int
        nextState;
        // the next state of the IP
356     TAPInterpreterCPU cpu;
        // the IP
        core
357     TAPInterpreterFuncCall * code;
        // the code
358     int32_t
        instructionCount;
        // number of
        instructions at the code
359     TAPInterpreterVariable * variables;
        // the variables
360     int32_t
        variableCount;
        // number of
        the variables
361     int
        sysEndian;
        // endian of the system
362 } TAPrealInterpreter;
363
364 // create a new interpreter
365 TAPInterpreter APIInterpreterCreate (void * pAP) {
366     TAPrealInterpreter * pIP = NULL;
367     pIP = (TAPrealInterpreter *) malloc (sizeof(
        TAPrealInterpreter));
368     if (!pIP) return NULL;
369
370     pIP->state = eAPIInterpreterState_idle;
371     pIP->nextState = eAPIInterpreterState_idle;
372     pIP->sysEndian = ((TAP *)pAP)->sysEndian;
373     pIP->cpu.CF = 0;
374     pIP->cpu.EF = 0;
375     pIP->cpu.pCodeStart = NULL;
376     pIP->cpu.pCodeEnd = NULL;
377     pIP->cpu.pIP = NULL;
378
379     pIP->code = NULL;
380     pIP->instructionCount = 0;
381
382     pIP->variables = NULL;
383     pIP->variableCount = 0;
384
385     return pIP;

```

```

386 }
387
388 // cleans the interpreter
389 void APIInterpreterClean (TAPIInterpreter IP) {
390     TAPrealInterpreter * pIP = (TAPrealInterpreter
391         *) IP;
392
393     // clean code
394     if (pIP->code) {
395         free (pIP->code);
396         pIP->code = NULL;
397     }
398     pIP->instructionCount = 0;
399
400     // clean variables
401     TAPIInterpreterVariable * pV = pIP->variables;
402     int i;
403     for (i = 0; i < pIP->variableCount; i++) {
404         if (pV->pVI) pV->pVI->pFkt_delete(pV->
405             pData);
406         pV++;
407     }
408     if (pIP->variables) {
409         free (pIP->variables);
410         pIP->variables = NULL;
411     }
412     pIP->variableCount = 0;
413 }
414
415 // deletes the interpreter
416 void APIInterpreterDelete (TAPIInterpreter IP) {
417     TAPrealInterpreter * pIP = (TAPrealInterpreter
418         *) IP;
419     APIInterpreterClean(IP);
420     free (pIP);
421 }
422
423 int APIInterpreterStateRun(TAPIInterpreter IP) {
424     TAPrealInterpreter * pIP = (TAPrealInterpreter
425         *) IP;
426     TAPIInterpreterFuncCall * pFC;
427
428     // setup cpu
429     pIP->cpu.CF = 0;
430     pIP->cpu.EF = 0;
431     pIP->cpu.pIP = pIP->code;
432     pIP->cpu.pCodeStart = pIP->code;
433     pIP->cpu.pCodeEnd = pIP->code + pIP->
434         instructionCount;
435
436     // run code
437     while (eAPIInterpreterState_run == pIP->state) {
438         pFC = pIP->cpu.pIP;
439         // check if we reached the end of the

```

```

436         code
437         if (pFC > pIP->cpu.pCodeEnd) {
438             return 0;
439         }
440         // execute command
441         pFC->pHALFkt (&(pIP->cpu), pFC->param);
442         // check error flags
443         if (pIP->cpu.EF) {
444             return -1;
445         }
446     }
447     return 1;
448 }
449
450 // process the actual state
451 int APInterpreterProcessState(TAPInterpreter IP){
452     TAPrealInterpreter * pIP = (TAPrealInterpreter
453         *) IP;
454     pIP->state = pIP->nextState;
455     int rc = 0;
456
457     switch (pIP->state) {
458         case eAPInterpreterState_idle:
459             break;
460         case eAPInterpreterState_loadProgramm:
461             break;
462         case eAPInterpreterState_run:
463             rc = APInterpreterStateRun(IP);
464             if (rc >= 0) pIP->state =
465                 eAPInterpreterState_idle;
466             break;
467         case eAPInterpreterState_oneStep:
468             break;
469         case eAPInterpreterState_halt:
470             break;
471         default:
472             return -10;
473     }
474     return rc;
475 }
476
477 // set interpreter state
478 int APInterpreterSetState (TAPInterpreter IP, int
479     msgEndian, int32_t state) {
480     TAPrealInterpreter * pIP = (TAPrealInterpreter
481         *) IP;
482     if (msgEndian != pIP->sysEndian) {
483         APendianConversation32Bit((uint32_t *)&
484             state);
485     }
486     pIP->nextState = (int) state;
487     return 0;
488 }

```

```

485
486 // setup the interpreter for a new program
487 int APInterpreterInitNewProgramm (TAPInterpreter IP, int
msgEndian, int32_t instructionsNumber, int32_t
VariableNumber) {
488     TAPrealInterpreter * pIP = (TAPrealInterpreter
*) IP;
489     int i;
490
491     APInterpreterClean (IP);
492
493     if (msgEndian != pIP->sysEndian) {
494         APendianConversation32Bit((uint32_t *)&
instructionsNumber);
495         APendianConversation32Bit((uint32_t *)&
VariableNumber);
496     }
497
498     pIP->code = (TAPInterpreterFuncCall *) malloc(
sizeof(TAPInterpreterFuncCall)*
instructionsNumber);
499     pIP->instructionCount = instructionsNumber;
500
501     pIP->variables = (TAPInterpreterVariable *)
malloc(sizeof(TAPInterpreterVariable) * (
VariableNumber));
502     for (i = 0; i < VariableNumber; i++) {
503         pIP->variables[i].pData = NULL;
504         pIP->variables[i].pVI = NULL;
505     }
506     pIP->variableCount = VariableNumber;
507
508     return 0;
509 }
510
511 // load a variable/~array to an index
512 int APInterpreterLoadVar (TAPInterpreter IP, int
msgEndian, int32_t index, int32_t varTypeID, int32_t
numberOfElements)
513 {
514     TAPrealInterpreter * pIP = (TAPrealInterpreter
*) IP;
515     if (msgEndian != pIP->sysEndian) {
516         APendianConversation32Bit((uint32_t *)&
index);
517         APendianConversation32Bit((uint32_t *)&
varTypeID);
518     }
519
520     if ((index < 0) || (index > pIP->variableCount))
return -1;
521
522     // set pointer to the runtime variable
523     TAPInterpreterVariable * pRTV = pIP->variables +
index;

```

```

524     THAL_Variable const * pV = HALfindVar(varTypeID)
525     ;
526     if (!pV) return -2;
527     pRTV->pData = pV->pFkt_create((unsigned int)
528         numberOfElements);
529     //if (!pRTV->pData) return -3;
530     pRTV->pVI = pV;
531     return 0;
532 }
533
534 // load a single Instruction to an index
535 int APInterpreterLoadInstr (TAPInterpreter IP,int
536     msgEndian, int32_t index, int32_t * pRawInstr)
537 {
538     TAPrealInterpreter * pIP = (TAPrealInterpreter
539         *) IP;
540     if (msgEndian != pIP->sysEndian) {
541         APendianConversation32Bit((uint32_t *)&
542             index);
543     }
544     if ((index < 0) || (index > pIP->
545         instructionCount)) return -1;
546     TAPInterpreterFuncCall * pIFC = pIP->code +
547         index;
548     memset (pIFC, 0, sizeof(TAPInterpreterFuncCall))
549     ;
550     // get function
551     int32_t fid = *pRawInstr;
552     if (msgEndian != pIP->sysEndian) {
553         APendianConversation32Bit((uint32_t *)&
554             fid);
555     }
556     THALFunction const * pF = HALfindFunction(fid);
557     if (!pF) return -2;
558     pIFC->pHALFkt = pF->pFktHAL;
559     // convert parameters
560     pRawInstr++; // set to the first parameter
561     int i;
562     THALFunctionParam const * pP = pF->paramList.pL;
563     TuAPIInterpreterFunctionParameter * pIFP = pIFC->
564         param;
565     for (i = 0; i < pF->paramList.number; i++) {
566         if (APconvertRawParamData (msgEndian,pIP
567             ->sysEndian,pRawInstr,pP,pIFP,pIP->
568                 variables)) return -3;
569         pP++;
570         pRawInstr++;
571         pIFP++;
572     }
573     return 0;
574 }

```

```

567
568 typedef struct SAPrealMsgSystem {
569     TAPMsg *      pOldRXMsg;
570                 // pointer to the oldest received
571     TAPMsg *      pNewRXMsg;
572                 // pointer to the newest received
573     TAPMMU        mmu;
574                 // the mmu
575
576     int           sysEndianness; // the system endianness
577     int           messagecounter; // a counter for checkin if a
578                                     new message has been received
579
580 } TAPrealMsgSystem;
581
582 int SMinitial (
583     void *      pVoidSM,
584                 // pointer to the
585     uint32_t *  pD,
586                 // pointer to the
587     data        number
588                 // the number of data
589     elements
590 );
591
592 int SMdata (
593     void *      pVoidSM,
594                 // pointer to the
595     uint32_t *  pD,
596                 // pointer to the
597     data        number
598                 // the number of data
599     elements
600 );
601
602 int SMmessageFinished (
603     void *      pVoidSM
604                 // pointer to the statemachine
605 );
606
607 // create AP message system
608 TAPMsgSystem APMScreate (
609     TAPMMU        mmu,
610                 // the mmu

```

```

600         int
           sysEndianness    // the system
           endianness
601     ) {
602         TAPrealMsgSystem * pMS = (TAPrealMsgSystem *)
           malloc (sizeof(TAPrealMsgSystem));
603         if (!pMS) return NULL;
604         pMS->mmu = mmu;
605         pMS->sysEndianness = sysEndianness;
606         pMS->pOldRXMsg = NULL;
607         pMS->pNewRXMsg = NULL;
608         pMS->messagecounter = 0;
609
610         return pMS;
611     }
612 }
613
614 void APMSdelete (
615     TAPMsgSystem ms
616 ) {
617     TAPrealMsgSystem * pMS = (TAPrealMsgSystem *) ms
           ;
618     free (pMS);
619 }
620
621 // frees a message from the message system
622 void APMSdeleteMsg (
623     TAPMsgSystem ms,
624     TAPMsg * pM
625 ) {
626     TAPrealMsgSystem * pMS = (TAPrealMsgSystem *) ms
           ;
627     AP_MMU_free(pMS->mmu, pM->memory);
628 }
629
630 // get memory for a new message
631 TAPMsg * APMSgetNewMsg (
632     TAPrealMsgSystem * pMsgSys,
633     int
           dataElementsNumber,
634     const TAPMsgDrv * pDrv
635 ) {
636     TAPMMUmemory m = AP_MMU_get (pMsgSys->mmu,
           sizeof(TAPMsg)/sizeof(int) +
           dataElementsNumber*sizeof(uint32_t)/sizeof(
           int));
637     if (!m) return NULL;
638
639     int * pRD = (int *) AP_MMU_getData(m);
640     TAPMsg * pM = (TAPMsg *) pRD;
641     pM->memory = m;
642     pM->extraData.pDrv = pDrv;
643     pM->pH = (TAPMsgHeader *) ((int *) pRD + sizeof(
           TAPMsg)/sizeof(int));
644 }

```

```

645     pM->pData = (int *)pM->pH + sizeof(TAPMsgHeader)
        /sizeof(int);
646     pM->pNext = NULL;
647     return pM;
648 }
649
650 // insert a new message into the message queue
651 void APMSInsertMsg (
652     TAPrealMsgSystem *      pMS,
653     TAPMsg *                pM
654 ) {
655
656     if (pMS->pNewRXMsg) {
657         pMS->pNewRXMsg->pNext = pM;
658     }
659     pMS->pNewRXMsg = pM;
660     if (!pMS->pOldRXMsg) {
661         pMS->pOldRXMsg = pM;
662     }
663     pMS->messagecounter++;
664
665 }
666
667 // get oldest message
668 TAPMsg * APMSgetMsg (
669     TAPMsgSystem            ms,
670     // the message system
671     TAPMessageID            msgID,
672     // if 0 all messages are allowed
673     TAPNodeID              sender,
674     // if 0 all senders are
675     allowed
676     uint32_t                mNumber
677     // if 0 all numbers are
678     allowed
679 ) {
680
681     TAPrealMsgSystem * pMS = (TAPrealMsgSystem *) ms
        ;
682
683     TAPMsg * res = NULL;
684     // search msg list
685
686     TAPMsg * pM = pMS->pOldRXMsg;
687     TAPMsg * pAntecessorM = NULL;
688     // flags
689     int senderOK;
690     int msgIDok;
691     int numberOK;
692
693     while (pM) {
694         senderOK = 0;
695         msgIDok = 0;

```



