Audio Processor definition and implementation documentation

Stefan Jaritz

July 10, 2013

Contents

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 jungest	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	koefficent	rational
3	indx	index	integer
4	array2	array nummero 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 = 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 = 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 != 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 <= 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 >= 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 writen 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 frequnecy	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 frequnecy	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 frequnecy	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 ferquency 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 frequnecy	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 ferquency 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 frequnecy	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 rational
3	AT	attack value for the smoothing	basic io type	basic type rational
4	RT	release value for the smoothing	basic io type	basic type rational
5	NT	noise cut off threshold	basic io type	basic type rational
6	NS	slope	basic io type	basic type rational

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 rational
3	AT	attack value for the smoothing	basic io type	basic type rational
4	RT	release value for the smoothing	basic io type	basic type rational
5	ET	expander threshold	basic io type	basic type rational
6	ES	slope	basic io type	basic type rational

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 rational
3	AT	attack value for the smoothing	basic io type	basic type rational
4	RT	release value for the smoothing	basic io type	basic type rational
5	CT	compressor threshold	basic io type	basic type rational
6	CS	slope	basic io type	basic type rational

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 (Tdelay = N * Ta)	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,yle,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	fbc	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:

```
TX_logout [handle=none]() {
        declare recv    Node;
        declare mNum    mNum;

```

```
        recv = ALL;
        loopAll drivers[driver] {
                mNum = getNewMsgNum();
                driver>>send(logout(recv,mNum));
        }
return 0;
}
```

2.2.2 updateVariable (msgProcId =0)

Informations:

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

code:

```
TX_updateVariable [handle=none](VarIndex vi) {
        declare mNum    mNum;
        declare apV             Variable;

        apV = getVariableByIndex(vi);

        loopAll nodes[node] {
                if (getNodeIDfromNode(node) !=
                   getSelfSenderID()) {
                        mNum = getNewMsgNum();
                        apV>>call(sendUpdate(
                           getVariableData(apV),
                           getDriverFromNode(node),ALL,
                           mNum,vi));
                        if (waitACK(mNum)) {
                                return -1;
                        }
                }
        }
        return 0;
}
```

2.2.3 login (msgProcId =10)

Informations:

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

code:

```
TX_login [handle=none]() {
        declare recv    Node;
        declare mNum    mNum;

        recv = ALL;
        loopAll drivers[driver] {
                mNum = getNewMsgNum();
                driver>>send(login(recv,mNum));
        }
return 0;
}
```

2.2.4 run (msgProcId =20)

Informations:

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

code:

```
TX_run [handle=none](Node dest) {
        declare driver  Driver;
        declare sender  Node;
        declare mNum    mNum;

        sender = getSelfSenderID();
        if (dest != ALL) {
                mNum = getNewMsgNum();
                driver = getDriver(dest);
                driver>>send(run(sender,mNum));
                return waitACK(mNum);
        }

        loopAll drivers[drv] {
                mNum = getNewMsgNum();
                drv>>send(run(sender,mNum));
                if (waitACK(mNum)) {
                        return -1;
                }
        }
        return 0;
}
```

2.2.5 run (msgProcId =22)

Informations:

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

code:

```
RX_run [handle=run]() {
        declare sender                  Node;
        declare driver                  Driver;
        declare mNum                    mNum;

        sender = getSender();
        driver = getDriver();
        mNum = getMsgNum();

        if (!runAP()) {
                return driver>>send(ACK(sender,mNum));
        } else {
                return driver>>send(NACK(sender,mNum));
        }
}
```

2.2.6 updateVariable (msgProcId =23)

Informations:

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

code:

```
RX_updateVariable [handle=updateVariable]() {
        declare sender                  Node;
        declare driver                  Driver;
        declare mNum                    mNum;
        declare apV                             Variable
           ;
        declare gIndex                  int;

        sender = getSender();
        driver = getDriver();
        mNum = getMsgNum();
        gIndex = getMsgDataToInt(0);

        apV = getVariableByIndex(gIndex);
        if (!apV) {
```

```
                return -1;
        }

        apV>>call(recvUpdate(getVariableData(apV),
           getMsgDataRef()));

        return driver>>send(ACK(sender,mNum));
}
```

2.2.7 login (msgProcId =30)

Informations:

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

code:

```
RX_login [handle=login]() {
        declare sender  Node;
        declare driver  Driver;
        declare mNum    mNum;

        sender = getSender();
        driver = getDriver();
        mNum = getMsgNum();

        if (!addNode(sender, driver)) {
                return driver>>send(login(sender,mNum));
        }
        return 0;
}
```

2.2.8 logout (msgProcId =31)

Informations:

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

code:

```
RX_logout [handle=logout]() {
        declare sender  Node;

        sender = getSender();

```

```
            removeNode(sender);
            return 0;
}
```

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:

```
// The following definition allows the SRU macro to check for errors. Once the routings have
// been verified, this definition can be removed to save some program memory space.
// The preprocessor will issue a warning stating this when using the SRU macro without this
// definition
#define SRUDEBUG  // Check SRU Routings for errors.
#include <SRU.h>


#define NUM_SAMPLES (1024)
#define DAC4

// ===========================================================
```

```
// AD 1835 defines
// =====================================================

//
//  AD1835.h
//
//  Configuration values for the AD1835A codec
//

#define DACCTRL1        (0x0000)  // DAC control register 1   (R/W)
#define DACCTRL2        (0x1000)  // DAC control register 2   (R/W)
#define DACVOL_L1    (0x2000)  // DAC volume - left 1   (R/W)
#define DACVOL_R1    (0x3000)  // DAC volume - right 1  (R/W)
#define DACVOL_L2    (0x4000)  // DAC volume - left 2   (R/W)
#define DACVOL_R2    (0x5000)  // DAC volume - right 2  (R/W)
#define DACVOL_L3    (0x6000)  // DAC volume - left 3   (R/W)
#define DACVOL_R3    (0x7000)  // DAC volume - right 3  (R/W)
#define DACVOL_L4    (0x8000)  // DAC volume - left 4   (R/W)
#define DACVOL_R4    (0x9000)  // DAC volume - right 4  (R/W)
#define ADCPEAKL        (0xA000)  // ADC left peak        (R)
#define ADCPEAKR        (0xB000)  // ADC right peak      (R)
#define ADCCTRL1        (0xC000)  // ADC control 1       (R/W)
#define ADCCTRL2        (0xD000)  // ADC control 2       (R/W)
#define ADCCTRL3        (0xE000)  // ADC control 3       (R/W)

#define RD              (0x0800)
#define WR              (0x0000)  // Write to register


// DAC control register 1
#define DEEMPH44_1      (0x0100)  // Deemphasis filter for 44.1 KHz
#define DEEMPH32        (0x0200)  // Deemphasis filter for 32.0 KHz
#define DEEMPH48        (0x0300)  // Deemphasis filter for 48.0 KHz

#define DACI2S          (0x0000)  // DAC receives I2S
```

```
format
#define DACRJ           (0x0020)  // DAC receives I2S format
#define DACDSP          (0x0040)  // DAC receives I2S format
#define DACLJ           (0x0060)  // DAC receives I2S format
#define DACPACK256      (0x0080)  // DAC receives I2S format

#define DAC24BIT        (0x0000)  // 24-bit output word length
#define DAC20BIT        (0x0008)  // 20-bit output word length
#define DAC16BIT        (0x0010)  // 16-bit output word length

#define DACPOWERDN      (0x0004)  // DAC into power-down mode

#define DACFS48         (0x0000)  // Sample rate = 48 KHz (x8)
#define DACFS96         (0x0001)  // Sample rate = 96 KHz (x4)
#define DACFS192        (0x0002)  // Sample rate = 192 KHz (x2)


// DAC control register 2

#define DACREPLICATE  (0x0100)  // Replicate output of DAC 1/2 on 3/4, 5/6 & 7/8
#define DACMUTE_R4    (0x0080)  // Mute DAC output channel (clear to un-mute)
#define DACMUTE_L4    (0x0040)  // Mute DAC output channel (clear to un-mute)
#define DACMUTE_R3    (0x0020)  // Mute DAC output channel (clear to un-mute)
#define DACMUTE_L3    (0x0010)  // Mute DAC output channel (clear to un-mute)
#define DACMUTE_R2    (0x0008)  // Mute DAC output channel (clear to un-mute)
#define DACMUTE_L2    (0x0004)  // Mute DAC output channel (clear to un-mute)
#define DACMUTE_R1    (0x0002)  // Mute DAC output channel (clear to un-mute)
#define DACMUTE_L1    (0x0001)  // Mute DAC output channel (clear to un-mute)


// ----------------------------------------------------------------------
//DAC Volume Control - 10-bit granularity (1024 levels)
#define DACVOL_MIN      (0x000)
```

```
#define DACVOL_LOW      (0X100)
#define DACVOL_MED      (0X200)
#define DACVOL_HI       (0X300)
#define DACVOL_MAX      (0x3FF)
#define DACVOL_MASK     (0x3FF)  // Volume in dB is in 10 LSBs
                                 //   3FF = 0 dBFS = 1023/1023
                                 //   3FE = -0.01 dBFS = 1022/1023
                                 //      ...
                                 //   002 = -50.7 dBFS = 3/1023
                                 //   001 = -54.2 dBFS = 2/1023

// ---------------------------------------------------------------------
//  ADC Control 1

#define ADCHPF      (0x0100)  // High pass filter (AC-coupled)
#define ADCPOWERDN  (0x0080)  // DAC into power-down mode
#define ADCFS48     (0x0000)  // Sample rate = 48 KHz
#define ADCFS96     (0x0040)  // Sample rate = 96 KHz

// ---------------------------------------------------------------------
//  ADC Control 2

#define AUXSLAVE    (0x0000)  // Aux input is in slave mode
#define AUXMASTER   (0x0200)  // Aux input is in master mode

#define ADCI2S      (0x0000)  // ADC transmits in I2S format
#define ADCRJ       (0x0040)  // ADC transmits in right-justified format
#define ADCDSP      (0x0080)  // ADC transmits in DSP (TDM) format
#define ADCLJ       (0x00C0)  // ADC transmits in left-justified format
#define ADCPACK256  (0x0100)  // ADC transmits in packed 256 format
#define ADCAUX256   (0x0180)  // ADC transmits in packed 128 format

#define ADC24BIT    (0x0000)  // 24-bit output word length
#define ADC20BIT    (0x0010)  // 20-bit output word length
#define ADC16BIT    (0x0020)  // 16-bit output word
```

```
   length

#define ADCMUTER   (0x0002)  // Mute right channel from
   ADC
#define ADCMUTEL   (0x0001)  // Mute right channel from
   ADC

//
   ---------------------------------------------------------------------

//  ADC Control 3

#define IMCLKx2    (0x0000)  // Internal MCLK = external
    MCLK x 2
#define IMCLKx1    (0x0040)  // Internal MCLK = external
    MCLK
#define IMCLKx23   (0x0080)  // Internal MCLK = external
    MCLK x 2/3

#define PEAKRDEN   (0x0020)  // Enable reads of peak ADC
    levels
#define PEAKLEVELMASK  (0x003F)  // Six significant bit
   of level



//
   =======================================================
// talk through interface
//
   =======================================================

// Function prototypes for this talkthrough code

extern void InitPLL_SDRAM(void);
extern void processBlock(unsigned int *);

extern void InitSRU(void);
extern void Init1835viaSPI(void);
```

```
extern void InitSPORT(void);
extern void TalkThroughISR(int);
extern void ClearSPORT(void);

extern void SetupSPI1835 (void) ;
extern void DisableSPI1835 (void) ;
extern void Configure1835Register (int i) ;
extern unsigned int Get1835Register (int i) ;

extern void SetupIRQ01 (void) ;
extern void Irq0ISR (int i) ;
extern void Irq1ISR (int i) ;

typedef void (* TFkt_ADSPuartCB) (unsigned int value);

extern void initUART(TFkt_ADSPuartCB cbRXFunction);
extern void UARTisr(int i);
extern void sendUARTuint32Values(unsigned int *pD, int
    amount);

extern void Delay (int i) ;

//
    ==========================================================
// init AD1835
//
    ==========================================================
/* Setup the SPI pramaters here in a buffer first */
unsigned int Config1835Param [] = {
                WR | DACCTRL1 | DACI2S | DAC24BIT | DACFS48,
                WR | DACCTRL2 ,//| DACMUTE_R4 | DACMUTE_L4,
                WR | DACVOL_L1 | DACVOL_MAX,
                WR | DACVOL_R1 | DACVOL_MAX,
                WR | DACVOL_L2 | DACVOL_MAX,
                WR | DACVOL_R2 | DACVOL_MAX,
                WR | DACVOL_L3 | DACVOL_MAX,
                WR | DACVOL_R3 | DACVOL_MAX,
                WR | DACVOL_L4 | DACVOL_MAX,
                WR | DACVOL_R4 | DACVOL_MAX,
                WR | ADCCTRL1 | ADCFS48,
                WR | ADCCTRL2 | ADCI2S | ADC24BIT,
                WR | ADCCTRL3 | IMCLKx2
        } ;

volatile int spiFlag ;

//Set up the SPI port to access the AD1835
void SetupSPI1835 ()
{
    /* First configure the SPI Control registers */
    /* First clear a few registers      */
    *pSPICTL = (TXFLSH | RXFLSH) ;
```

```c
    *pSPIFLG = 0;
    *pSPICTL = 0;

    /* Setup the baud rate to 500 KHz */
    *pSPIBAUD = 100;

    /* Setup the SPI Flag register to FLAG3 : 0xF708*/
    *pSPIFLG = 0xF708;

    /* Now setup the SPI Control register : 0x5281*/
    *pSPICTL = (SPIEN | SPIMS | MSBF | WL16 | TIMOD1) ;

}

//Disable the SPI Port
void DisableSPI1835 ()
{
    *pSPICTL = (TXFLSH | RXFLSH);
}

//Send a word to the AD1835 via SPI
void Configure1835Register (int val)
{
    *pTXSPI = val ;
    Delay(100);

    //Wait for the SPI to indicate that it has finished.
    while (1)
    {
        if (*pSPISTAT & SPIF)
            break ;
    }
    Delay (100) ;
}

//Receive a register setting from the AD1835
unsigned int Get1835Register (int val)
{
    *pTXSPI = val ;
    Delay(100);

    //Wait for the SPI port to indicate that it has
       finished
    while (1)
    {
        if (SPIF & *pSPISTAT)
            break ;
    }
    Delay (100) ;
    return *pRXSPI ;
//  return i ;
}

//Set up all AD1835 registers via SPI
void Init1835viaSPI()
```

```
{
    int configSize = sizeof (Config1835Param) / sizeof (
       int) ;
    int i ;

    SetupSPI1835 () ;

    for (i = 0; i < configSize; ++i)
    {
        Configure1835Register (Config1835Param[i]) ;
    }

    DisableSPI1835 () ;

}

//Delay loop
void Delay (int i)
{
    for (;i>0;--i)
        asm ("nop;") ;
}

//
   ============================================================

// PLL for SDRAM init
//
   ============================================================

void InitPLL_SDRAM(){

int i, pmctlsetting;

//Change this value to optimize the performance for
   quazi-sequential accesses (step > 1)
#define SDMODIFY 1

    pmctlsetting= *pPMCTL;
    pmctlsetting &= ~(0xFF); //Clear

    // CLKIN= 24.576 MHz, Multiplier= 27, Divisor= 1,
       INDIV=1, CCLK_SDCLK_RATIO= 2.
    // Core clock = (24.576 MHz * 27) /2 = 331.776 MHz
    pmctlsetting= SDCKR2|PLLM27|INDIV|DIVEN;
    *pPMCTL= pmctlsetting;
    pmctlsetting|= PLLBP;
    *pPMCTL= pmctlsetting;

    //Wait for around 4096 cycles for the pll to lock.
    for (i=0; i<4096; i++)
          asm("nop;");

    *pPMCTL ^= PLLBP;        //Clear Bypass Mode
    *pPMCTL |= (CLKOUTEN);   //and start clkout
```

```


    // Programming SDRAM control registers and enabling SDRAM read optimization
    // CCLK_SDCLK_RATIO= 2.5
    // RDIV = ((f SDCLK X t REF )/NRA) - (tRAS + tRP )
    // (166*(10^6)*64*(10^-3)/4096) - (7+3) = 2583

    *pSDRRC= (0xA17)|(SDMODIFY<<17)|SDROPT;

    // =================================================================
    //
    // Configure SDRAM Control Register (SDCTL) for PART MT48LC4M32B2
    //
    //  SDCL3   : SDRAM CAS Latency= 3 cycles
    //  DSDCLK1: Disable SDRAM Clock 1
    //  SDPSS   : Start SDRAM Power up Sequence
    //  SDCAW8  : SDRAM Bank Column Address Width= 8 bits
    //  SDRAW12: SDRAM Row Address Width= 12 bits
    //  SDTRAS7: SDRAM tRAS Specification. Active Command delay = 7 cycles
    //  SDTRP3 : SDRAM tRP Specification. Precharge delay = 3 cycles.
    //  SDTWR2 : SDRAM tWR Specification. tWR = 2 cycles.
    //  SDTRCD3: SDRAM tRCD Specification. tRCD = 3 cycles.
    //
    // -----------------------------------------------------------------

    *pSDCTL= SDCL3|DSDCLK1|SDPSS|SDCAW8|SDRAW12|SDTRAS7|SDTRP3|SDTWR2|SDTRCD3;

    // Note that MS2 & MS3 pin multiplexed with flag2 & flag3.
    // MSEN bit must be enabled to access SDRAM, but LED7 cannot be driven with sdram
    *pSYSCTL |=MSEN;

    // Mapping Bank 2 to SDRAM
    // Make sure that jumper is set appropriately so that MS2 is connected to
    // chip select of 16-bit SDRAM device
    *pEPCTL |=B2SD;
    *pEPCTL &= ~(B0SD|B1SD|B3SD);

    // =================================================================
    //
```

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

    //SRAM Settings
    *pAMICTL0 = WS2|HC1|AMIEN|BW8;

    //
       ==================================================================

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

    //Flash Settings
    *pAMICTL1 = WS23|AMIEN|BW8;
}

//
   =======================================================

// serial Port
//
   =======================================================

/*
   Here is the mapping between the SPORTS and the DACS
   ADC -> DSP  : SPORT0A : I2S
   DSP -> DAC1 : SPORT1A : I2S
   DSP -> DAC2 : SPORT1B : I2S
   DSP -> DAC3 : SPORT2A : I2S
   DSP -> DAC4 : SPORT2B : I2S
*/

unsigned int PCI = 0x00080000 ;
unsigned int OFFSET = 0x00080000 ;

// TCB blocks for Chaining
```

```
//Each block will be used for:
//      Filling from the ADC
//      Processing filled data
//      Sending to DAC
//
//Each one is doing only one of these steps for each SPORT interrupt.

//For this example the startup state is
// Start to 1st interrupt: gBlock_A is filled first, gBlock_C is sent
// 1st int to 2nd int: gBlock_C filled, gBlock_A processed, gBlock_B sent
// 2nd int to 3rd int: gBlock_B filled, gBlock_C processed, gBlock_A sent
// 3rd int to 4th int: gBlock_A filled, gBlock_B processed, gBlock_C sent
unsigned int gBlock_A[NUM_SAMPLES] ;
unsigned int gBlock_B[NUM_SAMPLES] ;
unsigned int gBlock_C[NUM_SAMPLES] ;

//Set up the TCBs to rotate automatically
int TCB_gBlock_A[4] = { 0, sizeof(gBlock_A), 1, 0};;
int TCB_gBlock_B[4] = { 0, sizeof(gBlock_B), 1, 0};
int TCB_gBlock_C[4] = { 0, sizeof(gBlock_C), 1, 0};

void InitSPORT()
{
    //Proceed from Block A to Block C
    TCB_gBlock_A[0] = (int) TCB_gBlock_C + 3 - OFFSET + PCI ;
    TCB_gBlock_A[3] = (unsigned int) gBlock_A - OFFSET ;

    //Proceed from Block B to Block A
    TCB_gBlock_B[0] = (int) TCB_gBlock_A + 3 - OFFSET + PCI ;
    TCB_gBlock_B[3] = (unsigned int) gBlock_B - OFFSET ;

    //Proceed from Block C to Block B
    TCB_gBlock_C[0] = (int) TCB_gBlock_B + 3 - OFFSET + PCI ;
    TCB_gBlock_C[3] = (unsigned int) gBlock_C - OFFSET ;

    //Clear the Mutlichannel control registers
    *pSPMCTL0 = 0;
    *pSPMCTL1 = 0;
    *pSPMCTL2 = 0;
    *pSPCTL0 = 0 ;
    *pSPCTL1 = 0 ;
    *pSPCTL2 = 0 ;

    // ============================================================

    //
```

```c
    // Configure SPORT 0 for input from ADC
    //
    // ------------------------------------------------------------


    *pSPCTL0 = (OPMODE | SLEN24 | SPEN_A | SCHEN_A | SDEN_A);

    // Enabling Chaining
    // Block A will be filled first
    *pCPSP0A = (unsigned int) TCB_gBlock_A - OFFSET + 3;

    // ============================================================
    //
    // Configure SPORTs 1 & 2 for output to DACs 1-4
    //
    // ------------------------------------------------------------

    #ifdef DAC1
    *pSPCTL1 = (SPTRAN | OPMODE | SLEN24 | SPEN_A | SCHEN_A | SDEN_A) ;
    // write to DAC1
    *pCPSP1A = (unsigned int) TCB_gBlock_C - OFFSET + 3;
    #endif

    #ifdef DAC2
    *pSPCTL1 |= (SPTRAN | OPMODE | SLEN24 | SPEN_B | SCHEN_B | SDEN_B) ;
    // write to DAC2
    *pCPSP1B = (unsigned int) TCB_gBlock_C - OFFSET + 3;
    #endif

    #ifdef DAC3
    *pSPCTL2 = (SPTRAN | OPMODE | SLEN24 | SPEN_A | SCHEN_A | SDEN_A) ;
    // write to DAC3
    *pCPSP2A = (unsigned int) TCB_gBlock_C - OFFSET + 3;
    #endif

    #ifdef DAC4
    *pSPCTL2 |= (SPTRAN | OPMODE | SLEN24 | SPEN_B | SCHEN_B | SDEN_B) ;
    // write to DAC4
    *pCPSP2B = (unsigned int) TCB_gBlock_C - OFFSET + 3;
```

```
    #endif
}


// ======================================================
// init SRU
// ======================================================

void InitSRU(){

// -------------------------------------------------------------------
//
//  MCLK: The output of the 12.288 MHz xtal is either directly connected to the
//        codec, but also connected to DAI_P06, or just to DAI_P17. This is
//        determined by switch SW3 For this example we route the MCLK into
//        DAI_P17 and supply the clock to the ADC via DAI_P06  by routing the
//        signal through the SRU.

//  Tie the pin buffer input LOW.
    SRU(LOW,DAI_PB17_I);

//  Tie the pin buffer enable input LOW
    SRU(LOW,PBEN17_I);

// -------------------------------------------------------------------
//
//  Connect the ADC: The codec drives a BCLK output to DAI pin 7, a LRCLK
//          (a.k.a. frame sync) to DAI pin 8 and data to DAI pin 5.
//
//          Connect the ADC to SPORT0, using data input A
//
//          All three lines are always inputs to the SHARC so tie the pin
//          buffer inputs and pin buffer enable inputs all low.


// -------------------------------------------------------------------
```

```
//  Connect the ADC to SPORT0, using data input A

    //  Clock in on pin 7
    SRU(DAI_PB07_O,SPORT0_CLK_I);

    //  Frame sync in on pin 8
    SRU(DAI_PB08_O,SPORT0_FS_I);

    //  Data in on pin 5
    SRU(DAI_PB05_O,SPORT0_DA_I);

// ---------------------------------------------------------------------------
//    Tie the pin buffer inputs LOW for DAI pins 5, 7 and 8.  Even though
//    these pins are inputs to the SHARC, tying unused pin buffer inputs
//    LOW is "good coding style" to eliminate the possibility of
//    termination artifacts internal to the IC.  Note that signal
//    integrity is degraded only with a few specific SRU combinations.
//    In practice, this occurs VERY rarely, and these connections are
//    typically unnecessary.


    SRU(LOW,DAI_PB05_I);
    SRU(LOW,DAI_PB07_I);
    SRU(LOW,DAI_PB08_I);

// ---------------------------------------------------------------------------
//  Tie the pin buffer enable inputs LOW for DAI pins 5, 6, 7 and 8 so
//  that they are always input pins.

    SRU(LOW,PBEN05_I);
    SRU(LOW,PBEN07_I);
    SRU(LOW,PBEN08_I);

// ---------------------------------------------------------------------------
//
//  Connect the DACs: The codec accepts a BCLK input from DAI pin 13 and
//          a LRCLK (a.k.a. frame sync) from DAI pin 14 and has four
//          serial data outputs to DAI pins 12, 11, 10 and 9
//
```

```
//          Connect DAC1 to SPORT1, using data output A
//          Connect DAC2 to SPORT1, using data output B
//          Connect DAC3 to SPORT2, using data output A
//          Connect DAC4 to SPORT2, using data output B
//
//          Connect the clock and frame sync inputs to SPORT1 and SPORT2
//          should come from the ADC on DAI pins 7 and 8, respectively
//
//          Connect the ADC BCLK and LRCLK back out to the DAC on DAI
//          pins 13 and 14, respectively.
//
//          All six DAC connections are always outputs from the SHARC
//          so tie the pin buffer enable inputs all high.
//

// --------------------------------------------------------------------
//  Connect the pin buffers to the SPORT data lines and ADC BCLK & LRCLK

    SRU(SPORT2_DB_O,DAI_PB09_I);
    SRU(SPORT2_DA_O,DAI_PB10_I);
    SRU(SPORT1_DB_O,DAI_PB11_I);
    SRU(SPORT1_DA_O,DAI_PB12_I);

// --------------------------------------------------------------------
//  Connect the clock and frame sync input from the ADC directly
//    to the output pins driving the DACs.

    SRU(DAI_PB07_O,DAI_PB13_I);
    SRU(DAI_PB08_O,DAI_PB14_I);
    SRU(DAI_PB17_O,DAI_PB06_I);

// --------------------------------------------------------------------
//  Connect the SPORT clocks and frame syncs to the clock and
//  frame sync from the SPDIF receiver

    SRU(DAI_PB07_O,SPORT1_CLK_I);
    SRU(DAI_PB07_O,SPORT2_CLK_I);
    SRU(DAI_PB08_O,SPORT1_FS_I);
    SRU(DAI_PB08_O,SPORT2_FS_I);

//
```

```
//  Tie the pin buffer enable inputs HIGH to make DAI pins 9-14 outputs.
    SRU(HIGH,PBEN06_I);
    SRU(HIGH,PBEN09_I);
    SRU(HIGH,PBEN10_I);
    SRU(HIGH,PBEN11_I);
    SRU(HIGH,PBEN12_I);
    SRU(HIGH,PBEN13_I);
    SRU(HIGH,PBEN14_I);
// ------------------------------------------------------------------------
// Route SPI signals to AD1835.