```

692         numberOK = 0;
693
694         if (!sender) {
695             senderOK = 1;
696         } else {
697             if (*(pM->pH)[
698                 eAPMsgHeaderPosition_sender]
699                 == sender) senderOK = 1;
700
701             if (!msgID) {
702                 msgIDok = 1;
703             } else {
704                 if (*(pM->pH)[
705                     eAPMsgHeaderPosition_msgTypeID
706                     ] == msgID) msgIDok = 1;
707
708                 if (!mNumber) {
709                     numberOK = 1;
710                 } else {
711                     if (*(pM->pH)[
712                         eAPMsgHeaderPosition_msgNumber
713                         ] == mNumber) numberOK = 1;
714
715                     if ((senderOK) && (msgIDok) && (numberOK
716                         )) {
717                         res = pM;
718                         goto exit;
719                     }
720                     // the id was wrong but the number was
721                     // ok >> so the action failed
722                     if (numberOK) goto fail;
723                     pAntecessorM = pM;
724                     pM = pM->pNext;
725                 }
726             }
727         }
728     fail:
729         return NULL;
730     exit:
731         if (pAntecessorM) {
732             pAntecessorM->pNext = pM->pNext;
733         } else {
734             pMS->pOldRXMsg = pM->pNext;
735         }
736         if (pM == pMS->pNewRXMsg) {
737             pMS->pNewRXMsg = NULL;
738         }
739
740         return res;
741 }
742
743 // wait till a new message has been received
744 void APMSwaitForNewMessage (TAPrealMsgSystem * pMS)
745 {
746     volatile int mc;
747
748

```

```

739         mc = pMS->messagecounter;
740
741         while (mc == pMS->messagecounter) {
742
743         }
744     }
745
746     // returns 0 if a new message is available
747     inline int APMSisMessageAvailble (TAPrealMsgSystem * pMS
748         ) {
749         return (!pMS->pOldRXMsg) ? 0 : -1;
750     }
751
752     // =====
753     // the receive state machine
754     // =====
755     // the receive state machine state function for
756     // receiving the msg header
757     int SMinitial (
758         void *                pVoidSM,
759         // pointer to the
760         statemachine
761         uint32_t *            pD,
762         // pointer to the
763         data
764         int                    number
765         // the number of data
766         elements
767     ) {
768         TAPReceiveStateMachine *
769         pSM = (TAPReceiveStateMachine *) pVoidSM;
770         TAPrealMsgSystem *
771         pMS = (TAPrealMsgSystem *) pSM->pMS;
772
773         int
774         copyAmount = number;
775
776         int
777         i;
778
779         // 1. try to copy the data to the header
780         if (pSM->elementsLeft < copyAmount) copyAmount =
781             pSM->elementsLeft;
782         // copy
783         for (i = 0; i < copyAmount; i++) {
784             *pSM->pD = *pD;
785             pSM->pD++;
786             pD++;
787         }
788         pSM->elementsLeft -= copyAmount;
789
790         // check if we have to change the statemachine
791         // because we received the header
792         if (pSM->elementsLeft) return 0;
793
794         // yes! alloc msg buffer and (opt.) transfer

```

```

data
780
781 // 2. convert endian
782 int msgEndian = pSM->header[
    eAPMsgHeaderPosition_endian];
783 if (pMS->sysEndianness != msgEndian) {
784     for (i = 1; i <
        eAPMsgHeaderPosition_headerElementNumber
        ;i++) {
785         ApendianConversation32Bit(&pSM->
            header[i]);
786     }
787 }
788 // 3. now alloc message
789 // 3.1 get length
790 int msgElementNumber = (int) pSM->header[
    eAPMsgHeaderPosition_length];
791 // 3.2. get memory
792 pSM->pMsg = APMSgetNewMsg (pMS,msgElementNumber,
    pSM->pDrv);
793 if (!pSM->pMsg) return -100;
794
795 // 3.3 check getMemory result
796 if (!pSM->pMsg) return -1;
797 // copy message header
798 pSM->pD = (uint32_t *) pSM->pMsg->pH;
799 for (i = 0; i <
    eAPMsgHeaderPosition_headerElementNumber;i++)
    {
800         *pSM->pD = pSM->header[i];
801         pSM->pD++;
802     }
803 pSM->elementsLeft = pSM->header[
    eAPMsgHeaderPosition_length];
804 // set up the data
805 // 1. check if there is an data element
806 if (!pSM->elementsLeft) {
807     // no! now finish the message
808     return SMmessageFinished(pVoidSM);
809 }
810 // 2. yes
811 // 2.1 setup the sm for the data receiving
812 pSM->state = SMdata;
813 // 2.2 now check if we have to copy some data
814 number -= copyAmount;
815 if (number) {
816     // set the data pointer
817     pD += copyAmount;
818     // and copy the data
819     return SMdata (pVoidSM,pD,number);
820 }
821 return 0;
822 }
823
824

```

```

825 // the receive state machine state function for
      receiving the data
826 int SMdata (
827     void *                pVoidSM,
                        // pointer to the
                        statemachine
828     uint32_t *            pD,
                        // pointer to the
                        data
829     int                    number
                        // the number of data
                        elements
    ) {
830     TAPReceiveStateMachine *
831     pSM = (TAPReceiveStateMachine *) pVoidSM;
832     int
833         copyAmount = number;
834         i;
835     // 1. transfer the data
836     // do some clipping
837     if (pSM->elementsLeft < copyAmount) copyAmount =
838         pSM->elementsLeft;
839     // copy
840     for (i = 0; i < copyAmount; i++) {
841         *pSM->pD = *pD;
842         pSM->pD++;
843         pD++;
844     }
845     // set statemachine work data
846     pSM->elementsLeft -= copyAmount;
847     // check if we have to change the statemachine
848     if (pSM->elementsLeft) return 0;
849     int res = SMmessageFinished (pVoidSM);
850     if (res) return res;
851     // check if there some bytes left to copy
852     number -= copyAmount;
853     if (number) {
854         // set the data pointer
855         pD += copyAmount;
856         // and copy the data
857         return pSM->state (pVoidSM, pD, number);
858     }
859     return 0;
860 }
861
862 // this function is called when all data have been
      received
863 int SMmessageFinished (
864     void *                pVoidSM
                        // pointer to the statemachine
865     ) {
866     TAPReceiveStateMachine *

```

```

867     pSM = (TAPReceiveStateMachine *) pVoidSM;
      TAPMsg *
868         pM;
869     // 1. reset SM
870     // set the helper
      pSM->elementsLeft =
871         eAPMsgHeaderPosition_headerElementNumber;
872     pSM->pD = pSM->header;
873     // data
874     pM = pSM->pMsg; // save msg info for inserting
875     pSM->pMsg = NULL;
876     // right state function
877     pSM->state = SInitial; // the state
878     // 2. insert message at the message system
879     APMSInsertMsg ((TAPrealMsgSystem *)pSM->pMS,pM);
880     return 0;
881 }
882
883 // inits the state machine
884 void APInitReceiveStateMachine (
885     TAPReceiveStateMachine *
886     pSM, // pointer to the
887     state machine
888     TAPMsgSystem
889     pMS, // pointer to
890     the message system
891     const TAPMsgDrv *
892     pDrv // the driver
893     associated with the statemachine
894 ) {
895     pSM->state = SInitial;
896     pSM->pMS = pMS;
897     pSM->pDrv = pDrv;
898     // set the helper
899     pSM->elementsLeft =
900         eAPMsgHeaderPosition_headerElementNumber;
901     pSM->pD = pSM->header;
902     // data
903     pSM->pMsg = NULL;
904 }
905
906 int APHandleMsg (
907     TAP *
908     TAPMsg *
909     pAP,
910     pM
911     TAPMessageID
912     msgID;
913     const THALMsgProcessMessageAssociation *
914     pMsgIDandFuncAsso;

```

```

910         int
911                                     i;
912
913         // get message id
914         msgID = (*(pM->pH))[
915             eAPMsgHeaderPosition_msgTypeID];
916         // search handler
917         pMsgIDandFunctAsso = gHALMsgProcessRXHandlers.pL
918             ;
919         for (i = 0; i < gHALMsgProcessRXHandlers.number;
920             i++) {
921             if (((TAPMessageID)pMsgIDandFunctAsso->
922                 msgID) == msgID) {
923                 return pMsgIDandFunctAsso->
924                     pfktHandle(pAP,pM);
925             }
926             pMsgIDandFunctAsso++;
927         }
928         return -1;
929     }
930 }
931
932 void APMessageProcessingThread (TAP * pAP) {
933
934     TAPrealMsgSystem *          pMS = (
935         TAPrealMsgSystem *) pAP->MS;
936     TAPMsg *                    pM;
937     TAPNodeID                   recv;
938     while (1) {
939         // wait for a message
940         APMSwaitForNewMessage(pMS);
941         // get the message
942         pM = APMSgetMsg (pMS,0,0,0);
943         // search the message handler
944         recv = (*(pM->pH))[
945             eAPMsgHeaderPosition_receiver];
946         if ((recv == dAPNodeID_ALL) || (recv ==
947             pAP->nodeID)) {
948             if(APHandleMsg (pAP,pM)) goto
949                 exit;
950         }
951         // free memory
952         AP_MMU_free(pMS->mmu, pM->memory);
953     }
954 exit:
955     AP_MMU_free(pMS->mmu, pM->memory);
956 }

```

3.5.5 audio processor blueprint 5 (MSP430-169STK)

Informations:

description:	a audio processor for the MSP430
--------------	----------------------------------

includes:

c-Include	c-Library	system lib
AP.h		no

code:

```

1 // =====
2 // AP uuid = 11
3 // =====
4
5 // inits the AP
6 int APinit (
7
8             TAP *
9             pAP,
10            TAPNodeID
11            nodeID,
12            const TAPMsgDrv *      pDrvList
13            ,
14            const int
15            driverNumber,
16            size_t
17            messagePoolSize,
18            int
19            sysEndian
20
21        )
22 {
23     gAPendianFlag = sysEndian;
24
25     pAP->nodeID = nodeID;
26     pAP->pNodeList = NULL;
27     pAP->pDrvList = pDrvList;
28     pAP->driverNumber = driverNumber;
29     pAP->msgSysMMU = AP_MMU_create(messagePoolSize);
30     pAP->IP = APInterpreterCreate(pAP);
31     pAP->MS = APMScreate (pAP->msgSysMMU,sysEndian);
32     pAP->msgNumber = 0;
33     pAP->APstate = eAPstate_idle;
34
35     if(
36         (!pAP->msgSysMMU) ||
37         (!pAP->IP) ||
38         (!pAP->MS)
39     ) return -1;
40
41     msp430_initHW(10);
42
43     // init drv
44     TAPMsgDrv * pDrv = (TAPMsgDrv *)pDrvList;
45     int i;
46     for (i = 0; i < driverNumber; i++) {
47         pDrv->pfkt_open(pAP, pDrv);
48         pDrv++;
49     }
50
51     // login the ap to the message system

```

```

44         return TX_login(pAP);
45     }
46
47     // deletes the AP
48     void APdelete (TAP * pAP)
49     {
50         TX_logout(pAP);
51         // close & destroy drv
52         TAPMsgDrv * pDrv = (TAPMsgDrv *)pAP->pDrvList;
53         int i;
54         for (i = 0; i < pAP->driverNumber; i++) {
55             pDrv->pfkt_close(pDrv);
56             pDrv->pfkt_destroy(pDrv);
57             pDrv++;
58         }
59         APMSdelete (pAP->MS);
60         APInterpreterDelete(pAP->IP);
61         AP_MMU_delete(pAP->msgSysMMU);
62     }
63
64     // find a node at the list
65     TAPNode * APfindNode(TAP * pAP, TAPNodeID nodeID) {
66         TAPNode * pN = pAP->pNodeList;
67         while (pN) {
68             if (pN->nodeID == nodeID) return pN;
69             pN = pN->pNext;
70         };
71         return NULL;
72     }
73
74     // adds a new node to the node list
75     int APaddNode(TAP * pAP, TAPNodeID newNodeID, const
76         TAPMsgDrv * pDrv) {
77         if (APfindNode(pAP, newNodeID)) return 1;
78         TAPNode * pN = (TAPNode *) malloc(sizeof(TAPNode
79             ));
80         if (!pN) return -1;
81         pN->nodeID = newNodeID;
82         pN->pDrv = pDrv;
83         pN->pNext = pAP->pNodeList;
84         pAP->pNodeList = pN;
85         return 0;
86     }
87
88     // removes a node from the node list
89     void APremoveNode(TAP * pAP, TAPNodeID nodeID){
90         TAPNode * pAntN = pAP->pNodeList; // antecessor
91         TAPNode * pActN = pAP->pNodeList; // actual node
92
93         while (pActN) {
94             // compare node id's
95             if (pActN->nodeID == nodeID) {

```



```

96         // check if we at the first
97         position at the list
98         if (pAP->pNodeList == pAntN) {
99             // reset the pointer
100             pAP->pNodeList = pActN->
101                 pNext;
102         } else {
103             // set the antecessor
104             pAntN->pNext = pActN->
105                 pNext;
106         }
107         // free node
108         free(pActN);
109         // and abort
110         return;
111     }
112     // the actual element becomes the
113     precessor element
114     pAntN = pActN;
115     pActN = pActN->pNext;
116 }
117
118 // get a new message number
119 unsigned int APgetNewMessageNumber (TAP *pAP) {
120     pAP->msgNumber++;
121     return pAP->msgNumber;
122 }
123
124 // find the driver associated with der nodeID
125 const TAPMsgDrv * APfindDrvBySenderId (TAP * pAP,
126     TAPNodeID node) {
127     TAPNode * pN = pAP->pNodeList;
128     while (pN) {
129         if (pN->nodeID == node) {
130             return pN->pDrv;
131         }
132         pN = pN->pNext;
133     }
134     return NULL;
135 }
136
137 // runs the AP
138 int APrun(TAP *pAP) {
139     pAP->APstate = eAPstate_run;
140     return 0;
141 }
142
143 // =====
144 // MMU functions
145 // =====
146
147 typedef struct SAPrealMMUMemory {
148     uint32_t *
149         pData; // the data

```

```

145         int                count;                // amount of
            data elements
146         struct SAPrealMMUMemory *                pNext;
            // next element
147         struct SAPrealMMUMemory *                pPrev;
            // previous element
148     } TAPrealMMUMemory;
149
150     //the mmu type
151     typedef struct SAPrealMMU {
152         uint32_t *                memory;
            // the memory block
153         TAPrealMMUMemory *                pStart;
            // first element
154         TAPrealMMUMemory *                pEnd;
            // second element
155         TAPrealMMUMemory *                pUnusedList;
            // list with the unused elements
156         uint32_t *                pUnusedData;
            // pointer to the unused memory
157         int
            elementsAvailable;                // amount of elements
            witch are available without using the
            garbage collector
158         int
            totalAvailable;                // total amount of
            free bytes
159     } TAPrealMMU;
160
161     // =====
162     // memory entry functions
163     // =====
164
165     // a little macro for unchaining an element
166     #define DMemoryEntryUnchain(pM) \
167         if (pM->pNext) pM->pNext->pPrev = pM->pPrev; \
168         if (pM->pPrev) pM->pPrev->pNext = pM->pNext
169
170
171     //creates an memory entry
172     TAPrealMMUMemory * MemoryEntry_create () {
173         TAPrealMMUMemory * pM = NULL;
174
175         pM = (TAPrealMMUMemory *) malloc(sizeof(
176             TAPrealMMUMemory));
177         if (!pM) return NULL;
178         pM->pData = NULL;
179         pM->count = 0;
180         pM->pNext = NULL;
181         pM->pPrev = NULL;
182
183         return pM;
184     }

```

```

185
186 //deletes an memory Entry
187 void MemoryEntry_delete (
188     TAPrealMMUMemory * pM    // the memory to
        delete
189 )
190 {
191     // put the entry out of the chain
192     DMemoryEntryUnchain(pM);
193     // now we delete it
194     free(pM);
195 }
196
197 // =====
198 // mmu helper
199 // =====
200
201 //alloc if needed a new memory entry
202 TAPrealMMUMemory * MMU_helper_createMemoryEntry (
203     TAPrealMMU *    pMMU    // MMU
        structure to init
204 )
205 {
206     // check if we have to alloc a new memory entry
207     if (!pMMU->pUnusedList) return
        MemoryEntry_create();
208     // no there is some left at the list
209     TAPrealMMUMemory * pM;
210     // take the first one
211     pM = pMMU->pUnusedList;
212     // reset the list
213     pMMU->pUnusedList = pM->pNext;
214     // now unchain the element (for sure)
215     DMemoryEntryUnchain(pM);
216     // set the element pointers
217     pM->pNext = NULL;
218     pM->pPrev = NULL;
219     return pM;
220 }
221
222 //the garbage collector
223 void MMU_helper_garbageCollector (
224     TAPrealMMU *    pMMU    // MMU
        structure to init
225 )
226 {
227     TAPrealMMUMemory * pM = pMMU->pStart;
228     uint32_t * pD = pMMU->memory;
229     while (pM) {
230         // check if we have to move the data
231         if (pD != pM->pData) {
232             // move the data
233             memmove(pD, pM->pData, pM->count*
                sizeof(uint32_t));
234         }

```

```

235         // reset the destination pointer
236         pD += pM->count;
237         pM = pM->pNext;
238     }
239     // compressing memory finished
240     // set the mmu vars new
241     pMMU->elementsAvailable = pMMU->totalAvailable;
242     pMMU->pUnusedData = pD;
243 }
244
245 // create a mmu
246 TAPMMU AP_MMU_create (size_t elementsNumber) {
247     TAPrealMMU * pMMU;
248
249     pMMU = (TAPrealMMU *) malloc (sizeof(TAPrealMMU)
250 );
251     if (!pMMU) return NULL;
252
253     pMMU->memory = (uint32_t *) malloc (
254         elementsNumber*sizeof(uint32_t));
255     pMMU->pUnusedData = pMMU->memory;
256
257     // setup lists
258     pMMU->pStart = NULL;
259     pMMU->pEnd = NULL;
260     pMMU->pUnusedList = NULL;
261
262     pMMU->elementsAvailable =elementsNumber;
263
264     pMMU->totalAvailable = elementsNumber;
265     return pMMU;
266 }
267
268 // destroying the mmu
269 void AP_MMU_delete (TAPMMU mmu) {
270     TAPrealMMU * pMMU = (TAPrealMMU *) mmu;
271
272     TAPrealMMUMemory * pM;
273     TAPrealMMUMemory * pMnext;
274
275     // 1. delete al mmu entry's
276     // 1.1 unused entry
277     pM = pMMU->pUnusedList;
278     while (pM) {
279         pMnext = pM->pNext;
280         MemoryEntry_delete(pM);
281         pM = pMnext;
282     }
283     pMMU->pUnusedList = NULL;
284     // 1.2 used blocks
285     pM = pMMU->pStart;
286     while (pM) {
287         pMnext = pM->pNext;
288         MemoryEntry_delete(pM);

```

```

288         pM = pMnext;
289     }
290     pMMU->pStart = NULL;
291     pMMU->pEnd = NULL;
292     // 2. delete mmu memory
293     free (pMMU->memory);
294 }
295
296 // getting memmory from the mmu
297 TAPMMUmemory AP_MMU_get (TAPMMU mmu, size_t elements) {
298     TAPrealMMU * pMMU = (TAPrealMMU *) mmu;
299
300     // check if there is enough space
301     if (pMMU->totalAvailable < elements) goto error;
302     // check if we have to use the garbage collector
303     if (pMMU->elementsAvailable < elements) {
304         // start garbage collector
305         MMU_helper_garbageCollector(pMMU);
306     }
307     // we have enough memory so let's allocate some
308
309     // get a new entry
310     TAPrealMMUMemory * pM;
311     pM = MMU_helper_createMemoryEntry(pMMU);
312     if (!pM) return NULL;
313     // get some memory
314     pM->pData = pMMU->pUnusedData;
315     pM->count = elements;
316     // refresh data
317     pMMU->pUnusedData += elements;
318     pMMU->totalAvailable -= elements;
319     pMMU->elementsAvailable -= elements;
320     // insert memory element at the end of the list
321     // and update last element
322     pM->pPrev = pMMU->pEnd;
323     if (pMMU->pEnd) pMMU->pEnd->pNext = pM;
324     if (!pMMU->pStart) pMMU->pStart = pM;
325     pMMU->pEnd = pM;
326     return pM;
327 error:
328     return NULL;
329 }
330
331 // free memmory from the mmu
332 void AP_MMU_free (TAPMMU mmu, TAPMMUmemory memory) {
333     TAPrealMMU * pMMU = (TAPrealMMU *) mmu;
334     TAPrealMMUMemory * pM = (TAPrealMMUMemory *)
335         memory;
336
337     if (!pM) return;
338     // set mmu settings
339     if (pMMU->pStart == pM) pMMU->pStart = pM->pNext;
340     if (pMMU->pEnd == pM) pMMU->pEnd = pM->pPrev;
341     // unchain element

```

```

340     DMemoryEntryUnchain (pM);
341     // and put it to the chain of unused
342     pM->pNext = pMMU->pUnusedList;
343     pM->pPrev = NULL;
344     if (pMMU->pUnusedList) {
345         pMMU->pUnusedList->pPrev = pM;
346     }
347     pMMU->pUnusedList = pM;
348     // now set the mmu data new
349     pMMU->totalAvailable += pM->count;
350 }
351
352 // getting access to the MMU data
353 void * AP_MMU_getData (TAPMMUmemory memory) {
354     TAPrealMMUMemory * pM = (TAPrealMMUMemory *)
355         memory;
356     return pM->pData;
357 }
358 // the real interpreter
359 typedef struct SAPrealInterpreter {
360     TAP *
361         pAP; //
362         // pointer to the audio processor
363         int
364             state; // state of the IP
365         int
366             nextState; // the next state of the IP
367     TAPInterpreterCPU cpu; // the IP
368     core
369     TAPInterpreterFuncCall * code; // the code
370     int32_t
371         instructionCount; // number of
372         instructions at the code
373     TAPInterpreterVariable * variables; // the variables
374     int32_t
375         variableCount; // number of
376         the variables
377 } TAPrealInterpreter;
378
379 // create a new interpreter
380 TAPInterpreter APIInterpreterCreate (void * pAP) {
381     TAPrealInterpreter * pIP = NULL;
382     pIP = (TAPrealInterpreter *) malloc (sizeof(
383         TAPrealInterpreter));
384     if (!pIP) return NULL;
385
386     pIP->pAP = pAP;
387     pIP->state = eAPIInterpreterState_idle;
388     pIP->nextState = eAPIInterpreterState_idle;

```

```

379     pIP->cpu.IP = pIP;
380     pIP->cpu.CF = 0;
381     pIP->cpu.EF = 0;
382     pIP->cpu.pCodeStart = NULL;
383     pIP->cpu.pCodeEnd = NULL;
384     pIP->cpu.pIP = NULL;
385
386     pIP->code = NULL;
387     pIP->instructionCount = 0;
388
389     pIP->variables = NULL;
390     pIP->variableCount = 0;
391
392     return pIP;
393 }
394
395 // cleans the interpreter
396 void APIInterpreterClean (TAPIInterpreter IP) {
397     TAPrealInterpreter * pIP = (TAPrealInterpreter
398         *) IP;
399
400     // clean code
401     if (pIP->code) {
402         free (pIP->code);
403         pIP->code = NULL;
404     }
405     pIP->instructionCount = 0;
406
407     // clean variables
408     TAPIInterpreterVariable * pV = pIP->variables;
409     int i;
410     for (i = 0; i < pIP->variableCount; i++) {
411         if (pV->pVI) pV->pVI->pFkt_delete(pV->
412             pData);
413         pV++;
414     }
415     if (pIP->variables) {
416         free (pIP->variables);
417         pIP->variables = NULL;
418     }
419     pIP->variableCount = 0;
420 }
421
422 // deletes the interpreter
423 void APIInterpreterDelete (TAPIInterpreter IP) {
424     TAPrealInterpreter * pIP = (TAPrealInterpreter
425         *) IP;
426     APIInterpreterClean(IP);
427     free (pIP);
428 }
429
430 extern void drv_3_feedRecvStateM (void);
431
432 int APIInterpreterStateRun(TAPIInterpreter IP) {

```