    SRU(SPI_MOSI_O,DPI_PB01_I)      //Connect MOSI to DPI PB1.
    SRU(DPI_PB02_O, SPI_MISO_I)     //Connect DPI PB2 to MISO.
    SRU(SPI_CLK_O, DPI_PB03_I)      //Connect SPI CLK to DPI PB3.
    SRU(SPI_FLG3_O, DPI_PB04_I)     //Connect SPI FLAG3 to DPI PB4.
// ------------------------------------------------------------------------
// Tie pin buffer enable from SPI peipherals to determine whether they are
// inputs or outputs

    SRU(SPI_MOSI_PBEN_O, DPI_PBEN01_I);
    SRU(SPI_MISO_PBEN_O, DPI_PBEN02_I);
    SRU(SPI_CLK_PBEN_O, DPI_PBEN03_I);
    SRU(SPI_FLG3_PBEN_O, DPI_PBEN04_I);

// ------------------------------------------------------------------------
// UART config
    SRU2(UART0_TX_O,DPI_PB09_I); // UART transmit signal is connected to DPI pin 9
    SRU2(HIGH,DPI_PBEN09_I);
    SRU2(DPI_PB10_O,UART0_RX_I); // connect the pin buffer output signal to the UART0 receive
    SRU2(LOW,DPI_PB10_I);
    SRU2(LOW,DPI_PBEN10_I);      // disables DPI pin10 as input
}

// =======================================================
// IRQ's
//
```

```
    ==========================================================
void SetupIRQ01 ()
{
    //Enable the pins as IRQ0 and IRQ1
    *pSYSCTL|= IRQ0EN | IRQ1EN;
    asm ("#include <def21369.h>") ;
    //Set the IRQ pins to be edge sensitive
    asm ("bit set mode2 IRQ0E;") ;
    asm ("bit set mode2 IRQ1E;") ;
}

void Irq0ISR (int i)
{
    int leftDAC4Vol, rightDAC4Vol ;

    // IRQ0 is used to decrease volume
    SetupSPI1835 () ;
    leftDAC4Vol = Get1835Register (RD | DACVOL_L4) ;
    rightDAC4Vol = Get1835Register (RD | DACVOL_R4) ;

    // Now decrease by a step size of 0x3F
    leftDAC4Vol -= 0x3F ;
    rightDAC4Vol -= 0x3F ;

    if (leftDAC4Vol > 0)
        Configure1835Register (WR | DACVOL_L4 |
           leftDAC4Vol) ;

    if (rightDAC4Vol > 0)
        Configure1835Register (WR | DACVOL_R4 |
           rightDAC4Vol) ;

    DisableSPI1835 () ;
}

void Irq1ISR (int i)
{
    int leftDAC4Vol, rightDAC4Vol ;

    // IRQ1 is used to decrease volume
    SetupSPI1835 () ;
    leftDAC4Vol = Get1835Register (RD | DACVOL_L4) ;
    rightDAC4Vol = Get1835Register (RD | DACVOL_R4) ;

    // Now decrease by a step size of 0x3F
    leftDAC4Vol += 0x3F ;
    rightDAC4Vol += 0x3F ;

    if (leftDAC4Vol < 0x3FF)
        Configure1835Register (WR | DACVOL_L4 |
           leftDAC4Vol) ;

    if (rightDAC4Vol < 0x3FF)
        Configure1835Register (WR | DACVOL_R4 |
```

```
            rightDAC4Vol) ;

    DisableSPI1835 () ;
}

// =======================================================
// UART
// =======================================================

// type for the low level receive
typedef struct SADSPuartRecv {
        unsigned int    val;    //!< the value
        int                             i;      //!< the index
        TFkt_ADSPuartCB cb;             //!< the callback function when we received 4 bytes of data
} TADSPuartRecv;

TADSPuartRecv gUARTrx;


// init ADSP uart
//          Bits per Second  -> 19200
//          Data Bits        -> 8
//          Parity           -> odd
//          Stop Bits        -> 2
//          Flow Control     -> None
void initUART(TFkt_ADSPuartCB cbRXFunction) {
        // Sets the Baud rate for UART0
        *pUART0LCR = UARTDLAB;  //enables access to Divisor register to set baud rate
//      *pUART0DLL = 0x1c;      //0x21c = 540 for divisor value and gives a baud rate of19200 for core clock 331.776MHz
//    *pUART0DLH = 0x02;

        *pUART0DLL = 0x38;      //1080 = 0x438 for divisor value and gives a baud rate of 9600 for core clock 331.776MHz
        *pUART0DLH = 0x04;



    // Configures UART0 LCR
//    *pUART0LCR = UARTWLS8|    // word length 8
//                 UARTPEN|     // parity enable ODD parity
//                 UARTSTB ;    // Two stop bits
```

```
        *pUART0LCR = UARTWLS8;  // 8Bit 1StopBit
           NoParity

    //enables UART0 in receive mode
    *pUART0RXCTL = UARTEN;
    //enables UART0 in core driven mode
    *pUART0TXCTL = UARTEN;

    // set rx callback function and the state machine
    gUARTrx.val = 0;
    gUARTrx.i = 3;
    gUARTrx.cb = cbRXFunction;
}

// UART isr
void UARTisr(int i) {
        unsigned int v;
        v = *pUART0RBR;
        // shift register
        gUARTrx.val <<= 8;
        gUARTrx.val |= v;
        if (gUARTrx.i) {
                gUARTrx.i--;
        } else {
                gUARTrx.cb(gUARTrx.val);
                gUARTrx.i = 3;
                gUARTrx.val = 0;
        }
}

// UART send
// the system encoding of int values is big endian
// the uart send stream put this out as little endian
// value: 32 - 0 Bit 44 33 22 11
// @ADSP: 11 22 33 44
// send: 44 33 22 11 = little endian
void sendUARTuint32Value(unsigned int v) {
        // 44
        // wait till the transmitter is ready
        while ((*pUART0LSR & UARTTHRE) == 0);
        // mask all other bytes out and send the lowest
           byte
        *pUART0THR = v & 0xFF;
        v >>= 8;

        // 33
        while ((*pUART0LSR & UARTTHRE) == 0);
        *pUART0THR = v & 0xFF;
        v >>= 8;

        // 22
        while ((*pUART0LSR & UARTTHRE) == 0);
        *pUART0THR = v & 0xFF;
        v >>= 8;

```

```
        // 11
        while ((*pUART0LSR & UARTTHRE) == 0);
        *pUART0THR = v & 0xFF;

        // wait till the transmitter is ready
        while ((*pUART0LSR & UARTTHRE) == 0);
}

void sendUARTFloatVector (float * pD, int amount) {
        uint32_t * pS = (uint32_t *) pD;
        while (amount) {
                sendUARTuint32Value(*pS);
                pS++;
                amount--;
        }
}

void sendUARTraw4ByteVector (uint32_t * pD, int amount) {
        while (amount) {
                sendUARTuint32Value(*pD);
                pD++;
                amount--;
        }
}


void sendUARTintVectorBigEndian (int32_t * pD, int amount) {
        uint32_t * pS = (uint32_t *) pD;
        unsigned int v;
        while (amount) {
                // rotate bytes
                v = (*pS & 0x000000FF) << 24;
                v |= (*pS & 0x0000FF00) << 8;
                v |= (*pS & 0x00FF0000) >> 8;
                v |= (*pS & 0xFF000000) >> 24;
                // send value
                sendUARTuint32Value(v);
                pS++;
                amount--;
        }
}

void sendUARTuintVectorBigEndian (uint32_t * pD, int amount) {
        uint32_t * pS = (uint32_t *) pD;
        unsigned int v;
        while (amount) {
                // rotate bytes
                v = (*pS & 0x000000FF) << 24;
                v |= (*pS & 0x0000FF00) << 8;
                v |= (*pS & 0x00FF0000) >> 8;
                v |= (*pS & 0xFF000000) >> 24;

```

```
                    // send value
                    sendUARTuint32Value(v);
                    pS++;
                    amount--;
            }
}

//
   ================================================================
// SPORT IRQs
//
   ================================================================

     //Pointer to the blocks

unsigned int *gpProcessBuffer[3] = {gBlock_A,gBlock_C,
   gBlock_B};

// Counter to choose which buffer to process
volatile int gProcessBufferCounter=2;
// Semaphore to indicate to main that a block is ready
   for processing
volatile int gProcessBufferReady=0;

void TalkThroughISR(int sig_int)
{
     //Increment the block pointer
     gProcessBufferCounter++;
     gProcessBufferCounter %= 3;

     gProcessBufferReady = 1;
}

//
   ================================================================
// LED func
//
   ================================================================

void LEDSRUinit () {
          // Init LED Ports
          SRU(LOW,DPI_PB06_I);
             // Connect GND to DPI_PB06 input (LED1)
          SRU(LOW,DPI_PB07_I);                      //
             Connect GND to DPI_PB07 input (LED2)
          SRU(LOW,DPI_PB08_I);                      //
             Connect GND to DPI_PB08 input (LED3)
          SRU(LOW,DPI_PB13_I);                      //
             Connect GND to DPI_PB13 input (LED4)
          SRU(LOW,DPI_PB14_I);                      //
             Connect GND to DPI_PB14 input (LED5)
```

```
        SRU(LOW,DAI_PB15_I);    // Connect GND to DAI_PB15 input (LED6)
        SRU(LOW,DAI_PB16_I);    // Connect GND to DAI_PB16 input (LED7)

        //Enabling the Buffer using the following sequence: High -> Output, Low -> Input

        SRU(HIGH,DPI_PBEN06_I);    // LED 1
        SRU(HIGH,DPI_PBEN07_I);    // LED 2
        SRU(HIGH,DPI_PBEN08_I);    // LED 3
        SRU(HIGH,DPI_PBEN13_I);    // LED 4
        SRU(HIGH,DPI_PBEN14_I);    // LED 5
        SRU(HIGH,PBEN15_I);            // LED 6
        SRU(HIGH,PBEN16_I);            // LED 7
}

#define set_LED_1 SRU(HIGH,DPI_PB06_I)
#define set_LED_2 SRU(HIGH,DPI_PB07_I)
#define set_LED_3 SRU(HIGH,DPI_PB08_I)
#define set_LED_4 SRU(HIGH,DPI_PB13_I)
#define set_LED_5 SRU(HIGH,DPI_PB14_I)
#define set_LED_6 SRU(HIGH,DPI_PB15_I)
#define set_LED_7 SRU(HIGH,DPI_PB16_I)

#define clear_LED_1 SRU(LOW,DPI_PB06_I)
#define clear_LED_2 SRU(LOW,DPI_PB07_I)
#define clear_LED_3 SRU(LOW,DPI_PB08_I)
#define clear_LED_4 SRU(LOW,DPI_PB13_I)
#define clear_LED_5 SRU(LOW,DPI_PB14_I)
#define clear_LED_6 SRU(LOW,DPI_PB15_I)
#define clear_LED_7 SRU(LOW,DPI_PB16_I)

// ==========================================================
// init HW
// ==========================================================

void initHW(TFkt_ADSPuartCB cbRXFunction) {

        // uart stuff
        *pPICR2 &= ~(0x3E0); //Sets the UART0 receive interrupt to P13

        *pPICR2 |= (0x13<<5);
```

```c


    //Initialize PLL to run at CCLK= 331.776 MHz & SDCLK
       = 165.888 MHz.
    //SDRAM is setup for use, but cannot be accessed
       until MSEN bit is enabled
    InitPLL_SDRAM();

    // Setting up IRQ0 and IRQ1
    SetupIRQ01() ;

    // Need to initialize DAI because the sport signals
       need to be routed
    InitSRU();

    // This function will configure the codec on the kit
    Init1835viaSPI();

    interrupt (SIG_SP0,TalkThroughISR);
    interrupt (SIG_IRQ0, Irq0ISR) ;
    interrupt (SIG_IRQ1, Irq1ISR) ;

        *pUART0LCR=0;
    *pUART0IER   = UARTRBFIE;    // enables UART0
       receive interrupt
        interrupt(SIG_P13,UARTisr);


    // init LEDs
    LEDSRUinit();

    // init UART
    initUART (cbRXFunction);
}

void startHW() {
    // Finally setup the sport to receive / transmit the
        data
    InitSPORT();
}


//
   =====================================================

// processing
//
   =====================================================


#ifndef INT24_MAX
        #define INT24_MIN (-16777215-1)
        #define INT24_MAX (16777215)
#endif
```

```
#define dAD1835_ChannelAmount (2)
#define dAD1835_leftChannelOffset (1)
#define dAD1835_rightChannelOffset (0)

#define dAD1835_ChannelFlag_left ('l')
#define dAD1835_ChannelFlag_right ('r')

// the adsp codec channel type
typedef struct SCodecChannel {
        unsigned int    size;                   //!<
           size of the channel
        char                    channelFlag;    //!< l=
           left r=right channel
} TCodecChannel;

// the codec channel list
typedef struct SCodecChannelList {
        TCodecChannel *         pCC;
        int                                     number;
} TCodecChannelList;

// find the channel struct by a given channel
inline TCodecChannel * ADSP_getChannel (int channel) {
        extern TCodecChannelList gADSPcodecChannels;
        if ((channel < 0) || (channel >=
           gADSPcodecChannels.number)) return NULL;
        return &gADSPcodecChannels.pCC[channel];
}

// wait for the sample frame
inline void ADSP_waitForSamples () {
        if (!gProcessBufferReady) {
                set_LED_1;
                while (!gProcessBufferReady) {

                };
                gProcessBufferReady = 0;
                clear_LED_1;
        }
}

// ADSP ADC 24Bit value format
// 011...11             +FS
// 0......0             0
// 111...11             -FS
//
// reads some samples from the input channel
void ADSP_readSamplesFromChannel (TCodecChannel * pIC,
   float * pBuffer, unsigned int amount, int
   waitForNewFrame) {
        // wait for new samples
        if (waitForNewFrame)    ADSP_waitForSamples ();

        // after that get the pointer to the buffer
```

```
	int * pBinSRC  = (int *) gpProcessBuffer[gProcessBufferCounter];
	unsigned int i;
	// point to the first sample
	if (dAD1835_ChannelFlag_right == pIC->channelFlag) {
		pBinSRC += dAD1835_rightChannelOffset;
	} else {
		pBinSRC += dAD1835_leftChannelOffset;
	}

//	#pragma SIMD_for
	for (i = 0; i < amount; i++) {
		*pBuffer = ((float) ((int)(*pBinSRC)<<8)) * (1.0/2147483648.0);
		pBuffer++;
		pBinSRC += dAD1835_ChannelAmount;
	}
}


// writes some samples to the output channel
void ADSP_writesSamplesToChannel (TCodecChannel * pOC, float * pBuffer, unsigned int amount) {
	int * pBinSRC = (int *) gpProcessBuffer[gProcessBufferCounter];
	unsigned int i;
	float tv;
	// point to the first sample
	if (dAD1835_ChannelFlag_right == pOC->channelFlag) {
		pBinSRC += dAD1835_rightChannelOffset;
	} else {
		pBinSRC += dAD1835_leftChannelOffset;
	}
//	#pragma SIMD_for
	for (i = 0; i < amount; i++) {
		*pBinSRC = ((int)(2147483648.0 * pBuffer[i]))>>8;
		pBinSRC += dAD1835_ChannelAmount;
	}
}

TCodecChannel gADSPcodecChannel []= {
		{NUM_SAMPLES/2,dAD1835_ChannelFlag_left},	// ADC left input
		{NUM_SAMPLES/2,dAD1835_ChannelFlag_left}	// DAC4 & DAC3 left output
};

TCodecChannelList gADSPcodecChannels = {
		gADSPcodecChannel,
		sizeof(gADSPcodecChannel) / sizeof(TCodecChannel)
};
```

3.1.2 ANSI C strings

includes:

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

code:

```
// --------------------------------------
// TAPstringVector
// --------------------------------------

// ap string
typedef struct SAPstring {
        char *  szTxt;  //!< pointer to char buffer
        size_t  maxLen; //!< length of the char buffer(without the zero at the end)
} TAPstring;

// vector of strings type
typedef struct SAPstringVector {
        TAPstring *             sv;             //!< string vector
        unsigned int    num;    //!< amount of strings at the vector
} TAPstringVector;

// creates a AP string
TAPstringVector * APstringVector_create (
                int             num             //!<(in) number of strings at the vector
        ) {
        TAPstringVector * pR;
        int i;

        pR = malloc(sizeof(TAPstringVector));
        if (!pR) return NULL;
        pR->sv = malloc(sizeof(TAPstring)*num);
        if (!pR->sv){
                free (pR);
                return NULL;
        }
        // init str
        TAPstring * ps = pR->sv;
        for (i = 0; i < num; i++) {
                ps->szTxt = NULL;
                ps->maxLen = 0;
                ps++;
```

```
        }
        pR->num = num;
        return pR;
}

// frees a AP string
void APstringVector_free (
                TAPstringVector *       pSV     //!<(in) pointer to the string vector
        ) {
        unsigned int i;
        if (pSV->sv) {
                TAPstring * ps = pSV->sv;
                for (i = 0; i < pSV->num; i++) {
                        if (ps->szTxt){
                                free(ps->szTxt);
                        }
                }
                free(pSV->sv);
        }
        free(pSV);
}

// resizes the amount of strings at the string vector
int APstringVector_resizeVector (
                TAPstringVector *       pSV,    //!<(in) pointer to the string vector
                int                     num             //!<(in) number of strings at the vector
        ) {
        if (pSV->num == num) return 0;
        unsigned int    i;
        TAPstring *     ps;

        if (pSV->sv) {
                ps = pSV->sv;
                for (i = 0; i < pSV->num; i++) {
                        if (ps->szTxt){
                                free(ps->szTxt);
                        }
                }
                free(pSV->sv);
        }
        pSV->sv = malloc(sizeof(TAPstring)*num);
        if (!pSV->sv){
                return -1;
        }
        // init str
        ps = pSV->sv;
        for (i = 0; i < num; i++) {
                ps->szTxt = NULL;
                ps->maxLen = 0;
                ps++;
        }
```

```
	pSV->num = num;
	return 0;
}

// resizes a string at the vector
int APstringVector_resize (
		TAPstringVector *	pSV,	//!<(in) pointer to the string vector
		int			index,	//!<(in) index of the string at the stringarray
		size_t			newLen	//!<(in) new length of the string
	) {
	TAPstring * ps = pSV->sv + index;
	if (ps->maxLen < newLen) {
		// allocate new string
		char * nsz = malloc(sizeof(char) * (newLen +1));
		if (!nsz) return -1;
		if (ps->szTxt) {
			// copy old string
			strcpy(nsz,ps->szTxt);
			// release old string
			free(ps->szTxt);
			ps->szTxt = NULL;
		} else {
			*nsz = 0;
		}
		// set the new string
		ps->szTxt =nsz;
		// remember the length
		ps->maxLen = newLen;
	} else {
		if(ps->szTxt) {
			// trail old string
			ps->szTxt[newLen] = 0;
		}
	}
	return 0;
}

// fills the string from an extern source
void APstringVector_fill(
		TAPstringVector *	pSV,	//!<(in) pointer to the string vector
		int			index,	//!<(in) index of the string at the stringarray
		int			iStart,	//!<(in) start index at the string
		int			iEnd,	//!<(in) end index of the string
```

```
                char *                                      pSource
                   //!<(in) source from where the chars
                   are copied
        ) {
        int i, imax;
        char * pDest = pSV->sv[index].szTxt;
        pDest += iStart;
        // clip str
        imax = pSV->sv[index].maxLen;
        if (iEnd > imax) {
                iEnd = imax;
        }

        for (i = iStart;i < iEnd; i++) {
                *pDest = *pSource;
                pDest++;
                pSource++;
        }
        *pDest = 0;
}


// concat string 2 with string 1
int APstringVector_concat(
                TAPstringVector *        pSV1,
                   //!<(in) pointer to the first string
                   vector
                int
                   indexSV1,        //!<(in) index of the
                    string at the stringarray
                TAPstringVector *        pSV2,
                   //!<(in) pointer to the second string
                    vector
                int
                   indexSV2         //!<(in) index of the
                    string at the stringarray
        ) {
        TAPstring * ps1 = pSV1->sv + indexSV1;
        TAPstring * ps2 = pSV2->sv + indexSV2;
        // get new size
        size_t sl1;
        size_t sl2;
        sl1 = ps1->szTxt ? strlen(ps1->szTxt) : 0;
        sl2 = ps2->szTxt ? strlen(ps2->szTxt) : 0;
        size_t newL = sl1 + sl2;
        // reallocate?
        if (ps1->maxLen < newL) {
                if (APstringVector_resize(pSV1, indexSV1
                   , newL)) return -1;
        }
        // copy string 2 at the end of string 1
        strcpy(ps1->szTxt + sl1,ps2->szTxt);

        return 0;
}
```

```
// assigns string 2 to string 1
int APstringVector_assign(
                TAPstringVector *       pSV1,       //!<(in) pointer to the first string vector
                int                     indexSV1,       //!<(in) index of the string at the stringarray
                TAPstringVector *       pSV2,       //!<(in) pointer to the second string vector
                int                     indexSV2        //!<(in) index of the string at the stringarray
        ) {
        TAPstring * ps1 = pSV1->sv + indexSV1;
        TAPstring * ps2 = pSV2->sv + indexSV2;
        // get new size
        size_t sl2 = strlen(ps2->szTxt);
        // reallocate?
        if (ps1->maxLen < sl2) {
                if (APstringVector_resize(pSV1, indexSV1, sl2)) return -1;
        }
        // copy string 2 at the end of string 1
        strcpy(ps1->szTxt,ps2->szTxt);

        return 0;
}


// print a floating point number into the string
void APstringVector_printFloat(
                TAPstringVector *       pSV,            //!<(in) pointer to the first string vector
                int                     indexSV,                    //!<(in) index of the string at the stringarray
                float                                           number                  //!<(in) number to be printed
        ) {
        TAPstring * ps = pSV->sv + indexSV;
        snprintf(ps->szTxt, ps->maxLen+1, "%f", (double)number);
}

// print a integer number into the string
void APstringVector_printInt(
                TAPstringVector *       pSV,            //!<(in) pointer to the first
```

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

3.1.3 AP client interface useing 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:

```
//
    ===========================================================

// socket functions to communicate via UDP
//
    ===========================================================

// written by Stefan Jaritz -> prefix stj
// part I: defines

enum eAdminMsgTypes {
        eAdminMsgType_login      = 'i',
        eAdminMsgType_logout     = 'o',
        eAdminMsgType_exit               = 'e',
        eAdminMsgType_ack                = 'a',
        eAdminMsgType_nack               = 'n'
};


typedef struct SstjSocket_addr {
        struct sockaddr_in      Addr;
        int                                     len;
```

```
} TstjSocket_addr;


typedef struct SstjSocket_loginMsg {
        uint8_t         msgID;
        uint16_t        uuid;
        uint16_t        dataOutPort;
        uint16_t        dataInPort;
} TstjSocket_loginMsg;

typedef uint8_t TstjSocket_loginRAWmsg [7];

// creates a addinfo struct with the address of the local
struct addrinfo * stjSocket_getLocalSocketAddress (uint16_t port);

// creates a UPD server
int stjSocket_createServer (
                uint16_t                        port,   //!<(in) port
                SOCKET *                        pS,             //!<(out) socket
                TstjSocket_addr *       pAI     //!<(out) address info
        );


// creates a UDP client witch connects to a local server
int stjSocket_createClient (
                uint16_t                        port,   //!<(in) port
                SOCKET *                        pS,             //!<(out) socket
                TstjSocket_addr *       pAI     //!<(out) address info
        );

// closes socket & address
void stjSocket_close (
                SOCKET                          S,              //!<(in) socket
                TstjSocket_addr *       pAI     //!<(in) address info
        );

// sends some data to an address
int stjSocket_send (
                SOCKET                          S,                      //!< socket
                void *                          pData,          //!< data to send
                int                                     dataSize,       //!< amount of bytes
                TstjSocket_addr *       pAddr
```

```c
                    //!< address
        );

// receives some data and filles the address with the
   sender
int stjSocket_recv (
                SOCKET                          S,
                                    //!< socket
                void *                          pData,
                            //!< data to send
                int
                   dataSize,        //!< amount of bytes
                TstjSocket_addr *       pAddr
                   //!< address
        );

//
   ===========================================================

// socket functions to communicate via UDP
//
   ===========================================================

// written by Stefan Jaritz -> prefix stj
// part II: implementation

// defines
#define dMaxHostNameChars (254)

/*
// creates a addinfo struct with the address of the
   local
struct addrinfo * stjSocket_getLocalSocketAddress (
   uint16_t port) {
        char                            szPath[
           dMaxHostNameChars] = "";
        char                            szDummyStr[255]
           = "";
        struct hostent *        pHostInfo;
        struct addrinfo         hints;
        struct addrinfo *       pAI;

        memset ( &hints, 0,sizeof(hints));
        hints.ai_family = AF_UNSPEC; // IPv4 and IPv6
        hints.ai_socktype = SOCK_DGRAM;
        hints.ai_protocol = 0; // any protocol
        hints.ai_flags = 0;

        // resolve hostname
    if (gethostname(szPath,dMaxHostNameChars)) goto
       mainErrorWithMsg;

        pHostInfo = gethostbyname(szPath);
        if (!pHostInfo) goto mainErrorWithMsg;

```

```
        sprintf(szDummyStr,"%u",(unsigned int)port);

        // build address info
        if (getaddrinfo (pHostInfo->h_name,szDummyStr,&hints,&pAI)) {
                fprintf (stderr,"getting address-information of the TCP port failed (error code %s)!\n",gai_strerror(WSAGetLastError()));
                goto mainError;
        }

        return pAI;

mainErrorWithMsg:
        fprintf (stderr, "error: %s\n",gai_strerror(WSAGetLastError()));
mainError:
    return NULL;

}
*/

// creates a UPD server
int stjSocket_createServer (
                uint16_t                        port,           //!<(in) port
                SOCKET *                        pS,             //!<(out) socket
                TstjSocket_addr *       pAI             //!<(out) address info
        ) {
        // create socket
        *pS=socket(AF_INET,SOCK_DGRAM,0);
        if (!*pS) goto mainErrorWithMsg;

        // create address
        pAI->Addr.sin_family=AF_INET; // AF_UNSPEC
        pAI->Addr.sin_port=htons(port);
        pAI->Addr.sin_addr.s_addr=ADDR_ANY;
        pAI->len = sizeof(SOCKADDR_IN);

        // bind socket to adress
        if (SOCKET_ERROR == bind(*pS,(SOCKADDR*)&(pAI->Addr),pAI->len)) goto mainErrorWithMsg;
    return 0;
mainErrorWithMsg:
        printf ("error: %s\n",gai_strerror(WSAGetLastError()));
        return -1;
}

// creates a UDP client witch connects to a local server
int stjSocket_createClient (
                uint16_t                        port,
```

```
                    //!<(in) port
                SOCKET *                        pS,
                    //!<(out) socket
                TstjSocket_addr *       pAI
                    //!<(out) address info
        ) {
        char                            szPath[dMaxHostNameChars] = "";
        struct hostent *        pHostInfo;

        // resolve hostname
    if (gethostname(szPath,dMaxHostNameChars)) goto mainErrorWithMsg;

        pHostInfo = gethostbyname(szPath);
        if (!pHostInfo) goto mainErrorWithMsg;

        // create socket
        *pS=socket(AF_INET,SOCK_DGRAM,0);
        if (!*pS) goto mainErrorWithMsg;

        // get the local ip from the host
        char * szLocalIP;
        szLocalIP = inet_ntoa (*(struct in_addr *)*pHostInfo->h_addr_list);

        unsigned long addr = inet_addr(szLocalIP);
        if ((INADDR_NONE == addr) || (INADDR_ANY == addr)) {
                closesocket(*pS);
                printf ("unknown inet address\nerror: %s\n",gai_strerror(WSAGetLastError()));
                return -2;
        }

        // create address
        pAI->Addr.sin_family=AF_INET; // AF_UNSPEC
        pAI->Addr.sin_port=htons(port);
        pAI->Addr.sin_addr.s_addr= addr;
        pAI->len = sizeof(SOCKADDR_IN);

    return 0;
mainErrorWithMsg:
        printf ("error: %s\n",gai_strerror(WSAGetLastError()));
        return -1;
}

// closes socket & address
void stjSocket_close (
                SOCKET                          S,
                    //!<(in) socket
                TstjSocket_addr *       pAI
                    //!<(in) address info
        ) {
```

```
        // pAI is self build so we don't need a free call
        //freeaddrinfo(gMsgServer.aiAdmin);
        closesocket(S);
}

// sends some data to an address
int stjSocket_send (
                SOCKET                          S,                      //!< socket
                void *                          pData,          //!< data to send
                int                             dataSize,       //!< amount of bytes
                TstjSocket_addr *       pAddr           //!< address
        ) {
        int n;
        n = sendto (S, (const char *) pData, dataSize, 0,(struct sockaddr *)&(pAddr->Addr), pAddr->len);
        if (n != dataSize) {
                fprintf (stderr,"sending data failed (%i bytes send)!(error: %s)!\n",n, gai_strerror(WSAGetLastError()));
                return -1;
        }
        return 0;
}

// receives some data and filles the address with the sender
int stjSocket_recv (
                SOCKET                          S,                      //!< socket
                void *                          pData,          //!< data to send
                int                             dataSize,       //!< amount of bytes
                TstjSocket_addr *       pAddr           //!< address
        ) {
        int n;
        n = recvfrom(S, (char *) pData, dataSize, 0,(struct sockaddr *)&(pAddr->Addr), &pAddr->len);
        if (n != dataSize) {
                fprintf (stderr,"receiving data failed (%i bytes received)!(error: %s)!\n",n,gai_strerror(WSAGetLastError()));
                return -1;
        }
        return 0;
}

```

```c
// ========================================
// a TCP/IP client running the RX at an
// own thread and use a callback-function
// to sign that some data are received
// ========================================
// part I: header defines

typedef int (* TpfktAPClientRecvCallback) (void * pUserData, uint16_t number, uint8_t * pData);

typedef struct SAPClient {
        uint16_t uuid;                  //!< uuid of the client

        SOCKET sAdmin;                  //!< admin socket
        uint16_t adminPort;             //!< admin port
        TstjSocket_addr aAdmin;         //!< socket address of the admin

        SOCKET sDataIn;                 //!< data in socket
        uint16_t dataInPort;            //!< port of the data in socket
        TstjSocket_addr aDataIn;        //!< socket address of the data port

        SOCKET sDataOut;                //!< data out socket
        uint16_t dataOutPort;   //!< port of the data out socket
        TstjSocket_addr aDataOut;       //!< socket address of the data port

        pthread_t recvThread;           //!< receive thread
        uint8_t * recvBuffer;           //!< temporaly receive buffer
        uint16_t recvBufferSize; //!< size in bytes of the receive buffer
        void * pUserData;               //!< user data (can be NULL)
        TpfktAPClientRecvCallback fktRecvCB;    //!< receive call back funtion
} TAPClient;
```