```

431     TAPrealInterpreter * pIP = (TAPrealInterpreter
        *) IP;
432     TAPInterpreterFuncCall * pFC;
433     int cc; // cycle counter
434
435     // setup cpu
436     pIP->cpu.CF = 0;
437     pIP->cpu.EF = 0;
438     pIP->cpu.pIP = pIP->code;
439     pIP->cpu.pCodeStart = pIP->code;
440     pIP->cpu.pCodeEnd = pIP->code + pIP->
        instructionCount;
441
442     // run code
443     cc = 10;
444     while (eAPIInterpreterState_run == pIP->state) {
445         pFC = pIP->cpu.pIP;
446         // check if we reached the end of the
            code
447         if (pFC > pIP->cpu.pCodeEnd) {
448             return 0;
449         }
450         // execute command
451         pFC->pHALFkt (&(pIP->cpu), pFC->param);
452         // check error flags
453         if (pIP->cpu.EF) {
454             return -1;
455         }
456         // process msg receiving
457         drv_3_feedRecvStateM ();
458
459         // check message system after x cycles
460         if (!cc) {
461             APMessageProcess(pIP->pAP);
462             cc = 100;
463         } else {
464             cc--;
465         }
466     }
467     return 1;
468 }
469
470 // process the actual state
471 int APIInterpreterProcessState(TAPInterpreter IP){
472     TAPrealInterpreter * pIP = (TAPrealInterpreter
        *) IP;
473     pIP->state = pIP->nextState;
474     int rc = 0;
475
476     switch (pIP->state) {
477         case eAPIInterpreterState_idle:
478             break;
479         case eAPIInterpreterState_loadProgramm:
480             break;
481     }

```



```

482         case eAPIInterpreterState_run:
483             rc = APIInterpreterStateRun(IP);
484             if (rc >= 0) pIP->state =
485                 eAPIInterpreterState_idle;
486             break;
487         case eAPIInterpreterState_oneStep:
488             break;
489         case eAPIInterpreterState_halt:
490             break;
491         default:
492             return -10;
493     }
494     return rc;
495 }
496
497 // set interpreter state
498 int APIInterpreterSetState (TAPIInterpreter IP, int
499     msgEndian, int32_t state) {
500     TAPrealInterpreter * pIP = (TAPrealInterpreter
501         *) IP;
502     if (msgEndian != gAPendianFlag) {
503         APendianConversation32Bit((uint32_t *)&
504             state, msgEndian);
505     }
506     pIP->nextState = (int) state;
507     return 0;
508 }
509
510 // setup the interpreter for a new program
511 int APIInterpreterInitNewProgramm (TAPIInterpreter IP, int
512     msgEndian, int32_t instructionsNumber, int32_t
513     VariableNumber) {
514     TAPrealInterpreter * pIP = (TAPrealInterpreter
515         *) IP;
516     int i;
517
518     APIInterpreterClean (IP);
519
520     if (msgEndian != gAPendianFlag) {
521         APendianConversation32Bit((uint32_t *)&
522             instructionsNumber, msgEndian);
523         APendianConversation32Bit((uint32_t *)&
524             VariableNumber, msgEndian);
525     }
526
527     pIP->code = (TAPIInterpreterFuncCall *) malloc(
528         sizeof(TAPIInterpreterFuncCall)*
529         instructionsNumber);
530     pIP->instructionCount = instructionsNumber;
531
532     pIP->variables = (TAPIInterpreterVariable *)
533         malloc(sizeof(TAPIInterpreterVariable) * (
534             VariableNumber));
535     for (i = 0; i < VariableNumber; i++) {

```

```

524         pIP->variables[i].pData = NULL;
525         pIP->variables[i].pVI = NULL;
526     }
527     pIP->variableCount = VariableNumber;
528
529     return 0;
530 }
531
532 // load a variable/~array to an index
533 int APInterpreterLoadVar (TAPInterpreter IP, int
    msgEndian, int32_t index, int32_t varTypeID, int32_t
    numberOfElements)
534 {
535     TAPrealInterpreter * pIP = (TAPrealInterpreter
        *) IP;
536     if (msgEndian != gAPendianFlag) {
537         APendianConversation32Bit((uint32_t *)&
            index, msgEndian);
538         APendianConversation32Bit((uint32_t *)&
            varTypeID, msgEndian);
539         APendianConversation32Bit((uint32_t *)&
            numberOfElements, msgEndian);
540     }
541
542     if ((index < 0) || (index > pIP->variableCount))
543         return -1;
544
545     // set pointer to the runtime variable
546     TAPInterpreterVariable * pRTV = pIP->variables +
        index;
547     THAL_Variable const * pV = HALfindVar(varTypeID)
548     ;
549     if (!pV) return -2;
550
551     pRTV->pData = pV->pFkt_create((unsigned int)
        numberOfElements);
552     //if (!pRTV->pData) return -3;
553
554     pRTV->pVI = pV;
555     return 0;
556 }
557
558 // load a single Instruction to an index
559 int APInterpreterLoadInstr (TAPInterpreter IP, int
    msgEndian, int32_t index, int32_t * pRawInstr)
560 {
561     TAPrealInterpreter * pIP = (TAPrealInterpreter
        *) IP;
562     if (msgEndian != gAPendianFlag) {
563         APendianConversation32Bit((uint32_t *)&
            index, msgEndian);
564     }
565     if ((index < 0) || (index > pIP->
        instructionCount)) return -1;
566     TAPInterpreterFuncCall * pIFC = pIP->code +

```

```

565         index;
566         memset (pIFC, 0, sizeof(TAPInterpreterFuncCall))
567         ;
568         // get function
569         int32_t fid = *pRawInstr;
570         if (msgEndian != gAPendianFlag) {
571             APendianConversation32Bit((uint32_t *)&
572                                     fid, msgEndian);
573         }
574         THALFunction const * pF = HALfindFunction(fid);
575         if (!pF) return -2;
576         pIFC->pHALFkt = pF->pfktHAL;
577         // convert parameters
578         pRawInstr++; // set to the first parameter
579         int i;
580         THALFunctionParam const * pP = pF->paramList.pL;
581         TuAPInterpreterFunctionParameter * pIFP = pIFC->
582             param;
583         for (i = 0; i < pF->paramList.number; i++) {
584             if (APconvertRawParamData (msgEndian,
585                                         pRawInstr, pP, pIFP, pIFP->variables))
586                 return -3;
587             pP++;
588             pRawInstr++;
589             pIFP++;
590         }
591         return 0;
592     }
593 }
594 // gets the variable by it's index
595 TAPInterpreterVariable * APInterpreterGetVariableByIndex
596 (TAPInterpreter IP, int index) {
597     return &(((TAPrealInterpreter *) IP)->variables[
598         index]);
599 }
600 // gets the AP from the IP
601 void * APInterpreterGetAPfromIP (TAPInterpreter IP) {
602     return ((TAPrealInterpreter *) IP)->pAP;
603 }
604
605 typedef struct SAPrealMsgSystem {
606     TAPMsg * pOldRXMsg;
607         // pointer to the oldest received
608     messages
609     TAPMsg * pNewRXMsg;
610         // pointer to the newest received
611     messages
612     TAPMMU mmu;
613         // the mmu
614     int
615     sysEndianness; // the system endianness

```

```

606         int
            messagecounter; // a counter for checkin if a
                           new message has been received
607
608     } TAPrealMsgSystem;
609
610
611
612     int SMinitial (
613         void *                pVoidSM,
                           // pointer to the
                           statemachine
614         uint32_t *            pD,
                           // pointer to the
                           data
615         int                    number
                           // the number of data
                           elements
        );
616
617
618     int SMdata (
619         void *                pVoidSM,
                           // pointer to the
                           statemachine
620         uint32_t *            pD,
                           // pointer to the
                           data
621         int                    number
                           // the number of data
                           elements
        );
622
623
624     int SMmessageFinished (
625         void *                pVoidSM
                           // pointer to the statemachine
        );
626
627
628
629
630     // create AP message system
631     TAPMsgSystem APMScreate (
632         TAPMMU                mmu,
                           // the mmu
633         int                    sysEndianness // the system
                           endianness
        ) {
634         TAPrealMsgSystem * pMS = (TAPrealMsgSystem *)
635             malloc (sizeof(TAPrealMsgSystem));
636         if (!pMS) return NULL;
637         pMS->mmu = mmu;
638         pMS->sysEndianness = sysEndianness;
639         pMS->pOldRXMsg = NULL;
640         pMS->pNewRXMsg = NULL;
641         pMS->messagecounter = 0;

```

```

642
643
644         return pMS;
645     }
646
647     void APMSdelete (
648         TAPMsgSystem ms
649     ) {
650         TAPrealMsgSystem * pMS = (TAPrealMsgSystem *) ms
651         ;
652         free (pMS);
653     }
654
655     // frees a message from the message system
656     void APMSdeleteMsg (
657         TAPMsgSystem      ms,
658         TAPMsg *          pM
659     ) {
660         TAPrealMsgSystem * pMS = (TAPrealMsgSystem *) ms
661         ;
662         AP_MMU_free(pMS->mmu, pM->memory);
663     }
664
665     // get memory for a new message
666     TAPMsg * APMSgetNewMsg (
667         TAPrealMsgSystem * pMsgSys,
668         int
669             dataElementsNumber,
670         const TAPMsgDrv * pDrv
671     ) {
672         // we go for sure that we get enough memory
673         // if mod(sizeof(TAPMsg)/sizeof(uint32_t)) != 0
674         // we need one uint32_t more -> +1
675         TAPMMUmemory m = AP_MMU_get (pMsgSys->mmu,
676             sizeof(TAPMsg)/sizeof(uint32_t) + 1+
677             eAPMsgHeaderPosition_headerElementNumber +
678             dataElementsNumber);
679         if (!m) return NULL;
680
681         // set the pointers
682         uint32_t * pRD = (uint32_t *) AP_MMU_getData(m);
683         TAPMsg * pM = (TAPMsg *) pRD;
684         pM->memory = m;
685         pM->extraData.pDrv = pDrv;
686         pM->pH = (TAPMsgHeader *) ((uint32_t *) pRD +
687             sizeof(TAPMsg)/sizeof(uint32_t)+1);
688         pM->pData = (uint32_t *) pM->pH + sizeof(
689             TAPMsgHeader)/sizeof(uint32_t);
690         pM->pNext = NULL;
691         return pM;
692     }
693
694     // insert a new message into the message queue
695     void APMSinsertMsg (

```

```

688         TAPrealMsgSystem *      pMS,
689         TAPMsg *                  pM
690     ) {
691
692         if (pMS->pNewRXMsg) {
693             pMS->pNewRXMsg->pNext = pM;
694         }
695         pMS->pNewRXMsg = pM;
696         if (!pMS->pOldRXMsg) {
697             pMS->pOldRXMsg = pM;
698         }
699         pMS->messagecounter++;
700     }
701 }
702
703 // unchains a received message
704 void APMSunchainMessage (
705     TAPrealMsgSystem *      pMS,
706     TAPMsg *                  pM,
707     TAPMsg *                  pAntecessorM
708 ) {
709     if (pAntecessorM) {
710         pAntecessorM->pNext = pM->pNext;
711     } else {
712         pMS->pOldRXMsg = pM->pNext;
713     }
714     if (pM == pMS->pNewRXMsg) {
715         pMS->pNewRXMsg = NULL;
716     }
717
718     // now there is one message less left
719     pMS->messagecounter--;
720 }
721
722 // get oldest message
723 TAPMsg * APMSgetMsg (
724     TAPMsgSystem              ms,
725                               // the message system
726     TAPMessageID              msgID,
727                               // if 0 all messages are
728     allowed
729     TAPNodeID                 sender,
730                               // if 0 all senders
731     are allowed
732     uint32_t                   mNumber,
733                               // if 0 all numbers
734     are allowed
735     int                        ackMsgAllowed
736 ) {
737     TAPrealMsgSystem * pMS = (TAPrealMsgSystem *) ms
738     ;

```