```

// creates an admin client
int APclient_create (
                TAPClient *                 pC,                 //!<(in/out) pointer to a client structure to be filled
                uint16_t                    adminPort,          //!<(in) the port of the admin server port
                uint16_t                    recvBufferSize,          //!<(in) size in bytes of the receive buffer
                void *                      pUserData,          //!<(in) user data
                TpfktAPClientRecvCallback       fktCB              //!<(in) callback function when receiving data
        );

//! free's the client
int APclient_close (
                TAPClient *             pC              //!<(in) pointer to a client structure
        );


int APclient_send (
                TAPClient *             pC,             //!<(in) pointer to a client structure
                uint16_t                num,            //!<(in) amount of bytes to send
                uint8_t *               pData           //!<(in) pointer to the data
        );

// ==========================================
// a TCP/IP client running the RX at an
// own thread and use a callback-function
// to sign that some data are received
// ==========================================
// part II: implementation


// =====================================
// pre defs
// =====================================

// the thread function for receiving data
void * APclient_RecvThread (void *);
```

```c


// ======================================
// functions
// ======================================


// creates an admin client
int APclient_create (
                TAPClient *           pC,                 //!<(in/out) pointer to a client structure to be filled
                uint16_t              adminPort,          //!<(in) the port of the admin server port
                uint16_t              recvBufferSize,     //!<(in) size in bytes of the receive buffer
                void *                pUserData,          //!<(in) user data
                TpfktAPClientRecvCallback       fktCB     //!<(in) callback function when receiving data
        ) {
        uint8_t         msgID;

        // 0. save the user data
        pC->pUserData = pUserData;

        // 1. create a socket to communicate with the admin port
        if (stjSocket_createClient(
                        adminPort,
                        &(pC->sAdmin),
                        &(pC->aAdmin)
                )) {
                return -1;
        }

        pC->adminPort = adminPort;
        // 2. setup
        // request login
        msgID = eAdminMsgType_login;
        if (stjSocket_send(pC->sAdmin,&msgID,1,&(pC->aAdmin))) goto sendFailed;
        // get the ports
        if (stjSocket_recv(pC->sAdmin,(char *)&msgID,1,&(pC->aAdmin))) goto receivedFailed;
        if (msgID != eAdminMsgType_login) {
                fprintf(stderr,"admin send wrong message back\n");
```

```
                goto abortAndError;
        }

        if (stjSocket_recv(pC->sAdmin,(char *)&(pC->uuid),2,&(pC->aAdmin))) goto receivedFailed;
        if (stjSocket_recv(pC->sAdmin,(char *)&(pC->dataInPort),2,&(pC->aAdmin))) goto receivedFailed;
        if (stjSocket_recv(pC->sAdmin,(char *)&(pC->dataOutPort),2,&(pC->aAdmin))) goto receivedFailed;


        printf("data in: %u\ndata out: %u\n",(unsigned int)pC->dataInPort,(unsigned int)pC->dataOutPort);

        // create data sockets
        if (stjSocket_createServer(
                            pC->dataInPort,
                            &(pC->sDataIn),
                            &(pC->aDataIn)
                )) goto DataSocketCreationError;

        // create recv buffer & threads
        pC->recvBuffer = malloc((size_t)recvBufferSize);
        if (!pC) goto abortAndError;
        pC->recvBufferSize = recvBufferSize;
        pC->fktRecvCB = fktCB;

        // now we can start the handling thread
        pthread_create(&pC->recvThread,NULL,APclient_RecvThread,pC);

        // send ack to server
        msgID = eAdminMsgType_ack;
        if (stjSocket_send(pC->sAdmin,&msgID,1,&(pC->aAdmin))) goto sendFailed;

        // wait till server is ready
        if (stjSocket_recv(pC->sAdmin,&msgID,1,&(pC->aAdmin))) goto receivedFailed;
        if (eAdminMsgType_ack != msgID) {
                fprintf(stderr,"admin error\n");
                return -10;
        }

        // now open the client data port
        if (stjSocket_createClient(
                            pC->dataOutPort,
                            &(pC->sDataOut),
                            &(pC->aDataOut)
                )) goto DataSocketCreationError;

        // some info
```

```c
    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);
    fflush(stdout);
        return 0;

abortAndError:
        msgID = eAdminMsgType_nack;
        stjSocket_send(pC->sAdmin,&msgID,1,&(pC->aAdmin));
        return -2;

sendFailed:
        fprintf(stderr,"sending to admin failed\n");
        fprintf (stderr,"error: %s\n",gai_strerror(WSAGetLastError()));
        return -3;

receivedFailed:
        fprintf(stderr,"receiving from admin failed\n");
        fprintf (stderr,"error: %s\n",gai_strerror(WSAGetLastError()));
        return -4;

DataSocketCreationError:
        fprintf(stderr,"creating data sockets failed\n");
        fprintf (stderr,"error: %s\n",gai_strerror(WSAGetLastError()));
        return -4;
}

//! free's the client
int APclient_close (
                TAPClient *             pC                  //!<(in/out) pointer to a client structure to be filled
        ) {
        // vars
        uint8_t                         msgID;

        // request logout
        msgID = eAdminMsgType_logout;
        if (stjSocket_send(pC->sAdmin,(char *)&msgID,1,&(pC->aAdmin))) goto sendFailed;
        if (stjSocket_send(pC->sAdmin,(char *)&(pC->uuid),2,&(pC->aAdmin))) goto sendFailed;
        // wait till server finished it
        if (stjSocket_recv(pC->sAdmin,(char *)&msgID,1,&(pC->aAdmin))) goto receivedFailed;
        if (msgID != eAdminMsgType_ack) {
                fprintf(stderr,"logout failed\n");
        }

```

```
	stjSocket_close(pC->sAdmin,&(pC->aAdmin));
	stjSocket_close(pC->sDataIn,&(pC->aDataIn));
	stjSocket_close(pC->sDataOut,&(pC->aDataOut));

	// wait till thread is gone
	pthread_join(pC->recvThread,NULL);


	return 0;
sendFailed:
	fprintf(stderr,"sending to admin failed\n");
	return -3;

receivedFailed:
	fprintf(stderr,"receiving from admin failed\n");
	return -4;
}

// the thread function for receiving data
void * APclient_RecvThread (
		void *  pArg
	) {
	uint16_t				msgLen;
	uint8_t *				pNB;

	TAPClient *				pC = pArg;

	// get data
	for (;;) {
		// read amount of data to be received
		if (stjSocket_recv(pC->sDataIn,&msgLen,2,&(pC->aDataIn))) goto recvError;
		// check if we have enough memory allocated at the buffer
		if (msgLen > pC->recvBufferSize) {
			pNB = realloc(pC->recvBuffer,msgLen);
			if (pNB) {
				pC->recvBuffer = pNB;
				pC->recvBufferSize = msgLen;
			} else {
				fprintf (stderr,"realloc memory failed");
				goto recvError;
			}
		}
		// transmit data
		if (stjSocket_recv(pC->sDataIn,pC->recvBuffer,msgLen,&(pC->aDataIn))) goto recvError;
		// and handle the data
		pC->fktRecvCB(pC->pUserData, msgLen, pC->recvBuffer);

```

```
        }
        pthread_exit((void *)0);
        return NULL;

recvError:
        // close data connections
        fprintf (stderr,"receiving data failed!");
        pthread_exit((void *)-3);
        return NULL;
}


int APclient_send (
                TAPClient *             pC,
                    //!<(in) pointer to a client
                    structure
                uint16_t                num,
                    //!<(in) amount of bytes to send
                uint8_t *               pData
                    //!<(in) pointer to the data
        ) {
        if (stjSocket_send(pC->sDataOut,&pC->uuid,2,&(pC
            ->aDataOut))) goto sendFailed;
        if (stjSocket_send(pC->sDataOut,&num,2,&(pC->
            aDataOut))) goto sendFailed;
        if (stjSocket_send(pC->sDataOut,pData,(int)num
            ,&(pC->aDataOut))) goto sendFailed;

        return 0;
sendFailed:
        fprintf(stderr,"sending data to server failed\n"
            );
        return -1;

}
```

3.1.4 MSP430-169STK

includes:

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

code:

```
//
   ========================================================

// MSP 430 API
```

```
//=====================================================

#define   dMSP430_LED1_ON          P3OUT &= ~BIT6
#define   dMSP430_LED1_OFF         P3OUT |= BIT6
#define   dMSP430_LED2_ON          P3OUT &= ~BIT7
#define   dMSP430_LED2_OFF         P3OUT |= BIT7
#define   dMSP430_B1               BIT5&P1IN        //B1 - P1.5
#define   dMSP430_B2               BIT6&P1IN        //B2 - P1.6
#define   dMSP430_B3               BIT7&P1IN        //B3 - P1.7
#define   dMSP430_E_HIGH           P4OUT |= BIT1
#define   dMSP430_E_LOW            P4OUT &= ~BIT1
#define   dMSP430_RS_HIGH          P4OUT |= BIT3
#define   dMSP430_RS_LOW           P4OUT &= ~BIT3
#define   dMSP430_LCD_Data         P4OUT
#define   dMSP430_LCD_LIGHT_ON     P4OUT |= BIT0
#define   dMSP430_LCD_LIGHT_OFF    P4OUT &= ~BIT0

#define   dMSP430_INPUT            0
#define   dMSP430_OUTPUT           0xff
#define   dMSP430_ON               1
#define   dMSP430_OFF              0
#define   dMSP430_BUF_SIZE         25

#define   dMSP430__100us          7                //7 cycles *12 + 20 = 104 / 104*1us = 104us

//NAND FLASH
#define   dMSP430_MAX_BLOCK_NUMB               1024

#define   dMSP430_TRANS_LDY              50
#define   dMSP430_WRITE_DLY              400
#define   dMSP430_ERASE_DLY              4000

#define   dMSP430_OUT_PORT               P5OUT
#define   dMSP430_IN_PORT                        P5IN
#define   dMSP430_IO_DIR                         P5DIR

#define   dMSP430__CE_ON                         P2OUT &= ~BIT0
#define   dMSP430__CE_OFF                        P2OUT |= BIT0
#define   dMSP430__RE_ON                         P2OUT &= ~BIT1
#define   dMSP430__RE_OFF                        P2OUT |= BIT1
#define   dMSP430__WE_ON                         P2OUT &= ~BIT2
#define   dMSP430__WE_OFF                        P2OUT |= BIT2

```

```
#define   dMSP430_ALE_ON                                  P2OUT |= BIT3
#define   dMSP430_ALE_OFF                                 P2OUT &= ~BIT3
#define   dMSP430_CLE_ON                                  P2OUT |= BIT4
#define   dMSP430_CLE_OFF                                 P2OUT &= ~BIT4

#define   dMSP430_R_B                                     P2IN & BIT7
#define   dMSP430_DALLAS                                  P2IN & BIT5

#define   dMSP430_READ_SPARE                              0x50
#define   dMSP430_READ_0                                  0x00
#define   dMSP430_READ_1                                  0x01
#define   dMSP430_READ_STATUS                             0x70

#define   dMSP430_WRITE_PAGE                              0x80
#define   dMSP430_WRITE_AKN                               0x10

#define   dMSP430_ERASE_BLOCK                             0x60
#define   dMSP430_ERASE_AKN                               0xD0

#define   dMSP430_DEV_ID                                  0x90

#define   dMSP430_SAMSUNG_ID                              0xECE6


// ========================================
// helper
// ========================================

// delay cpu cycles
void msp430_Delay (unsigned int cycles)
{
        unsigned char k;
        for (k=0 ; k != cycles; k++);   //20+a*12 cycles (for 1MHz)
}

// delay a given time
void msp430_DelayN100us(unsigned char n)
{
        unsigned char j;
        for (j=0; j!=n; ++j) msp430_Delay (dMSP430__100us);
}

// ========================================
// LCD
```

```
// ========================================

enum eMSP430_LCDcommands {
        eMSP430_LCDcom_clear            = 0x01,
        eMSP430_LCDcom_returnHome       = 0x02,
        eMSP430_LCDcom_entryMode        = 0x04,
        eMSP430_LCDcom_display          = 0x08,
        eMSP430_LCDcom_cursorDisplay= 0x10,
        eMSP430_LCDcom_function         = 0x20,
        eMSP430_LCDcom_setCGram         = 0x40,
        eMSP430_LCDcom_setDDram         = 0x80
};


void msp430_LCD_E()
{
        dMSP430_E_HIGH;             //toggle E for LCD
        _NOP();
        _NOP();
        dMSP430_E_LOW;
}

// sends a char to the display
void msp430_LCD_sendChar (unsigned char d)
{
        unsigned char temp;

        msp430_DelayN100us(5);                      //.5ms
        temp = d & 0xf0;                    //get upper nibble
        dMSP430_LCD_Data &= 0x0f;
        dMSP430_LCD_Data |= temp;
        dMSP430_RS_HIGH;                            //set LCD to data mode
        msp430_LCD_E();                              //toggle E for LCD
        temp = d & 0x0f;
        temp = temp << 4;                   //get down nibble
        dMSP430_LCD_Data &= 0x0f;
        dMSP430_LCD_Data |= temp;
        dMSP430_RS_HIGH;                            //set LCD to data mode
        msp430_LCD_E();                              //toggle E for LCD
}

// sends a command to the LCD controller
void msp430_LCD_sendCmd (unsigned char e)
{
        unsigned char temp;

        msp430_DelayN100us(10);                     //10ms
        temp = e & 0xf0;                    //get upper nibble
```

```
        dMSP430_LCD_Data &= 0x0f;
        dMSP430_LCD_Data |= temp;                   //send CMD to LCD
        dMSP430_RS_LOW;                             //set LCD to CMD mode
        msp430_LCD_E();                             // toggle E for LCD
        temp = e & 0x0f;
        temp = temp << 4;                   //get down nibble
        dMSP430_LCD_Data &= 0x0f;
        dMSP430_LCD_Data |= temp;
        dMSP430_RS_LOW;                             //set LCD to CMD mode
        msp430_LCD_E();                             // toggle E for LCD
}

// cmd clear
inline void msp430_LCD_cmdClear () {
        msp430_LCD_sendCmd(eMSP430_LCDcom_clear);
}

// cmd cur. home
inline void msp430_LCD_cmdCurHome () {
        msp430_LCD_sendCmd(eMSP430_LCDcom_returnHome);
}

// cmd entry mode (if cursor is shifted)
inline void msp430_LCD_cmdEntry (
                unsigned char incrFlag,     //!< if 1 increment cursor, else decrement
                unsigned char enable        //!< if set incr/decr is enabled
                ) {
        unsigned char cmd;

        cmd = eMSP430_LCDcom_entryMode;
        if (incrFlag) cmd |= 0x2;
        if (enable) cmd |= 0x1;
        msp430_LCD_sendCmd(cmd);
}

// cmd set display on, show cursor, flash cursor
inline void msp430_LCD_cmdDisplay (
                unsigned char displayOn,    //!< if 1 display is turned on
                unsigned char cursorOn,     //!< if 1 the cursor is set on
                unsigned char cursorFlashOn //!< if 1 the cursor flashes
                ) {
        unsigned char cmd;

        cmd = eMSP430_LCDcom_display;
```

```
        if (displayOn) cmd |= 0x4;
        if (cursorOn) cmd |= 0x2;
        if (cursorFlashOn) cmd |= 0x1;
        msp430_LCD_sendCmd(cmd);
}

// cmd shift cursor
inline void msp430_LCD_cmdShiftCursor (
                unsigned char leftFlag, //!< if 1 cursor is shift left
                unsigned char num               //!< number of shifts
                ) {
        unsigned char cmd,n;

        cmd = eMSP430_LCDcom_cursorDisplay;
        if (!leftFlag) cmd |= 0x4;
        for (n = 0; n < num; n++) {
                msp430_LCD_sendCmd(cmd);
        }
}

// cmd shift display
inline void msp430_LCD_cmdShiftDisplay (
                unsigned char leftFlag, //!< if 1 cursor is shift left
                unsigned char num               //!< number of shifts
                ) {
        unsigned char cmd,n;

        cmd = eMSP430_LCDcom_cursorDisplay | 0x8;
        if (!leftFlag) cmd |= 0x4;
        for (n = 0; n < num; n++) {
                msp430_LCD_sendCmd(cmd);
        }
}

// cmd shift display
inline void msp430_LCD_cmdFunction (
                unsigned char datamode,         //!< 0=4 bit 1=8bit mode
                unsigned char displayLines      //!< 0=1 line, 1=2 lines
                ) {
        unsigned char cmd;

        cmd = eMSP430_LCDcom_function;
        if (datamode) cmd |= 0x10;
        if (displayLines) cmd |= 0x08;
                msp430_LCD_sendCmd(cmd);
}


// init the LCD display
```

```
void msp430_LCD_init()
{
	dMSP430_RS_LOW;

	//Delay 100ms
	msp430_DelayN100us(250);
	msp430_DelayN100us(250);
	msp430_DelayN100us(250);
	msp430_DelayN100us(250);

	// setup
	dMSP430_LCD_Data |= BIT4 | BIT5;			//D7-D4 = 0011
	dMSP430_LCD_Data &= ~BIT6 & ~BIT7;

	msp430_LCD_E();						// toggle E for LCD
	msp430_DelayN100us(100);				//10 ms
	msp430_LCD_E();						// toggle E for LCD
	msp430_DelayN100us(100);				//10 ms
	msp430_LCD_E();						// toggle E for LCD
	msp430_DelayN100us(100);				//10 ms
	dMSP430_LCD_Data &= ~BIT4;				//D7-D4 = 0010
	msp430_LCD_E();						// toggle E for LCD

	msp430_LCD_cmdFunction(0,1);
	msp430_LCD_cmdDisplay(1,1,1);
	msp430_LCD_cmdClear();
}

// writes a string at the display
void msp430_LCD_print (
		unsigned char	x,
		unsigned char	y,
		char *			szStr
	) {
	msp430_LCD_cmdClear();
	msp430_LCD_cmdCurHome();

	while (*szStr) {
		msp430_LCD_sendChar(*szStr);
		szStr++;
	}
}


// =======================================
// UART
```

```
// ========================================

// type for the fifo structure
typedef struct Sstj32BitFIFO {
        uint32_t *              buffer;
        uint32_t *              pBufferEnd; // pointer with the end value of the buffer
        uint32_t *              pW;             // write pointer
        uint32_t *              pR;             // read pointer
        // write element
        unsigned int    eC;             // element counter
        // status
        unsigned int    num;    // amount of elements at the buffer
        unsigned int    numMax; // maximum of elements
} Sstj32BitFIFO;

// inits the fifo
inline int stjFIFO_init (
                Sstj32BitFIFO * pFIFO,
                int                     elements
        ) {
        pFIFO->buffer = malloc(elements*sizeof(uint32_t));
        if (!pFIFO->buffer) return -1;

        pFIFO->numMax = elements;

        // setup the fifo
        pFIFO->pBufferEnd = pFIFO->buffer + elements;
        pFIFO->pW = pFIFO->buffer;
        pFIFO->pR = pFIFO->buffer;

        pFIFO->eC = 3;

        pFIFO->num = 0;
        while (elements) {
                elements--;
                *pFIFO->pW = 0;
                pFIFO->pW++;
        }
        pFIFO->pW = pFIFO->buffer;
        return 0;
}

// frees the fifo
inline int stjFIFO_free (
                Sstj32BitFIFO * pFIFO
        ) {
        free (pFIFO->buffer);
        pFIFO->buffer = NULL;

```

```
		pFIFO->numMax = 0;
		return 0;
}


// writes a char to the buffer
inline void stjFIFO_writeChar(Sstj32BitFIFO * pFIFO, uint8_t d) {
		*pFIFO->pW |= d;
		// some checks

		// 1. have we written 4 bytes?
		if (pFIFO->eC) {
				// no - do some shift stuff
				pFIFO->eC--;
				*(pFIFO->pW) <<= 8;
		} else {
				pFIFO->eC = 3;
				// yes - set that we got a new element
				pFIFO->num++;
				pFIFO->pW++;
				if (pFIFO->pW == pFIFO->pBufferEnd) {
						pFIFO->pW = pFIFO->buffer;
				}
		}
}

// writes a char to the buffer
inline void stjFIFO_writeCharWithRotation(Sstj32BitFIFO * pFIFO, uint8_t d) {
		*pFIFO->pW |= (((uint32_t)d) << 24);
		// some checks

		// 1. have we written 4 bytes?
		if (pFIFO->eC) {
				// no - do some shift stuff
				pFIFO->eC--;
				*(pFIFO->pW) >>= 8;
		} else {
				pFIFO->eC = 3;
				// yes - set that we got a new element
				pFIFO->num++;
				pFIFO->pW++;
				if (pFIFO->pW == pFIFO->pBufferEnd) {
						pFIFO->pW = pFIFO->buffer;
				}
		}
}


// read a element(uint32_t) from the buffer
inline int stjFIFO_readElement (Sstj32BitFIFO * pFIFO, uint32_t * pRes) {
		// if there are no entries report error
		if (!pFIFO->num) return -1;
```

```
	// ok let's read a value
	*pRes = *pFIFO->pR;
	*pFIFO->pR = 0;
	pFIFO->pR++;
	pFIFO->num--;
	if (pFIFO->pR == pFIFO->pBufferEnd) {
		pFIFO->pR = pFIFO->buffer;
	}

	return 0;
}

Sstj32BitFIFO gMsp430_uartFIFO;

// init UART0 port
void msp430_UART_init(int fifoElements)
{

	P3SEL |= 0x30;                          // P3.4 = USART0 TXD, P3.5 = USART0 RXD
	P3DIR |= 0x10;                          // P3.4 output direction

	// setup UART0
	UCTL0 = CHAR;                           // 8-bit character

	UTCTL0 = SSEL1;                         // UCLK = XT2
	UBR00 = 0x41;                           // 8 000 000/9600
	UBR10 = 0x03;                           //
	UMCTL0 = 0x0;

	ME1 |= UTXE0 + URXE0;                   // Enabled USART0 TXD/RXD
	IE1 |= URXIE0;                          // Enabled USART0 RX interrupt

	// setup the fifo
	stjFIFO_init(&gMsp430_uartFIFO,fifoElements);
}

//  UART rx isq
#pragma vector=UART0RX_VECTOR
__interrupt void msp430_UART_RXisr (void)
{
	unsigned char v;
	_NOP();
	// save value
	v = RXBUF0;
	stjFIFO_writeChar(&gMsp430_uartFIFO, v);
}

// send a bytes via uart
```

```
void msp430_UART_send (unsigned char * pD, unsigned int amount)
{
	unsigned int c;

	for (c = 0; c < amount; c++) {
		// wait till tx buffer is ready
		while ((IFG1 & UTXIFG0) == 0);
		// copy data
		TXBUF0 = *pD;
		pD++;
	}
	// wait till transfer has finished
	while ((IFG1 & UTXIFG0) == 0);
}


// ========================================
// AD/DA converter
// ========================================

// init DA converter
void msp430_DAC_init ()
{
	DAC12_0CTL = DAC12SREF1 + /*DAC12RES + */ DAC12IR + DAC12AMP_7;				//Ve REF+, 8-bit resolution
	DAC12_1CTL = DAC12SREF1 + /*DAC12RES + */ DAC12IR + DAC12AMP_7;				//Ve REF+, 8-bit resolution

//	DAC12_0CTL = DAC12SREF1 + DAC12RES + DAC12IR + DAC12AMP_7;				//Ve REF+, 8-bit resolution
//	DAC12_1CTL = DAC12SREF1 + DAC12RES + DAC12IR + DAC12AMP_7;
}

// init AD converter
void msp430_ADC_init ()
{
	ADC12CTL0 = SHT0_0 + ADC12ON;			// Set sampling time, turn on ADC12
	ADC12CTL1 = SHP;				// Use sampling timer
	//ADC12IE = 0x01;				// Enable interrupt
	ADC12MCTL0 = SREF_7;				//VR+ = VeREF+ and VR = VREF/VeREF
	ADC12CTL0 |= ENC;				// Conversion enabled
	P6SEL = BIT5 + BIT4 + BIT3 + BIT2 + BIT1 + BIT0;		// P6.4 ADC option
```

```
            select
}

// set AD channel
void msp430_setADChannel (unsigned int channel)
{
        ADC12CTL0 &= ~ENC;
                                                    //disable
            convertion
        ADC12MCTL0 &= 0xfff8;
                                                 //clear
            select chanel bits
        ADC12MCTL0 |= channel;
                                                      //select
             chanel
        ADC12CTL0 |= ENC;
                                                     //enable
            convertion
        ADC12CTL0 |= ADC12SC;
                                                 //Sampling
            open
        while ((ADC12CTL1 & ADC12BUSY) != 0);
}

// ========================================
// NAND FLASH
// ========================================

//pull flash pins to inactive condition
void msp430_Flash_inactive() {
        dMSP430_IO_DIR=dMSP430_INPUT;          //IO is
            inputs
        dMSP430__CE_OFF;              //=1
        dMSP430__RE_OFF;              //=1
        dMSP430__WE_OFF;              //=1
        dMSP430_ALE_OFF;              //=0
        dMSP430_CLE_OFF;              //=0
}

// write a data byte to the flash
void msp430_Flash_writeByte (unsigned char d) {
        dMSP430_IO_DIR=dMSP430_OUTPUT;         //IO is
            outputs
        dMSP430__WE_ON;
        dMSP430_OUT_PORT=d;
        dMSP430__WE_OFF;              //latch data
}

// reads a byte from the flash
unsigned char msp430_Flash_readByte()
{
        unsigned char f;

        dMSP430_IO_DIR=dMSP430_INPUT;          //IO is
            inputs
```

```
        dMSP430__RE_ON;
        f=dMSP430_IN_PORT;
        dMSP430__RE_OFF;                    //read data
        return f;
}

// write a block to the flash
unsigned char msp430_Flash_write(
                unsigned char page,
                unsigned char colAddr,
        unsigned char rowAddLow,
        unsigned char rowAddHigh,
        unsigned char num,
        unsigned char * pD
        ) {
        unsigned char k, l;

        msp430_Flash_inactive();
        dMSP430_CLE_ON;
        dMSP430__CE_ON;
        msp430_Flash_writeByte(page);
        dMSP430_CLE_OFF;
        dMSP430_ALE_ON;
        msp430_Flash_writeByte(colAddr);
        msp430_Flash_writeByte(rowAddLow);
        msp430_Flash_writeByte(rowAddHigh);
        dMSP430_ALE_OFF;
        for (k=0; k != num; k++) {
                msp430_Flash_writeByte(*pD);
                pD++;
        }
        dMSP430_CLE_ON;
        msp430_Flash_writeByte(dMSP430_WRITE_AKN);
        while ((dMSP430_R_B) == 0);
        msp430_Flash_writeByte(dMSP430_READ_STATUS);
        dMSP430_CLE_OFF;
        l = msp430_Flash_readByte();
        msp430_Flash_inactive();
        return l;
}

// read a block of bytes from the flash
void msp430_Flash_read (
                unsigned char colAddr,
        unsigned char rowAddLow,
        unsigned char rowAddHigh,
        unsigned char num,
        unsigned char * pD
        )
{
        unsigned char n;

        msp430_Flash_inactive();
        dMSP430_CLE_ON;
        dMSP430__CE_ON;
```

```
        msp430_Flash_writeByte(dMSP430_READ_0);
        dMSP430_CLE_OFF;
        dMSP430_ALE_ON;
        msp430_Flash_writeByte(colAddr);
        msp430_Flash_writeByte(rowAddLow);
        msp430_Flash_writeByte(rowAddHigh);
        dMSP430_ALE_OFF;
        while ((dMSP430_R_B) == 0);
        for (n=0; n != num; n++) {
                *pD = msp430_Flash_readByte();
                pD++;
        }
        msp430_Flash_inactive();
}

unsigned char msp430_Flash_erase(
        unsigned char blockAddLow,
        unsigned char blockAddHigh
        ) {
        unsigned char m;

        msp430_Flash_inactive();
        dMSP430_CLE_ON;
        dMSP430__CE_ON;
        msp430_Flash_writeByte(dMSP430_ERASE_BLOCK);
        dMSP430_CLE_OFF;
        dMSP430_ALE_ON;
        msp430_Flash_writeByte(blockAddLow);
        msp430_Flash_writeByte(blockAddHigh);
        dMSP430_ALE_OFF;
        dMSP430_CLE_ON;
        msp430_Flash_writeByte(dMSP430_ERASE_AKN);
        while ((dMSP430_R_B) == 0);
        msp430_Flash_writeByte(dMSP430_READ_STATUS);
        dMSP430_CLE_OFF;
        m = msp430_Flash_readByte();
        msp430_Flash_inactive();
        return m;
}

// ========================================
// HW init
// ========================================

void msp430_start() {
        // Stop watchdog timer
        WDTCTL = WDTPW + WDTHOLD;

        //XT2-ON
        BCSCTL1 &= ~BIT7;
        //XT2 is SMCLK
        BCSCTL2 |= BIT3;

        //hardware init
        // 1. configure I/O Pins
```

```
        P1DIR=BIT0;

        //NAND FLASH init
        P2OUT=0x07;
        P2DIR=0x1F;

        //LED1
        P3OUT = BIT6 | BIT7;
        //LED2
        P3DIR = BIT6 | BIT7;