```

733     // flags
734     int senderOK;
735     int msgIDok;
736     int numberOK;
737     // temp vars
738     TAPMsg * pM;
739     TAPMsg * pAntecessorM;
740     uint32_t * pH;
741
742     // result var
743     TAPMsg * res = NULL;
744
745     checkMessages:
746         // search msg list
747         pM = pMS->pOldRXMsg;
748         pAntecessorM = NULL;
749
750         if (!pM) goto waitForMessage;
751
752
753         senderOK = 0;
754         msgIDok = 0;
755         numberOK = 0;
756
757         pH = *(pM->pH);
758
759         if (!sender) {
760             senderOK = 1;
761         } else {
762             if (pH[eAPMsgHeaderPosition_sender] ==
763                 sender) senderOK = 1;
764         }
765         if (!msgID) {
766             // filter ack/nack msg
767             if (ackMsgAllowed) {
768                 msgIDok = 1;
769             } else {
770                 if (
771                     (pH[
772                         eAPMsgHeaderPosition_msgTypeID
773                     ] !=
774                     eAPMsgTypes_ACK
775                     ) &&
776                     (pH[
777                         eAPMsgHeaderPosition_msgTypeID
778                     ] !=
779                     eAPMsgTypes_NACK
780                     )
781                 ) {
782                     msgIDok = 1;
783                 }
784             }
785         }
786         } else {
787             if (pH[eAPMsgHeaderPosition_msgTypeID]
788                 == msgID) msgIDok = 1;

```

```

778     }
779     if (!mNumber) {
780         numberOK = 1;
781     } else {
782         if (pH[eAPMsgHeaderPosition_msgNumber]
783             == mNumber) numberOK = 1;
784     }
785     if ((senderOK) && (msgIDok) && (numberOK)) {
786         res = pM;
787         goto exit;
788     }
789     pAntecessorM = pM;
790     pM = pM->pNext;
791     if (pM) goto checkMessages;
792 waitForMessage:
793     drv_3_feedRecvStateM();
794     goto checkMessages;
795 exit:
796     // unchain message
797     APMSunchainMessage(pMS,pM,pAntecessorM);
798     // now one thread is less waiting for a message
799     return res;
800 error:
801     return NULL;
802 }
803 // wait till a new message has been received
804 void APMSwaitForNewMessage (TAPrealMsgSystem * pMS)
805 {
806     volatile int mc;
807
808     mc = pMS->messagecounter;
809
810     while (mc == pMS->messagecounter) {
811         drv_3_feedRecvStateM();
812     }
813 }
814
815 // returns 0 if a new message is available
816 inline int APMSisMessageAvailble (TAPrealMsgSystem * pMS
817     ) {
818     return (!pMS->pOldRXMsg) ? 0 : -1;
819 }
820 // =====
821 // the receive state machine
822 // =====
823
824 // the receive state machine state function for
825 // receiving the msg header
826 int SMinitial (
827     void *                pVoidSM,
828                     // pointer to the
829                     statemachine
830     uint32_t *            pD,

```



```

// pointer to the
data
int number
// the number of data
elements
) {
TAPReceiveStateMachine *
pSM = (TAPReceiveStateMachine *) pVoidSM;
TAPrealMsgSystem *
pMS = (TAPrealMsgSystem *) pSM->pMS;
int
copyAmount = number;
int
i;

// 1. try to copy the data to the header
if (pSM->elementsLeft < copyAmount) copyAmount =
pSM->elementsLeft;
// copy
for (i = 0; i < copyAmount; i++) {
    *pSM->pD = *pD;
    pSM->pD++;
    pD++;
}
pSM->elementsLeft -= copyAmount;

// check if we have to change the statemachine
because we received the header
if (pSM->elementsLeft) return 0;

// yes! alloc msg buffer and (opt.) transfer
data

// 2. convert endian
int msgEndian = pSM->header[
eAPMsgHeaderPosition_endian];
if (pMS->sysEndianness != msgEndian) {
    for (i = 0; i <
eAPMsgHeaderPosition_headerElementNumber
; i++) {
        APendianConversation32Bit(&pSM->
header[i], gAPendianFlag);
    }
}

// 3. now alloc message
// 3.1 get length
int msgElementNumber = (int) pSM->header[
eAPMsgHeaderPosition_length];
// 3.2. get memory
pSM->pMsg = APMSgetNewMsg (pMS, msgElementNumber,
pSM->pDrv);
if (!pSM->pMsg) return -100;

// 3.3 check getMemory result
if (!pSM->pMsg) return -1;

```

```

866 // copy message header
867 pSM->pD = (uint32_t *) pSM->pMsg->pH;
868 for (i = 0; i <
    eAPMsgHeaderPosition_headerElementNumber;i++)
    {
869         *pSM->pD = pSM->header[i];
870         pSM->pD++;
871     }
872 pSM->elementsLeft = pSM->header[
    eAPMsgHeaderPosition_length];
873 // set up the data
874 // 1. check if there is an data element
875 if (!pSM->elementsLeft) {
876     // no! now finish the message
877     return SMmessageFinished(pVoidSM);
878 }
879 // 2. yes
880 // 2.1 setup the sm for the data receiving
881 pSM->state = SMdata;
882 // 2.2 now check if we have to copy some data
883 number -= copyAmount;
884 if (number) {
885     // set the data pointer
886     pD += copyAmount;
887     // and copy the data
888     return SMdata (pVoidSM,pD,number);
889 }
890 return 0;
891 }
892
893
894 // the receive state machine state function for
    receiving the data
895 int SMdata (
896     void *                pVoidSM,
897     // pointer to the
    statemachine
898     uint32_t *            pD,
899     // pointer to the
    data
900     int                    number
901     // the number of data
    elements
902 ) {
903     TAPReceiveStateMachine *
    pSM = (TAPReceiveStateMachine *) pVoidSM;
904     int
905         copyAmount = number;
906     int
907         i;
908     // 1. transfer the data
909     // do some clipping
910     if (pSM->elementsLeft < copyAmount) copyAmount =
    pSM->elementsLeft;
911     // copy

```

```

907     for (i = 0; i < copyAmount; i++) {
908         *pSM->pD = *pD;
909         pSM->pD++;
910         pD++;
911     }
912     // set statemachine work data
913     pSM->elementsLeft -= copyAmount;
914     // check if we have to change the statemachine
915     if (pSM->elementsLeft) return 0;
916     int res = SMmessageFinished (pVoidSM);
917     if (res) return res;
918
919     // check if there some bytes left to copy
920     number -= copyAmount;
921     if (number) {
922         // set the data pointer
923         pD += copyAmount;
924         // and copy the data
925         return pSM->state (pVoidSM, pD, number);
926     }
927     return 0;
928 }
929 }
930
931 // this function is called when all data have been
932 // received
933 int SMmessageFinished (
934     void *                pVoidSM
935     // pointer to the statemachine
936 ) {
937     TAPReceiveStateMachine *
938     pSM = (TAPReceiveStateMachine *) pVoidSM;
939     TAPMsg *
940     pM;
941
942     // 1. reset SM
943     // set the helper
944     pSM->elementsLeft =
945         eAPMsgHeaderPosition_headerElementNumber;
946     pSM->pD = pSM->header;
947
948     // data
949     pM = pSM->pMsg; // save msg info for inserting
950     pSM->pMsg = NULL;
951
952     // right state function
953     pSM->state = SMinitial; // the state
954
955     // 2. insert message at the message system
956     APMSInsertMsg ((TAPrealMsgSystem *)pSM->pMS, pM);
957     return 0;
958 }
959
960 // inits the state machine
961 void APInitReceiveStateMachine (

```

```

957         TAPReceiveStateMachine *
           pSM,          // pointer to the
           state machine
958         TAPMsgSystem
           pMS,          // pointer to
           the message system
959         const TAPMsgDrv *
           pDrv          // the driver
           associated with the statemachine
960     ) {
961         pSM->state = SMinitial;
962         pSM->pMS = pMS;
963         pSM->pDrv = pDrv;
964         // set the helper
965         pSM->elementsLeft =
           eAPMsgHeaderPosition_headerElementNumber;
966         pSM->pD = pSM->header;
967
968         // data
969         pSM->pMsg = NULL;
970     }
971
972     int APHandleMsg (
973         TAP *                pAP,
974         TAPMsg *             pM
975     ) {
976
977         TAPMessageID
           msgID;
978         const THALMsgProcessMessageAssociation *
           pMsgIDandFuncAsso;
979         int
           i;
980
981         // get message id
982         msgID = (*(pM->pH))[
           eAPMsgHeaderPosition_msgTypeID];
983         // search handler
984         pMsgIDandFuncAsso = gHALMsgProcessRXHandlers.pL
           ;
985         for (i = 0; i < gHALMsgProcessRXHandlers.number;
           i++) {
986             if (((TAPMessageID)pMsgIDandFuncAsso->
           msgID) == msgID) {
987                 return pMsgIDandFuncAsso->
           pfktHandle(pAP, pM);
988             }
989             pMsgIDandFuncAsso++;
990         }
991         return -1;
992     }
993
994     void APMessageProcessingThread (TAP * pAP) {
995         TAPrealMsgSystem *
           pMS = (
           TAPrealMsgSystem *) pAP->MS;

```

```

996     TAPMsg *                pM;
997     TAPNodeID              recv;
998     while (1) {
999         // get the message
1000         pM = APMSgetMsg (pMS,0,0,0,0);
1001         if (!pM) goto error;
1002         // search the message handler
1003         recv = (*(pM->pH))[
1004             eAPMsgHeaderPosition_receiver];
1005         if ((recv == dAPNodeID_ALL) || (recv ==
1006             pAP->nodeID)) {
1007             if (APHandleMsg (pAP,pM)) goto
1008                 exit;
1009         }
1010         // free memory
1011         APMSdeleteMsg (pAP->MS,pM);
1012     }
1013 exit:
1014     // free memory
1015     APMSdeleteMsg (pAP->MS,pM);
1016 error:
1017     return;
1018 }
1019
1020 // if a message is in the queue available it will be
1021 // processed
1022 void APMessageProcess (TAP * pAP) {
1023     TAPrealMsgSystem *      pMS = (
1024         TAPrealMsgSystem *) pAP->MS;
1025     TAPMsg *                pM;
1026     TAPNodeID              recvID;
1027
1028     // search msg list
1029     pM = pMS->pOldRXMsg;
1030
1031     // if there is no message we will return
1032     if (!pM) return;
1033
1034     // unchain first message
1035     APMSunchainMessage(pMS, pM, NULL);
1036
1037     // check header
1038     recvID = (*(pM->pH))[
1039         eAPMsgHeaderPosition_receiver];
1040     if ((recvID == dAPNodeID_ALL) || (recvID == pAP
1041         ->nodeID)) {
1042         APHandleMsg (pAP,pM);
1043     }
1044     // free memory
1045     APMSdeleteMsg (pAP->MS,pM);
1046 }

```

3.5.6 audio processor blueprint 6 (pthreads and semaphores)

Informations:

description:	a super generic AP with multiple threads		
includes:			
c-Include	c-Library		system lib
AP.h			no

code:

```
1 // =====
2 // AP uuid = 10
3 // =====
4
5
6 // inits the AP
7 int APinit (
8
9             TAP *
10             pAP,
11             TAPNodeID
12             nodeID,
13             const TAPMsgDrv *      pDrvList
14             ,
15             const int
16             driverNumber ,
17             size_t
18             messagePoolSize ,
19             int
20             sysEndian
21
22 )
23 {
24     gAPendianFlag = sysEndian;
25
26     pAP->nodeID = nodeID;
27     pAP->pNodeList = NULL;
28     pAP->pDrvList = pDrvList;
29     pAP->driverNumber = driverNumber;
30     pAP->msgSysMMU = AP_MMU_create(messagePoolSize);
31     pAP->IP = APInterpreterCreate(pAP);
32     pAP->MS = APMScrate (pAP->msgSysMMU,sysEndian);
33     pAP->msgNumber = 0;
34     pAP->APstate = eAPstate_idle;
35
36     if(
37         (!pAP->msgSysMMU)||
38         (!pAP->IP)||
39         (!pAP->MS)
40     ) return -1;
41
42     // init drv
43     TAPMsgDrv * pDrv = (TAPMsgDrv *)pDrvList;
44     int i;
45     for (i = 0; i < driverNumber; i++) {
46         pDrv->pfkt_open(pAP, pDrv);
47     }
```

```

38         pDrv++;
39     }
40
41     // login the ap to the message system
42     return TX_login(pAP);
43 }
44
45 // deletes the AP
46 void APdelete (TAP * pAP)
47 {
48     // logout form all other devices
49     TX_logout(pAP);
50     // close & destroy drv
51     TAPMsgDrv * pDrv = (TAPMsgDrv *)pAP->pDrvList;
52     int i;
53     for (i = 0; i < pAP->driverNumber; i++) {
54         pDrv->pfkt_close(pDrv);
55         pDrv->pfkt_destroy(pDrv);
56         pDrv++;
57     }
58
59     APMSdelete (pAP->MS);
60     APInterpreterDelete(pAP->IP);
61     AP_MMU_delete(pAP->msgSysMMU);
62 }
63
64 // find a node at the list
65 TAPNode * APfindNode(TAP * pAP, TAPNodeID nodeID) {
66     TAPNode * pN = pAP->pNodeList;
67     while (pN) {
68         if (pN->nodeID == nodeID) return pN;
69         pN = pN->pNext;
70     };
71     return NULL;
72 }
73
74 // adds a new node to the node list
75 int APaddNode(TAP * pAP, TAPNodeID newNodeID, const
76     TAPMsgDrv * pDrv) {
77     if (APfindNode(pAP, newNodeID)) return 1;
78     TAPNode * pN = (TAPNode *) malloc(sizeof(TAPNode
79         ));
80     if (!pN) return -1;
81     pN->nodeID = newNodeID;
82     pN->pDrv = pDrv;
83     pN->pNext = pAP->pNodeList;
84     pAP->pNodeList = pN;
85     return 0;
86 }
87
88 // removes a node from the node list
89 void APremoveNode(TAP * pAP, TAPNodeID nodeID){
90     TAPNode * pAntN = pAP->pNodeList; // antecessor
91     node
92     TAPNode * pActN = pAP->pNodeList; // actual node

```

```

90
91     while (pActN) {
92         // compare node id's
93         if (pActN->nodeID == nodeID) {
94             // unchain
95
96             // check if we at the first
97             // position at the list
98             if (pAP->pNodeList == pAntN) {
99                 // reset the pointer
100                 pAP->pNodeList = pActN->
101                     pNext;
102             } else {
103                 // set the antecessor
104                 pAntN->pNext = pActN->
105                     pNext;
106             }
107             // free node
108             free(pActN);
109             // and abort
110             return;
111         }
112         // the actual element becomes the
113         // precessor element
114         pAntN = pActN;
115         pActN = pActN->pNext;
116     }
117 }
118
119 // get a new message number
120 unsigned int APgetNewMessageNumber (TAP *pAP) {
121     pAP->msgNumber++;
122     return pAP->msgNumber;
123 }
124
125 // find the driver associated with der nodeID
126 const TAPMsgDrv * APfindDrvBySenderID (TAP * pAP,
127     TAPNodeID node) {
128     TAPNode * pN = pAP->pNodeList;
129     while (pN) {
130         if (pN->nodeID == node) {
131             return pN->pDrv;
132         }
133         pN = pN->pNext;
134     }
135     return NULL;
136 }
137
138 // runs the AP
139 int APrun(TAP *pAP) {
140     pAP->APstate = eAPstate_run;
141     return 0;
142 }
143
144 typedef struct SAPrealMMUMemory {

```



```

140     uint32_t *
        pData;                                // the data
141     int
        count;                                // amount of
        data elements
142     struct SAPrealMMUMemory *      pNext;
        // next element
143     struct SAPrealMMUMemory *      pPrev;
        // previous element
144 } TAPrealMMUMemory;
145
146 //the mmu type
147 typedef struct SAPrealMMU {
148     uint32_t *                      memory;
        // the memory block
149     TAPrealMMUMemory *      pStart;
        // first element
150     TAPrealMMUMemory *      pEnd;
        // second element
151     TAPrealMMUMemory *      pUnusedList;
        // list with the unused elements
152     uint32_t *      pUnusedData;
        // pointer to the unused memory
153     int
        elementsAvailable;    // amount of elements
        witch are available without using the
        garbage collector
154     int
        totalAvailable;        // total amount of
        free bytes
155 } TAPrealMMU;
156
157 // =====
158 // memory entry functions
159 // =====
160
161 // a little macro for unchaining an element
162 #define DMemoryEntryUnchain(pM) \
163     if (pM->pNext) pM->pNext->pPrev = pM->pPrev; \
164     if (pM->pPrev) pM->pPrev->pNext = pM->pNext
165
166
167 //creates an memory entry
168 TAPrealMMUMemory * MemoryEntry_create () {
169     TAPrealMMUMemory * pM = NULL;
170
171     pM = (TAPrealMMUMemory *) malloc(sizeof(
        TAPrealMMUMemory));
172     if (!pM) return NULL;
173     pM->pData = NULL;
174     pM->count = 0;
175     pM->pNext = NULL;
176     pM->pPrev = NULL;
177
178

```

```

179         return pM;
180     }
181
182     //deletes an memory Entry
183     void MemoryEntry_delete (
184         TAPrealMMUMemory * pM    // the memory to
185             delete
186     )
187     {
188         // put the entry out of the chain
189         DMemoryEntryUnchain(pM);
190         // now we delete it
191         free(pM);
192     }
193
194     // =====
195     // mmu helper
196     // =====
197
198     //alloc if needed a new memory entry
199     TAPrealMMUMemory * MMU_helper_createMemoryEntry (
200         TAPrealMMU * pMMU    // MMU
201             structure to init
202     )
203     {
204         // check if we have to alloc a new memory entry
205         if (!pMMU->pUnusedList) return
206             MemoryEntry_create();
207         // no there is some left at the list
208         TAPrealMMUMemory * pM;
209         // take the first one
210         pM = pMMU->pUnusedList;
211         // reset the list
212         pMMU->pUnusedList = pM->pNext;
213         // now unchain the element (for sure)
214         DMemoryEntryUnchain(pM);
215         // set the element pointers
216         pM->pNext = NULL;
217         pM->pPrev = NULL;
218         return pM;
219     }
220
221     //the garbage collector
222     void MMU_helper_garbageCollector (
223         TAPrealMMU * pMMU    // MMU
224             structure to init
225     )
226     {
227         TAPrealMMUMemory * pM = pMMU->pStart;
228         uint32_t * pD = pMMU->memory;
229         while (pM) {
230             // check if we have to move the data
231             if (pD != pM->pData) {
232                 // move the data
233                 memmove(pD, pM->pData, pM->count *

```

```

230         sizeof(uint32_t));
231     }
232     // reset the destination pointer
233     pD += pM->count;
234     pM = pM->pNext;
235 }
236 // compressing memory finished
237 // set the mmu vars new
238 pMMU->elementsAvailable = pMMU->totalAvailable;
239 pMMU->pUnusedData = pD;
240 }
241
242 // create a mmu
243 TAPMMU AP_MMU_create (size_t elementsNumber) {
244     TAPrealMMU * pMMU;
245
246     pMMU = (TAPrealMMU *) malloc (sizeof(TAPrealMMU)
247 );
248     if (!pMMU) return NULL;
249
250     pMMU->memory = (uint32_t *) malloc (
251         elementsNumber*sizeof(uint32_t));
252     pMMU->pUnusedData = pMMU->memory;
253
254     // setup lists
255     pMMU->pStart = NULL;
256     pMMU->pEnd = NULL;
257     pMMU->pUnusedList = NULL;
258
259     pMMU->elementsAvailable =elementsNumber;
260
261     pMMU->totalAvailable = elementsNumber;
262     return pMMU;
263 }
264
265 // destroying the mmu
266 void AP_MMU_delete (TAPMMU mmu) {
267     TAPrealMMU * pMMU = (TAPrealMMU *) mmu;
268
269     TAPrealMMUMemory * pM;
270     TAPrealMMUMemory * pMnext;
271
272     // 1. delete al mmu entry's
273     // 1.1 unused entry
274     pM = pMMU->pUnusedList;
275     while (pM) {
276         pMnext = pM->pNext;
277         MemoryEntry_delete(pM);
278         pM = pMnext;
279     }
280     pMMU->pUnusedList = NULL;
281     // 1.2 used blocks

```

```

282     pM = pMMU->pStart;
283     while (pM) {
284         pMnext = pM->pNext;
285         MemoryEntry_delete(pM);
286         pM = pMnext;
287     }
288     pMMU->pStart = NULL;
289     pMMU->pEnd = NULL;
290 // 2. delete mmu memory
291     free (pMMU->memory);
292 }
293
294 // getting memmory from the mmu
295 TAPMMUmemory AP_MMU_get (TAPMMU mmu, size_t elements) {
296     TAPrealMMU * pMMU = (TAPrealMMU *) mmu;
297
298     // check if there is enough space
299     if (pMMU->totalAvailable < elements) goto error;
300     // check if we have to use the garbage collector
301     if (pMMU->elementsAvailable < elements) {
302         // start garbage collector
303         MMU_helper_garbageCollector(pMMU);
304     }
305     // we have enough memory so let's allocate some
306
307     // get a new entry
308     TAPrealMMUMemory * pM;
309     pM = MMU_helper_createMemoryEntry(pMMU);
310     if (!pM) return NULL;
311     // get some memory
312     pM->pData = pMMU->pUnusedData;
313     pM->count = elements;
314     // refresh data
315     pMMU->pUnusedData += elements;
316     pMMU->totalAvailable -= elements;
317     pMMU->elementsAvailable -= elements;
318     // insert memory element at the end of the list
319     // and update last element
320     pM->pPrev = pMMU->pEnd;
321     if (pMMU->pEnd) pMMU->pEnd->pNext = pM;
322     if (!pMMU->pStart) pMMU->pStart = pM;
323     pMMU->pEnd = pM;
324     return pM;
325 error:
326     return NULL;
327 }
328
329 // free memmory from the mmu
330 void AP_MMU_free (TAPMMU mmu, TAPMMUmemory memory) {
331     TAPrealMMU * pMMU = (TAPrealMMU *) mmu;
332     TAPrealMMUMemory * pM = (TAPrealMMUMemory *)
333         memory;
334     if (!pM) return;
335     // set mmu settings

```

```

335     if (pMMU->pStart == pM) pMMU->pStart = pM->pNext
336         ;
337     if (pMMU->pEnd == pM) pMMU->pEnd = pM->pPrev;
338     // unchain element
339     DMemoryEntryUnchain (pM);
340     // and put it to the chain of unused
341     pM->pNext = pMMU->pUnusedList;
342     pM->pPrev = NULL;
343     if (pMMU->pUnusedList) {
344         pMMU->pUnusedList->pPrev = pM;
345     }
346     pMMU->pUnusedList = pM;
347     // now set the mmu data new
348     pMMU->totalAvailable += pM->count;
349 }
350 // getting access to the MMU data
351 void * AP_MMU_getData (TAPMMUmemory memory) {
352     TAPrealMMUMemory * pM = (TAPrealMMUMemory *)
353         memory;
354     return pM->pData;
355 }
356 // =====
357 // the AP interpreter (thread save)
358 // =====
359 // the real interpreter
360 typedef struct SAPrealInterpreter {
361     TAP *
362         pAP; //
363         // pointer to the audio processor
364         int
365             state;
366             // state of the IP
367         int
368             nextState;
369             // the next state of the IP
370     TAPInterpreterCPU
371         cpu; // the IP
372         core
373         TAPInterpreterFuncCall *
374             code; // the code
375         int32_t
376             instructionCount; // number of
377             instructions at the code
378     TAPInterpreterVariable *
379         variables; // the variables
380         int32_t
381             variableCount; // number of
382             the variables
383         pthread_mutex_t
384             gM; // a
385             guarding mutex
386 } TAPrealInterpreter;

```

```

372
373 // create a new interpreter
374 TAPInterpreter APIInterpreterCreate (void * pAP) {
375     TAPrealInterpreter * pIP = NULL;
376     pIP = (TAPrealInterpreter *) malloc (sizeof(
        TAPrealInterpreter));
377     if (!pIP) return NULL;
378
379     pIP->pAP = pAP;
380     pIP->state = eAPIInterpreterState_idle;
381     pIP->nextState = eAPIInterpreterState_idle;
382     pIP->cpu.IP = pIP;
383     pIP->cpu.CF = 0;
384     pIP->cpu.EF = 0;
385     pIP->cpu.pCodeStart = NULL;
386     pIP->cpu.pCodeEnd = NULL;
387     pIP->cpu.pIP = NULL;
388
389     pIP->code = NULL;
390     pIP->instructionCount = 0;
391
392     pIP->variables = NULL;
393     pIP->variableCount = 0;
394
395     pIP->gM = PTHREAD_MUTEX_INITIALIZER;
396
397     return pIP;
398 }
399
400 // cleans the interpreter
401 void APIInterpreterClean (TAPInterpreter IP) {
402     TAPrealInterpreter * pIP = (TAPrealInterpreter
        *) IP;
403
404     // clean code
405     if (pIP->code) {
406         free (pIP->code);
407         pIP->code = NULL;
408     }
409     pIP->instructionCount = 0;
410
411     // clean variables
412     TAPInterpreterVariable * pV = pIP->variables;
413     int i;
414     for (i = 0; i < pIP->variableCount; i++) {
415         if (pV->pVI) pV->pVI->pFkt_delete(pV->
            pData);
416         pV++;
417     }
418     if (pIP->variables) {
419         free (pIP->variables);
420         pIP->variables = NULL;
421     }
422     pIP->variableCount = 0;
423

```