        //LCD init
        P4OUT = 0;
        P4DIR = 0xff;
}

void msp430_initHW (int fifoElements) {
        //configure modules

        //1 first the UART!
        msp430_UART_init(fifoElements);
        //2nd the LCD, if switched with UART the port
           sends a ghost sign because Port 3 is used by
           LCD and UART
        msp430_LCD_init();
        //3th DAC
        msp430_DAC_init();
        // 4th ADC
        msp430_ADC_init();

        // Enable interrupts
        _EINT();
}
```

3.1.5 audio dynamic processing (generic)

includes:

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

code:

```
// ========================================
// dynamics processing (DynProc)
// ========================================
// Literature:
// Udo Zlzer, Digitale Audiosignal Verarbeitung, 3.
   Auflage
// Udo Zlzer, DAFX - Digital Audio Effects

```

```
// short:
//              AT - attack time
//              RT - release time
//              TAV - average time
//              LT- limiter threshold
//              LS - limiter slope


// --------------------------------
// helper
// --------------------------------

float DynProc_calcSlope (
                float x1db, float y1db,
                float x2db, float y2db
        );

float DynProc_calcThreshold (
                float xDB,
                float xMax
        );

float DynProc_calcDB (float v);

float DynProc_calcValueFromDB (float dbVlaue, float xMax);

//! calc time parameter (AT, RT, TAV)
float DynProc_calcTimeParameter (
        float           Ta,     //!< (in) sampling period
        float           t               //!< (in) time parameter
        );

float DynProc_calcTimeFromTimeparameter (
                float Ta,
                float Tval
                );


// --------------------------------
// RMS
// --------------------------------
// RMS calc after Zlzer @p. 238

// structure for RMS calculation
typedef struct SDynProc_RMS {
        float   TAV;                    //!< time average
        float   oneMinusTAV;    //!< 1-TAV
        float   x2Old;                  //!< x(n-1)
} TDynProc_RMS;

```

```c
//! init rms
void DynProc_InitRMS (
        TDynProc_RMS *  pRMS,    //!< (in/out) structure to be filled
        float                   TAV             //!< (in) averaging time coefficient
        );

//! calc rms for one step (!rms = x!)
float DynProc_calcRMS (
                TDynProc_RMS *  pRMS,    //!< (in) structure to be filled
                float                   x       //!< (in) input signal witch should be rms
        );

// --------------------------------
// Peak
// --------------------------------
// Peak calc after Zlzer

// structure for Peak calculation
typedef struct SDynProc_Peak {
        float   AT;                     //!< attack time
        float   RT;                     //!< release time
        float   oneMinusAT;     //!< 1 - AT
        float   oneMinusRT;     //!< 1 - RT
        float   peak;           //!< peak
} TDynProc_Peak;


//! init peak
void DynProc_InitPeak (
        TDynProc_Peak * pPeak,  //!< (in/out) structure to be filled
        float                   AT,             //!< (in) attack time
        float                   RT              //!< (in) release time
        );

//! calc peak for one step
float DynProc_calcPeak (
                TDynProc_Peak * pPeak,  //!< (in) structure to be filled
                float                   x       //!< (in) input signal witch should be rms
        );


// --------------------------------
// smooth gain
```

```
// ---------------------------------
// smooth the gain value
// idea: use a hysteresis curve
//
// formula: g(n) = (1-k) * g(n - 1) + k * f(n)
//
// k = AT or k = RT
// k = (f(n) > f(n-1)) ? AT : RT
// if new value is over the old value -> attack
// else -> release

// structure for attack and release time adjustment calculation
typedef struct SDynProc_SmoothG {
        float                   AT;             //!< attack time
        float                   RT;             //!< release time
        float                   fOld;   //!< f(n-1)
        float                   gOld;   //!< g(n-1)
} TDynProc_SmoothG;

//! init AT/RT
void DynProc_InitSmoothG (
        TDynProc_SmoothG *      pSG,    //!< (in/out) structure to be filled
        float                           AT,     //!< (in) attack time
        float                           RT      //!< (in) release time
        );

//! init AT/RT
float DynProc_calcSmoothG (
        float                           fn,     //!< (in) input
        TDynProc_SmoothG *      pSG     //!< (in) structure to be filled
        );

// ---------------------------------
// limiter
// ---------------------------------
// LT - limiter treshold (where the limiter starts working)
// LS - limiter slope (how fast the limiter works)

// structure for dynamic range controler calculation
typedef struct SDynProc_Limiter {
        float                           LTlog;          //!< limiter threshold LT
        float                           LS;                     //!< limiter slope LS
        TDynProc_Peak           peakDetector;   //!< the peak detector
```

```
	TDynProc_SmoothG	SmoothG;	//!< AT/RT block
	float			delay;		//!< delay at the input - output lane
} TDynProc_Limiter;

//! init AT/RT
void DynProc_InitLimiter (
		TDynProc_Limiter *	pLim,		//!< (in/out) structure to be filled
		float			peakAT,		//!< (in) peak attack time
		float			peakRT,		//!< (in) peak release time
		float			smoothAT,	//!< (in) smoothing attack time
		float			smoothRT,	//!< (in) smoothing release time
		float			LT,		//!< (in) limiter threshold
		float			LS		//!< (in) limiter slope
);

//! init AT/RT
float DynProc_calcLimiter (
		TDynProc_Limiter *	pLim,	//!< (in) the limiter
		float			x		//!< (in) input signal
	);

// ---------------------------------
// compressor
// ---------------------------------
// CT - compressor treshold (where the compressor starts working)
// CS - compressor slope (how fast the compressor works)

// structure for dynamic range controller calculation
typedef struct SDynProc_Compressor {
	float			CTlog;		//!< compressor threshold
	float			CSlog;		//!< compressor slope
	TDynProc_RMS		RMS;		//!< RMS unit
	TDynProc_SmoothG	SmoothG;	//!< AT/RT block
	float			delay[2];	//!< delay at the input - output lane
```

```
} TDynProc_Compressor;



//! init compressor
void DynProc_InitCompressor (
        TDynProc_Compressor *   pComp,          //!< (in/out) structure to be filled
        float                                   rmsTAV,         //!< (in) rms time average coefficient
        float                                   smoothAT,       //!< (in) smoothing attack time
        float                                   smoothRT,       //!< (in) smoothing release time
        float                                   CT,             //!< (in) compressor treshold
        float                                   CS              //!< (in) compressor slope
);

//! calc compressor
float DynProc_calcCompressor (
                TDynProc_Compressor *   pComp,  //!< (in) the compressor
                float                   x               //!< (in) input signal
        );

// ---------------------------------
// expander
// ---------------------------------
// ET - expander threshold (where the expander starts working)
// ES - expander slope (how fast the expander works)

// structure for dynamic range controller calculation
typedef struct SDynProc_Expander {
        float                           ETlog;          //!< expander threshold log10 (ET)
        float                           ESlog;          //!< expander slope log10 (ES)
        TDynProc_RMS            RMS;            //!< RMS unit
        TDynProc_SmoothG        SmoothG;        //!< AT/RT block
        float                           delay[2];       //!< delay at the input - output lane
} TDynProc_Expander;

//! init expander
void DynProc_InitExpander (
        TDynProc_Expander *     pExp,           //!< (in/out) structure to be filled
```

```
        float                           rmsTAV,     //!< (in) rms time average coefficient
        float                           smoothAT,   //!< (in) smoothing attack time
        float                           smoothRT,   //!< (in) smoothing release time
        float                           ET,         //!< (in) expander treshold in dB
        float                           ES          //!< (in) expander slope in dB
);

//! calc expander
float DynProc_calcExpander (
                TDynProc_Expander *     pExp,   //!< (in) the expander
                float                   x               //!< (in) input signal
        );

// --------------------------------
// noisegate
// --------------------------------
// NT - noisegate threshold (till the noisegate works)

// structure for dynamic range controller calculation
typedef struct SDynProc_Noisegate {
        float                           NTlog;          //!< noisegate threshold log10 (NT)
        float                           NSlog;          //!< noisegate slope log10 (NS)
        TDynProc_RMS            RMS;            //!< RMS unit
        TDynProc_SmoothG        SmoothG;        //!< AT/RT block
        float                           delay[2];       //!< delay at the input - output lane
} TDynProc_Noisegate;

//! init noisegate
void DynProc_InitNoisegate (
        TDynProc_Noisegate *    pNG,            //!< (in/out) structure to be filled
        float                                   rmsTAV,         //!< (in) rms time average coefficient
        float                                   smoothAT,       //!< (in) smoothing attack time
        float                                   smoothRT,       //!< (in) smoothing release time
        float                                   NT,     //!< (in) noisegate threshold
        float                                   NS      //!< (in) noisegate slope
```

```
);

//! calc noisegate
float DynProc_calcNoisegate (
                TDynProc_Noisegate *    pNG,    //!< (in) the noisegate
                float                   x               //!< (in) input signal
        );

// ========================================
// dynamics processing (DynProc)
// ========================================
// Literature:
// Udo Zlzer, Digitale Audiosignal Verarbeitung, 3. Auflage
// Udo Zlzer, DAFX - Digital Audio Effects


// short:
//              AT - attack time
//              RT - release time
//              TAV - average time
//              LT- limiter threshold
//              LS - limiter slope

// it works in that way
// Y(n) = G(n) * x(n -D)
// g = {noisegate, expander, compressor, limiter}
// D - delay m - samples


// helper

float DynProc_calcSlope (
                float x1db, float y1db,
                float x2db, float y2db
        ) {
        // y = m * x + n; P1 & P2
        // m = (P2y - P1y) / (P2x - P1x)
        return (y2db - y1db) / (x2db - x1db);
}

float DynProc_calcThreshold (
                float xDB,
                float xMax
        ) {
        return xMax * powf (10.0f, xDB / 20.0f);
}

float DynProc_calcDB (float x) {
        return 10.0f * logf(x*x); // = 10.0 * log(x)  = 20.0f * log(x)
}
```

```
float DynProc_calcValueFromDB (float dbVlaue, float xMax) {
        return powf(10.0f, dbVlaue / 20.0) * xMax;
}


float DynProc_calcCompressionFactor (float slope) {
        return 1.f / (1.f- slope);
}

//! calc time parameter (AT, RT, TAV)
float DynProc_calcTimeParameter (
        float           Ta,      //!< (in) sampling period
        float           t               //!< (in) time parameter
        ) {
        // formular and 2.2 explained at "Digitale Audio Signalverarbeitung" @p.237
        return (1.0f - expf((-2.2f * Ta) / t));
}

float DynProc_calcTimeFromTimeparameter (
                float Ta,
                float Tval
                ) {
        return (-2.2f * Ta)/ (logf(1-Tval));
}

// ---------------------------------
// RMS
// ---------------------------------
// RMS calc after Zlzer @p. 238

//! init rms
void DynProc_InitRMS (
        TDynProc_RMS *  pRMS,   //!< (in/out) structure to be filled
        float                   TAV             //!< (in) averaging time coefficient
        ) {
        pRMS->TAV = TAV;
        pRMS->oneMinusTAV = 1.0f - TAV;
        pRMS->x2Old = 0.0f;
}

//! calc rms for one step
float DynProc_calcRMS (
                TDynProc_RMS *  pRMS,   //!< (in) structure to be filled
                float                   x                       //!< (in) input signal witch should be rms
        ) {
```

```
	float x2RMS;
	// calc forward
	x2RMS = pRMS->oneMinusTAV * pRMS->x2Old + pRMS->TAV * x * x;
	// calc backward
	pRMS->x2Old = x2RMS;
	return x2RMS;
}

// --------------------------------
// Peak
// --------------------------------
// Peak calc after Zlzer

//! init peak
void DynProc_InitPeak (
	TDynProc_Peak * pPeak,  //!< (in/out) structure to be filled
	float                   AT,             //!< (in) attack time
	float                   RT              //!< (in) release time
	) {
	pPeak->AT = AT;
	pPeak->RT = RT;
	pPeak->oneMinusAT = 1.0f - AT;
	pPeak->oneMinusRT = 1.0f - RT;
	pPeak->peak = 0.0f;
}

//! calc peak for one step
float DynProc_calcPeak (
		TDynProc_Peak * pPeak,  //!< (in) structure to be filled
		float                   x               //!< (in) input signal witch should be rms
	) {

	// formula from Zlzer @p. 235 wrong!
	// |x(n)| > xPeak(n-1) -> attack
	// |x(n)| <= xPeak(n-1) -> release
	// @attack: xpeak(n) = (1-AT) * xpeak(n-1) + AT * |x(n)|
	// @release: xpeak(n) = (1-RT) * peak(n-1)
	float a;
	a = fabsf(x);

	if (a > pPeak->peak) {
		pPeak->peak = pPeak->oneMinusAT * (pPeak->peak) + pPeak->AT * a;
	} else {
		pPeak->peak = pPeak->oneMinusRT * (pPeak->peak) + pPeak->RT * a;
	}
```

```
        return pPeak->peak;
}

// ---------------------------------
// smooth gain
// ---------------------------------
// smooth the gain value
// idea: use a hysteresis curve
//
// formula: g(n) = (1-k) * g(n - 1) + k * f(n)
//
// k = AT or k = RT
// k = (f(n) > f(n-1)) ? AT : RT
// if new value is over the old value -> attack
// else -> release

//! init AT/RT
void DynProc_InitSmoothG (
        TDynProc_SmoothG *      pSG,    //!< (in/out) structure to be filled
        float                   AT,     //!< (in) attack time
        float                   RT      //!< (in) release time
        ) {
        pSG->AT = AT;
        pSG->RT = RT;
        pSG->fOld = 0.0f;
        pSG->gOld = 0.0f;
}

//! init AT/RT
float DynProc_calcSmoothG (
        float                   fn,     //!< (in) input
        TDynProc_SmoothG *      pSG     //!< (in) structure to be filled
        ) {
        float k;
        float gn;

        // attack or release
        k = (fn > pSG->fOld) ? pSG->AT : pSG->RT;

        // calc gain
        gn = (1.0f-k) * pSG->gOld + k * fn;
        pSG->gOld = gn;
        pSG->fOld = fn;
        return gn;
}

// ---------------------------------
// limiter
// ---------------------------------
// LT - limiter treshold (where the limiter starts
```

```
    working)
// LS - limiter slope (how fast the limiter works)

//! init AT/RT
void DynProc_InitLimiter (
        TDynProc_Limiter *      pLim,           //!< (in
           /out) structure to be filled
        float                           peakAT,
           //!< (in) peak attack time
        float                           peakRT,
           //!< (in) peak release time
        float                           smoothAT,
           //!< (in) smoothing attack time
        float                           smoothRT,
           //!< (in) smoothing release time
        float                           LT,
                //!< (in) limiter threshold
        float                           LS
                //!< (in) limiter slope
) {
        DynProc_InitPeak(&(pLim->peakDetector),peakAT,
           peakRT);
        DynProc_InitSmoothG(&(pLim->SmoothG),smoothAT,
           smoothRT);
        pLim->LTlog = log10f(LT);
        pLim->LS = LS;
        pLim->delay = 0.0f;
}