```

424 }
425
426 // deletes the interpreter
427 void APIInterpreterDelete (TAPIInterpreter IP) {
428     TAPrealInterpreter * pIP = (TAPrealInterpreter
429         *) IP;
430     APIInterpreterClean(IP);
431     free (pIP);
432 }
433
434 int APIInterpreterStateRun(TAPIInterpreter IP) {
435     TAPrealInterpreter * pIP = (TAPrealInterpreter
436         *) IP;
437     TAPIInterpreterFuncCall * pFC;
438
439     // setup cpu
440     pIP->cpu.CF = 0;
441     pIP->cpu.EF = 0;
442     pIP->cpu.pIP = pIP->code;
443     pIP->cpu.pCodeStart = pIP->code;
444     pIP->cpu.pCodeEnd = pIP->code + pIP->
445         instructionCount;
446
447     // run code
448     while (eAPIInterpreterState_run == pIP->state) {
449         pthread_mutex_lock(&pIP->gM);
450
451         pFC = pIP->cpu.pIP;
452         // check if we reached the end of the
453         // code
454         if (pFC > pIP->cpu.pCodeEnd) {
455             pthread_mutex_unlock(&pIP->gM);
456             return 0;
457         }
458         // execute command
459         pFC->pHALFkt (&(pIP->cpu), pFC->param);
460         // check error flags
461         if (pIP->cpu.EF) {
462             pthread_mutex_unlock(&pIP->gM);
463             return -1;
464         }
465         pthread_mutex_unlock(&pIP->gM);
466     }
467     return 1;
468 }
469
470 // process the actual state
471 int APIInterpreterProcessState(TAPIInterpreter IP){
472     TAPrealInterpreter * pIP = (TAPrealInterpreter
473         *) IP;
474     pthread_mutex_lock(&pIP->gM);
475     pIP->state = pIP->nextState;
476     int rc = 0;

```

```

474         switch (pIP->state) {
475             case eAPIInterpreterState_idle:
476                 break;
477             case eAPIInterpreterState_loadProgramm:
478                 break;
479             case eAPIInterpreterState_run:
480                 pthread_mutex_unlock(&pIP->gM);
481                 rc = APIInterpreterStateRun(IP);
482                 pthread_mutex_lock(&pIP->gM);
483                 if (rc >= 0) pIP->state =
484                     eAPIInterpreterState_idle;
485                 break;
486             case eAPIInterpreterState_oneStep:
487                 break;
488             case eAPIInterpreterState_halt:
489                 break;
490             default:
491                 pthread_mutex_unlock(&pIP->gM);
492                 return -10;
493         }
494         pthread_mutex_unlock(&pIP->gM);
495         return rc;
496     }
497
498 // set interpreter state
499 int APIInterpreterSetState (TAPIInterpreter IP, int
500 msgEndian, int32_t state) {
501     TAPrealInterpreter * pIP = (TAPrealInterpreter
502 *) IP;
503     pthread_mutex_lock(&pIP->gM);
504     APendianConversation32Bit((uint32_t *)&state,
505 msgEndian);
506     pIP->nextState = (int) state;
507     pthread_mutex_unlock(&pIP->gM);
508     return 0;
509 }
510
511 // get interpreter state
512 int32_t APIInterpreterGetState (TAPIInterpreter IP) {
513     TAPrealInterpreter * pIP = (TAPrealInterpreter
514 *) IP;
515     return (int32_t) pIP->state;
516 }
517
518 // setup the interpreter for a new program
519 int APIInterpreterInitNewProgramm (TAPIInterpreter IP, int
520 msgEndian, int32_t instructionsNumber, int32_t
521 VariableNumber) {
522     TAPrealInterpreter * pIP = (TAPrealInterpreter
523 *) IP;
524     int i;
525
526     pthread_mutex_lock(&pIP->gM);

```



```

521     APInterpreterClean (IP);
522
523     APendianConversation32Bit((uint32_t *)&
524         instructionsNumber, msgEndian);
525     APendianConversation32Bit((uint32_t *)&
526         VariableNumber, msgEndian);
527
528     pIP->code = (TAPInterpreterFuncCall *) malloc(
529         sizeof(TAPInterpreterFuncCall)*
530         instructionsNumber);
531     pIP->instructionCount = instructionsNumber;
532
533     pIP->variables = (TAPInterpreterVariable *)
534         malloc(sizeof(TAPInterpreterVariable) * (
535             VariableNumber));
536     for (i = 0; i < VariableNumber; i++) {
537         pIP->variables[i].pData = NULL;
538         pIP->variables[i].pVI = NULL;
539     }
540     pIP->variableCount = VariableNumber;
541
542     pthread_mutex_unlock(&pIP->gM);
543
544     return 0;
545 }
546
547 // load a variable/~array to an index
548 int APInterpreterLoadVar (TAPInterpreter IP, int
549     msgEndian, int32_t index, int32_t varTypeID, int32_t
550     numberOfElements)
551 {
552     TAPrealInterpreter * pIP = (TAPrealInterpreter
553         *) IP;
554
555     pthread_mutex_lock(&pIP->gM);
556
557     APendianConversation32Bit((uint32_t *)&index,
558         msgEndian);
559     APendianConversation32Bit((uint32_t *)&varTypeID,
560         msgEndian);
561     APendianConversation32Bit((uint32_t *)&
562         numberOfElements, msgEndian);
563
564     if ((index < 0) || (index > pIP->variableCount))
565         return -1;
566
567     // set pointer to the runtime variable
568     TAPInterpreterVariable * pRTV = pIP->variables +
569         index;
570     THAL_Variable const * pV = HALfindVar(varTypeID)
571         ;
572     if (!pV) {
573         pthread_mutex_unlock(&pIP->gM);
574     }
575 }

```

```

561         return -2;
562     }
563
564     pRTV->pData = pV->pFkt_create((unsigned int)
565         numberOfElements);
566     //if (!pRTV->pData) return -3;
567
568     pRTV->pVI = pV;
569     pthread_mutex_unlock(&pIP->gM);
570     return 0;
571 }
572
573 // load a single Instruction to an index
574 int APInterpreterLoadInstr (TAPInterpreter IP, int
575     msgEndian, int32_t index, int32_t * pRawInstr)
576 {
577     TAPrealInterpreter * pIP = (TAPrealInterpreter
578         *) IP;
579
580     pthread_mutex_lock(&pIP->gM);
581
582     APendianConversation32Bit((uint32_t *)&index,
583         msgEndian);
584
585     if ((index < 0) || (index > pIP->
586         instructionCount)){
587         pthread_mutex_unlock(&pIP->gM);
588         return -1;
589     }
590     TAPInterpreterFuncCall * pIFC = pIP->code +
591         index;
592     memset (pIFC, 0, sizeof(TAPInterpreterFuncCall))
593     ;
594
595     // get function
596     int32_t fid = *pRawInstr;
597     APendianConversation32Bit((uint32_t *)&fid,
598         msgEndian);
599
600     THALFunction const * pF = HALfindFunction(fid);
601     if (!pF) {
602         pthread_mutex_unlock(&pIP->gM);
603         return -2;
604     }
605     pIFC->pHALFkt = pF->pfktHAL;
606
607     // convert parameters
608     pRawInstr++; // set to the first parameter
609     int i;
610     THALFunctionParam const * pP = pF->paramList.pL;
611     TuAPIInterpreterFunctionParameter * pIFP = pIFC->
612         param;
613     for (i = 0; i < pF->paramList.number; i++) {
614         if (APconvertRawParamData (msgEndian,
615             pRawInstr, pP, pIFP, pIP->variables)) {

```

```

606         pthread_mutex_unlock(&pIP->gM);
607         return -3;
608     }
609     pP++;
610     pRawInstr++;
611     pIFP++;
612 }
613 pthread_mutex_unlock(&pIP->gM);
614 return 0;
615 }
616
617 // gets the variable by it's index
618 TAPInterpreterVariable * APInterpreterGetVariableByIndex
619 (TAPInterpreter IP, int index) {
620     return &(((TAPrealInterpreter *) IP)->variables[
621         index]);
622 }
623
624 // gets the AP from the IP
625 void * APInterpreterGetAPfromIP (TAPInterpreter IP) {
626     return ((TAPrealInterpreter *) IP)->pAP;
627 }
628
629 // gets the CPU from the IP
630 TAPInterpreterCPU * APInterpreterGetCPUref (
631     TAPInterpreter IP) {
632     return &(((TAPrealInterpreter *) IP)->cpu);
633 }
634
635 // =====
636 // the AP message system (thread save)
637 // =====
638
639 // =====
640 // the AP message system (thread save)
641 // =====
642
643 typedef struct SAPrealMsgSystem {
644     TAPMsg * pOldRXMsg;
645         // pointer to the oldest received
646         messages
647     TAPMsg * pNewRXMsg;
648         // pointer to the newest received
649         messages
650     TAPMMU mmu;
651         // the mmu
652     int sysEndianness; // the system endianness
653     int messagecounter; // a counter for checkin if a
654         new message has been received
655     sem_t waitSem;
656         // a semaphore to wait for a message
657     int

```

```

        getMsgCounter; // a counter incremented how
                        many threads calling getMsg and are waiting
649 pthread_mutex_t    gM;
                        // a guarding mutex
650 } TAPrealMsgSystem;
651
652
653 int SMinitial (
654     void *                pVoidSM,
                        // pointer to the
                        statemachine
655     uint32_t *            pD,
                        // pointer to the
                        data
656     int                   number
                        // the number of data
                        elements
657 );
658
659 int SMdata (
660     void *                pVoidSM,
                        // pointer to the
                        statemachine
661     uint32_t *            pD,
                        // pointer to the
                        data
662     int                   number
                        // the number of data
                        elements
663 );
664
665 int SMmessageFinished (
666     void *                pVoidSM
                        // pointer to the statemachine
667 );
668
669
670
671 // create AP message system
672 TAPMsgSystem APMScreate (
673     TAPMMU                mmu,
                        // the mmu
674     int                   sysEndianness // the system
                        endianness
675 ) {
676     TAPrealMsgSystem * pMS = (TAPrealMsgSystem *)
        malloc (sizeof(TAPrealMsgSystem));
677     if (!pMS) return NULL;
678     pMS->mmu = mmu;
679     pMS->sysEndianness = sysEndianness;
680     pMS->pOldRXMsg = NULL;
681     pMS->pNewRXMsg = NULL;
682     pMS->messagecounter = 0;
683 }

```

```

684     pMS->gM = PTHREAD_MUTEX_INITIALIZER;
685     sem_init (&pMS->waitSem,0,0);
686     pMS->getMsgCounter = 0;
687
688     return pMS;
689 }
690
691 void APMSdelete (
692     TAPMsgSystem ms
693 ) {
694     TAPrealMsgSystem * pMS = (TAPrealMsgSystem *) ms
        ;
695     sem_destroy(&pMS->waitSem);
696     free (pMS);
697 }
698
699
700 // frees a message from the message system
701 void APMSdeleteMsg (
702     TAPMsgSystem      ms,
703     TAPMsg *          pM
704 ) {
705     TAPrealMsgSystem * pMS = (TAPrealMsgSystem *) ms
        ;
706     pthread_mutex_lock(&pMS->gM);
707
708     AP_MMU_free(pMS->mmu,pM->memory);
709
710     pthread_mutex_unlock(&pMS->gM);
711 }
712
713 // get memory for a new message
714 TAPMsg * APMSgetNewMsg (
715     TAPrealMsgSystem *      pMsgSys,
716     int                     dataElementsNumber,
717     const TAPMsgDrv *       pDrv
718 ) {
719     pthread_mutex_lock(&pMsgSys->gM);
720
721     // we go for shure that we get enough memory
722     // if mod(sizeof(TAPMsg)/sizeof(uint32_t)) != 0
723     // we need one uint32_t more -> +1
724     TAPMMUmemory m = AP_MMU_get (pMsgSys->mmu,
725         sizeof(TAPMsg)/sizeof(uint32_t) + 1+
726         eAPMsgHeaderPosition_headerElementNumber +
727         dataElementsNumber);
728     if (!m) return NULL;
729
730     // set the pointers
731     uint32_t * pRD = (uint32_t *) AP_MMU_getData(m);
732     TAPMsg * pM = (TAPMsg *) pRD;
733     pM->memory = m;
734     pM->extraData.pDrv = pDrv;
735     pM->pH = (TAPMsgHeader *)((uint32_t *) pRD +

```

```

732         sizeof(TAPMsg)/sizeof(uint32_t)+1);
733     pM->pData = (uint32_t *)pM->pH + sizeof(
734         TAPMsgHeader)/sizeof(uint32_t);
735     pM->pNext = NULL;
736     pthread_mutex_unlock(&pMsgSys->gM);
737     return pM;
738 }
739 // insert a new message into the message queue
740 void APMSInsertMsg (
741     TAPrealMsgSystem *      pMS,
742     TAPMsg *                pM
743 ) {
744     pthread_mutex_lock(&pMS->gM);
745     if (pMS->pNewRXMsg) {
746         pMS->pNewRXMsg->pNext = pM;
747     }
748     pMS->pNewRXMsg = pM;
749     if (!pMS->pOldRXMsg) {
750         pMS->pOldRXMsg = pM;
751     }
752     pMS->messagecounter++;
753     pthread_mutex_unlock(&pMS->gM);
754     sem_post(&pMS->waitSem);
755 }
756
757 // get oldest message
758 TAPMsg * APMSgetMsg (
759     TAPMsgSystem            ms,
760     TAPMessageID            msgID,
761     // if 0 all messages are
762     allowed
763     TAPNodeID              sender,
764     // if 0 all senders
765     are allowed
766     uint32_t                mNumber,
767     // if 0 all numbers
768     are allowed
769     int                      ackMsgAllowed
770 ) {
771     TAPrealMsgSystem * pMS = (TAPrealMsgSystem *) ms
772     ;
773     // flags
774     int senderOK;
775     int msgIDok;
776     int numberOK;
777     // result var
778     TAPMsg * res = NULL;
779 }

```

```

776     pthread_mutex_lock(&pMS->gM);
777     pMS->getMsgCounter++;
778     pthread_mutex_unlock(&pMS->gM);
779
780     checkMessages:
781         pthread_mutex_lock(&pMS->gM);
782
783         // search msg list
784         TAPMsg * pM = pMS->pOldRXMsg;
785         TAPMsg * pAntecessorM = NULL;
786         uint32_t * pH;
787
788         if (!pM) goto waitForMessage;
789
790
791         senderOK = 0;
792         msgIDok = 0;
793         numberOK = 0;
794
795         pH = *(pM->pH);
796
797         if (!sender) {
798             senderOK = 1;
799         } else {
800             if (pH[eAPMsgHeaderPosition_sender] ==
801                 sender) senderOK = 1;
802         }
803         if (!msgID) {
804             // filter ack/nack msg
805             if (ackMsgAllowed) {
806                 msgIDok = 1;
807             } else {
808                 if (
809                     (pH[
810                         eAPMsgHeaderPosition_msgTypeID
811                     ] !=
812                     eAPMsgTypes_ACK
813                     ) &&
814                     (pH[
815                         eAPMsgHeaderPosition_msgTypeID
816                     ] !=
817                     eAPMsgTypes_NACK
818                     )
819                     ) {
820                         msgIDok = 1;
821                     }
822             }
823         } else {
824             if (pH[eAPMsgHeaderPosition_msgTypeID]
825                 == msgID) msgIDok = 1;
826         }
827         if (!mNumber) {
828             numberOK = 1;
829         } else {
830             if (pH[eAPMsgHeaderPosition_msgNumber]

```

```

821         == mNumber) numberOK = 1;
822     }
823     if ((senderOK) && (msgIDok) && (numberOK)) {
824         res = pM;
825         goto exit;
826     }
827     pAntecessorM = pM;
828     pM = pM->pNext;
829     if (pM) goto checkMessages;
830 waitForMessage:
831     pthread_mutex_unlock(&pMS->gM);
832     if (sem_wait(&(pMS->waitSem)) == -1) goto error;
833
834     // if other threads are waiting for a message
835     // give the sign to them
836     pthread_mutex_lock(&pMS->gM);
837     if (pMS->getMsgCounter > 1) {
838         if (sem_post(&pMS->waitSem) == -1) goto
839         error;
840     }
841     pthread_mutex_unlock(&pMS->gM);
842     goto checkMessages;
843
844 exit:
845     if (pAntecessorM) {
846         pAntecessorM->pNext = pM->pNext;
847     } else {
848         pMS->pOldRXMsg = pM->pNext;
849     }
850     if (pM == pMS->pNewRXMsg) {
851         pMS->pNewRXMsg = NULL;
852     }
853
854     // now there is one message less left
855     pMS->messagecounter--;
856     // now one thread is less waiting for a message
857     pMS->getMsgCounter--;
858     pthread_mutex_unlock(&pMS->gM);
859     return res;
860 error:
861     pthread_mutex_unlock(&pMS->gM);
862     return NULL;
863 }
864
865 // =====
866 // the receive state machine
867 // =====
868
869 // the receive state machine state function for
870 // receiving the msg header
871 int SMinitial (
872     void *                pVoidSM,
873                        // pointer to the
874                        statemachine
875     uint32_t *            pD,

```



```

// pointer to the
data
int number
// the number of data
elements
) {
TAPReceiveStateMachine *
pSM = (TAPReceiveStateMachine *) pVoidSM;
TAPrealMsgSystem *
pMS = (TAPrealMsgSystem *) pSM->pMS;
int
copyAmount = number;
int
i;

// 1. try to copy the data to the header
if (pSM->elementsLeft < copyAmount) copyAmount =
pSM->elementsLeft;
// copy
for (i = 0; i < copyAmount; i++) {
    *pSM->pD = *pD;
    pSM->pD++;
    pD++;
}
pSM->elementsLeft -= copyAmount;

// check if we have to change the statemachine
because we received the header
if (pSM->elementsLeft) return 0;

// yes! alloc msg buffer and (opt.) transfer
data

// 2. convert endian
int msgEndian = pSM->header[
eAPMsgHeaderPosition_endian];
for (i = 0; i <
eAPMsgHeaderPosition_headerElementNumber; i++)
{
    ApendianConversation32Bit(&pSM->
header[i], msgEndian);
}
// 3. now alloc message
// 3.1 get length
int msgElementNumber = (int) pSM->header[
eAPMsgHeaderPosition_length];
// 3.2. get memory
pSM->pMsg = APMSgetNewMsg (pMS, msgElementNumber,
pSM->pDrv);
if (!pSM->pMsg) return -100;

// 3.3 check getMemory result
if (!pSM->pMsg) return -1;
// copy message header
pSM->pD = (uint32_t *) pSM->pMsg->pH;

```

```

908     for (i = 0; i <
        eAPMsgHeaderPosition_headerElementNumber;i++)
        {
909         *pSM->pD = pSM->header[i];
910         pSM->pD++;
911     }
912     pSM->elementsLeft = pSM->header[
        eAPMsgHeaderPosition_length];
913     // set up the data
914     // 1. check if there is an data element
915     if (!pSM->elementsLeft) {
916         // no! now finish the message
917         return SMmessageFinished(pVoidSM);
918     }
919     // 2. yes
920     // 2.1 setup the sm for the data receiving
921     pSM->state = SMdata;
922     // 2.2 now check if we have to copy some data
923     number -= copyAmount;
924     if (number) {
925         // set the data pointer
926         pD += copyAmount;
927         // and copy the data
928         return SMdata (pVoidSM,pD,number);
929     }
930     return 0;
931 }
932
933 // the receive state machine state function for
934 // receiving the data
935 int SMdata (
936     void *                pVoidSM,
937     // pointer to the
938     statemachine
939     uint32_t *            pD,
940     // pointer to the
941     data
942     int                    number
943     // the number of data
944     elements
945 ) {
946     TAPReceiveStateMachine *
947     pSM = (TAPReceiveStateMachine *) pVoidSM;
948     int
949         copyAmount = number;
950     int
951         i;
952     // 1. transfer the data
953     // do some clipping
954     if (pSM->elementsLeft < copyAmount) copyAmount =
955         pSM->elementsLeft;
956     // copy
957     for (i = 0; i < copyAmount;i++) {
958         *pSM->pD = *pD;

```

```

949         pSM->pD++;
950         pD++;
951     }
952     // set statemachine work data
953     pSM->elementsLeft -= copyAmount;
954     // check if we have to change the statemachine
955     if (pSM->elementsLeft) return 0;
956     int res = SMmessageFinished (pVoidSM);
957     if (res) return res;
958
959     // check if there some bytes left to copy
960     number -= copyAmount;
961     if (number) {
962         // set the data pointer
963         pD += copyAmount;
964         // and copy the data
965         return pSM->state (pVoidSM,pD,number);
966     }
967     return 0;
968 }
969 }
970
971 // this function is called when all data have been
972 // received
973 int SMmessageFinished (
974     void *                pVoidSM
975     // pointer to the statemachine
976 ) {
977     TAPReceiveStateMachine *
978     pSM = (TAPReceiveStateMachine *) pVoidSM;
979     TAPMsg *
980     pM;
981
982     // 1. reset SM
983     // set the helper
984     pSM->elementsLeft =
985         eAPMsgHeaderPosition_headerElementNumber;
986     pSM->pD = pSM->header;
987
988     // data
989     pM = pSM->pMsg; // save msg info for inserting
990     pSM->pMsg = NULL;
991
992     // right state function
993     pSM->state = SInitial; // the state
994
995     // 2. insert message at the message system
996     APMSInsertMsg ((TAPPrealMsgSystem *)pSM->pMS,pM);
997     return 0;
998 }
999
1000 // inits the state machine
1001 void APInitReceiveStateMachine (
1002     TAPReceiveStateMachine *
1003     pSM, // pointer to the

```

```

state machine
998     TAPMsgSystem
                                pMS,    // pointer to
                                the message system
999     const TAPMsgDrv *
                                pDrv    // the driver
                                associated with the statemachine
1000 ) {
1001     pSM->state = SInitial;
1002     pSM->pMS = pMS;
1003     pSM->pDrv = pDrv;
1004     // set the helper
1005     pSM->elementsLeft =
1006         eAPMsgHeaderPosition_headerElementNumber;
1007     pSM->pD = pSM->header;
1008     // data
1009     pSM->pMsg = NULL;
1010 }
1011
1012 int APHandleMsg (
1013     TAP *
1014     TAPMsg *
1015     pAP,
1016     pM
1017     TAPMessageID
1018     msgID;
1019     const THALMsgProcessMessageAssociation *
1020     pMsgIDandFunctAsso;
1021     int
1022     i;
1023
1024     // get message id
1025     msgID = (*(pM->pH))[
1026         eAPMsgHeaderPosition_msgTypeID];
1027     // search handler
1028     pMsgIDandFunctAsso = gHALMsgProcessRXHandlers.pL
1029     ;
1030     for (i = 0; i < gHALMsgProcessRXHandlers.number;
1031         i++) {
1032         if (((TAPMessageID)pMsgIDandFunctAsso->
1033             msgID) == msgID) {
1034             return pMsgIDandFunctAsso->
1035                 pfktHandle(pAP, pM);
1036         }
1037         pMsgIDandFunctAsso++;
1038     }
1039     return -1;
1040 }
1041
1042 void APMessageProcessingThread (TAP * pAP) {
1043     TAPPrealMsgSystem *
1044     TAPPrealMsgSystem *) pAP->MS;
1045     TAPMsg *
1046     pMS = (
1047     pM;

```

```

1038     TAPNodeID                                     recv;
1039     while (1) {
1040         // get the message
1041         pM = APMSgetMsg (pMS,0,0,0,0);
1042         if (!pM) goto error;
1043         // search the message handler
1044         recv = (*(pM->pH))[
1045             eAPMsgHeaderPosition_receiver];
1046         if ((recv == dAPNodeID_ALL) || (recv ==
1047             pAP->nodeID)) {
1048             if (APHandleMsg (pAP,pM)) goto
1049                 exit;
1050         }
1051         // free memory
1052         APMSdeleteMsg (pAP->MS,pM);
1053     }
1054 exit:
1055     // free memory
1056     APMSdeleteMsg (pAP->MS,pM);
1057 error:
1058     return;
1059 }

```