//! init AT/RT
float DynProc_calcLimiter (
                TDynProc_Limiter *      pLim,   //!< (in
                   ) the limiter
                float                           x
                        //!< (in) input signal
        ) {
        float y;
        float g;

        // 1.) calc direct feedthrough lane
        y = pLim->delay;
        // 1.1.) recalc delay
        pLim->delay = x;

        // 2.) calc limiter gain
        // 2.1) peak detect
        g = DynProc_calcPeak(&(pLim->peakDetector),x);
        g = log10f(g);
        // 2.3. if we are over the threshold
        if (g > pLim->LTlog) {
                g = -pLim->LS * (g - pLim->LTlog);
        } else {
                g = 0.0;
        }
        g = powf (10.0f, g);
```

```
        g = DynProc_calcSmoothG(g, &(pLim->SmoothG));
        // 3. calc result of delay mul compressor gain
        y *= g;

        // AND return
        return y;
}

// --------------------------------
// compressor
// --------------------------------
// CT - compressor treshold (where the compressor starts working)
// CS - compressor slope (how fast the compressor works)

//! init compressor
void DynProc_InitCompressor (
        TDynProc_Compressor *   pComp,          //!< (in/out) structure to be filled
        float                                   rmsTAV,         //!< (in) rms time average coefficient
        float                                   smoothAT,       //!< (in) smoothing attack time
        float                                   smoothRT,       //!< (in) smoothing release time
        float                                   CT,             //!< (in) compressor treshold
        float                                   CS              //!< (in) compressor slope
) {
        DynProc_InitRMS(&(pComp->RMS),rmsTAV);
        DynProc_InitSmoothG(&(pComp->SmoothG),smoothAT,smoothRT);
        pComp->CTlog = DynProc_calcDB(CT);
        pComp->CSlog = -CS;
        pComp->delay[0] = 0.0f;
        pComp->delay[1] = 0.0f;
}

//! calc compressor
float DynProc_calcCompressor (
                TDynProc_Compressor *   pComp,  //!< (in) the compressor
                float                   x               //!< (in) input signal
        ) {
        float y;
        float g;

        // 1.) calc direct feedthrough lane
        y = pComp->delay[1];
        // 1.1.) recalc delay
        pComp->delay[1] = pComp->delay[0];
```

```
        pComp->delay[0] = x;

        // 2.) calc compressor gain
        // 2.1) peak detect
        g = DynProc_calcRMS(&(pComp->RMS),x);
        // 2.2) lin/log
        g = log10f(g);
        g *= 10.0f; // rms = x - > log(x) = 2 log(x) -> Xdb = 20*log(x) -> 20 / 2 = 10

        // 2.3. if we are over the threshold
        if (g > pComp->CTlog) {
                g = pComp->CSlog*(g-pComp->CTlog) / 20.0f;
        } else {
                g = 0.0f;
        }
        g = powf (10.0f, g);
        g = DynProc_calcSmoothG(g, &(pComp->SmoothG));
        // 3. calc result of delay mul compressor gain
        y *= g;

        // AND return
        return y;
}


// ---------------------------------
// expander
// ---------------------------------
// ET - expander threshold (where the expander starts working)
// ES - expander slope (how fast the expander works)

//! init expander
void DynProc_InitExpander (
        TDynProc_Expander *     pExp,           //!< (in/out) structure to be filled
        float                                   rmsTAV,         //!< (in) rms time average coefficient
        float                                   smoothAT,       //!< (in) smoothing attack time
        float                                   smoothRT,       //!< (in) smoothing release time
        float                                   ET,             //!< (in) expander treshold in dB
        float                                   ES              //!< (in) expander slope in dB
) {
        DynProc_InitRMS(&(pExp->RMS),rmsTAV);
        DynProc_InitSmoothG(&(pExp->SmoothG),smoothAT,smoothRT);
        pExp->ETlog = DynProc_calcDB(ET);
        pExp->ESlog = ES;
        pExp->delay[0] = 0.0f;
```

```
        pExp->delay[1] = 0.0f;
}

//! calc expander
float DynProc_calcExpander (
                TDynProc_Expander *             pExp,
                   //!< (in) the expander
                float
                   x                //!< (in) input
                   signal
        ) {
        float y;
        float g;

        // 1.) calc direct feedthrough lane
        y = pExp->delay[1];
        // 1.1.) recalc delay
        pExp->delay[1] = pExp->delay[0];
        pExp->delay[0] = x;

        // 2.) calc compressor gain
        // 2.1) peak detect
        g = DynProc_calcRMS(&(pExp->RMS),x);
        // 2.2) lin/log
        g = log10f(g);
        g *= 10.0f; // rms = x - > log(x) = 2 log(x) ->
           Xdb = 20*log(x) -> 20 / 2 = 10

        // 2.3. if we are over the threshold
        if (g > pExp->ETlog) {
                g = pExp->ESlog*(pExp->ETlog-g) / 20.0;
        } else {
                g = 0.0f;
        }
        g = powf (10.0f, g);
        g = DynProc_calcSmoothG(g, &(pExp->SmoothG));
        // 3. calc result of delay mul compressor gain
        y *= g;

        // AND return
        return y;
}


// --------------------------------
// noisegate
// --------------------------------
// NT - noisegate threshold (till the noisegate works)

//! init noisegate
void DynProc_InitNoisegate (
        TDynProc_Noisegate *    pNG,             //!< (in
           /out) structure to be filled
        float                                    rmsTAV,
                   //!< (in) rms time average
```

```
coefficient
        float                                                smoothAT,          //!< (in) smoothing attack time
        float                                                smoothRT,          //!< (in) smoothing release time
        float                                                NT,                //!< (in) noisegate threshold
        float                                                NS                 //!< (in) noisegate slope
) {
        DynProc_InitRMS(&(pNG->RMS),rmsTAV);
        DynProc_InitSmoothG(&(pNG->SmoothG),smoothAT,smoothRT);
        pNG->NTlog = DynProc_calcDB(NT);
        pNG->NSlog = NS;
        pNG->delay[0] = 0.0f;
        pNG->delay[1] = 0.0f;
}

//! calc noisegate
float DynProc_calcNoisegate (
                TDynProc_Noisegate *    pNG,     //!< (in) the noisegate
                float                   x                //!< (in) input signal
        ) {
        float y;
        float g;

        // 1.) calc direct feedthrough lane
        y = pNG->delay[1];
        // 1.1.) recalc delay
        pNG->delay[1] = pNG->delay[0];
        pNG->delay[0] = x;

        // 2.) calc noisegate gain
        // 2.1) rms detect
        g = DynProc_calcRMS(&(pNG->RMS), x);
        // 2.2) lin/log
        g = log10f(g);
        g *= 10.0f; // rms = x - > log(x) = 2 log(x) -> Xdb = 20*log(x) -> 20 / 2 = 10

        // 2.3. if we are over the threshold
        if (g > pNG->NTlog) {
                // calc linear function at log space
                g = pNG->NSlog*(g-pNG->NTlog) / 20.0;
                g = powf(10.f, g);
        } else {
                g = 0.0f;
        }
        // and smooth
        g = DynProc_calcSmoothG(g, &(pNG->SmoothG));
        // 3. calc result of delay mul noisegate gain
```

```
        y *= g;

        // AND return
        return y;
}
```

3.1.6 biquad filters (generic)

includes:

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

code:

```
// ================================================
// a generic biquad filter lib based on the math.h
// part I - defines
// ================================================

// no print functions (they are only for debug purposes)
//#define dBQF_implementPrintFunctions

// the biquad structure at the direct form II
typedef struct SBQF_BiquadDF2 {
        float   k;      // gain
        float   n1;     // numerator
        float   n2;
        float   d1; // denominator
        float   d2;
        float   s1; // delays
        float   s2;
} TBQF_BiquadDF2;

// a cascade of biquad filters
typedef struct SBQF_BiquadCascade {
        int                                         num;
        TBQF_BiquadDF2 *        pB;
} TBQF_BiquadCascade;

//! creates a cascade of biquad filters
int BQF_BQFcascadeCreate (
                TBQF_BiquadCascade *    pBQC,   //!< (in
                   /out) pointer to a struct to be
                   filled
                int
                   num              //!< amount of
                   filters
        );

//! deletes a cascade of biquad filters
```

```
void BQF_BQFcascadeDelete (
                    TBQF_BiquadCascade *    pBQC    //!< (in) pointer to a struct to be freed
          );

//! convolut the input with the biquad filter cascade and generate the output (y = BQFc * x)
void BQF_BQFcascadeConvolute (
                    TBQF_BiquadCascade *    pBQC,   //!< (in) the biquad cascade
                    float *                 px,     //!< (in) input signal vector
                    unsigned int            xNum,   //!< (in) amount of elements at the input vector
                    float *                 py      //!< (out) output signal vector (size must be more or equal to px)
          );

//! Initializes a element of the biquad cascade as high-pass(HP) filter
int BQF_BQFcascadeInitHP (
                    TBQF_BiquadCascade *    pBQC,   //!< (in/out) the biquad cascade
                    unsigned int            index,  //!< (in) the index of the filter witch should be used as HP
                    float                   fs,     //!< (in) sample frequency
                    float                   fc      //!< (in) cut off frequency
          );

//! initializes a element of the biquad cascade as low-pass(LP) filter
int BQF_BQFcascadeInitLP (
                    TBQF_BiquadCascade *    pBQC,   //!< (in/out) the biquad cascade
                    unsigned int            index,  //!< (in) the index of the filter witch should be used as LP
                    float                   fs,     //!< (in) sample frequency
                    float                   fc      //!< (in) cut off frequency
          );

//! initializes a element of the biquad cascade as peak
```

```
   boost/cut filter
int BQF_BQFcascadeInitPeak (
                TBQF_BiquadCascade *    pBQC,   //!< (in/out) the biquad cascade
                unsigned int                    index,   //!< (in) the index of the filter witch should be used as peak filter
                float   fs,             //!< (in) sample frequency
                float   fc,             //!< (in) center frequency
                float   q,              //!< (in) quality (Q infinty)
                float   g               //!< (in) gain
        );

//! initializes a element of the biquad cascade as low frequency shelving filter
int BQF_BQFcascadeInitLowFreqShelving (
                TBQF_BiquadCascade *    pBQC,   //!< (in/out) the biquad cascade
                unsigned int                    index,   //!< (in) the index of the filter witch should be used as peak filter
                float   fs,             //!< (in) sample frequency
                float   f,              //!< (in) cut/boost frequency
                float   q,              //!< (in) quality (Q infinty)
                float   g               //!< (in) gain
        );

//! initializes a element of the biquad cascade as high frequency shelving filter
int BQF_BQFcascadeInitHighFreqShelving (
                TBQF_BiquadCascade *    pBQC,   //!< (in/out) the biquad cascade
                unsigned int                    index,   //!< (in) the index of the filter witch should be used as peak filter
                float   fs,             //!< (in) sample frequency
                float   f,              //!< (in) cut/boost frequency
```

```
                float
                   q,              //!< (in) quality (Q
                   infinty)
                float
                   g               //!< (in) gain
        );

//! initializes a element of the biquad cascade from a
   array with the coefficients
void BQF_BQFinitFromCoefficents (
                TBQF_BiquadCascade *    pBQC,
                           //!< (in/out) the biquad
                   cascade
                int
                           index,                   //!<
                   (in) the index of the filter witch
                   should be used as peak filter
                float
                   numerator [3],  //!< (in) n0 to n2
                float
                   denominator [3] //!< (in) d0 to d2
        );

//! gets from an element of the biquad cascade the
   coefficients
void BQF_BQFgetNumAndDenom (
                TBQF_BiquadCascade *    pBQC,
                           //!< (in/out) the biquad
                   cascade
                int
                           index,                   //!<
                   (in) the index of the filter witch
                   should be used as peak filter
                float *
                   pNumerator,             //!< (in) n0
                   to n2
                float *
                   pDenominator    //!< (in) d0 to d2
        );

#ifdef dBQF_implementPrintFunctions
// a simple print function (k, n1, n2, d1, d2)
void BQF_printBiquadDF2(TBQF_BiquadCascade *pBC, FILE *
   stream);

// a simple print function (n0, n1, n2, d0, d1, d2)
void BQF_PrintBiquad(TBQF_BiquadCascade *pBC, FILE *
   stream);
#endif

// =================================================
// a generic biquad filter lib based on the math.h
// part II - implementations
// =================================================

```

```
// math defines
#ifndef M_E
	#define M_E            2.7182818284590452354
#endif

#ifndef M_LOG2E
	#define M_LOG2E        1.4426950408889634074
#endif

#ifndef M_LOG10E
	#define M_LOG10E       0.43429448190325182765
#endif

#ifndef M_LN2
	#define M_LN2          0.69314718055994530942
#endif

#ifndef M_LN10
	#define M_LN10         2.30258509299404568402
#endif

#ifndef M_PI
	#define M_PI           3.14159265358979323846
#endif

#ifndef M_PI_2
	#define M_PI_2         1.57079632679489661923
#endif

#ifndef M_PI_4
	#define M_PI_4         0.78539816339744830962
#endif

#ifndef M_1_PI
	#define M_1_PI         0.31830988618379067154
#endif

#ifndef M_2_PI
	#define M_2_PI         0.63661977236758134308
#endif

#ifndef M_2_SQRTPI
	#define M_2_SQRTPI     1.12837916709551257390
#endif

#ifndef M_SQRT2
	#define M_SQRT2        1.41421356237309504880
#endif

#ifndef M_SQRT1_2
	#define M_SQRT1_2      0.70710678118654752440
#endif
```

```
//! creates a cascade of biquad filters
int BQF_BQFcascadeCreate (
                TBQF_BiquadCascade *    pBQC,   //!< (in/out) pointer to a struct to be filled
                int                     num             //!< amount of filters
        ) {
        int i;

        // alloc / realloc the biquad
        if (pBQC->num != num) {
                if (pBQC->pB) {
                        free(pBQC->pB);
                }
                pBQC->pB = malloc (sizeof(TBQF_BiquadDF2) * num);
                if (!pBQC->pB) {
                        pBQC->num = 0;
                        return -1;
                }
                pBQC->num = num;
        }

        // init data fields
        for (i = 0; i < num; i++) {
                pBQC->pB[i].k = 0.0f;
                pBQC->pB[i].n1 = 0.0f;
                pBQC->pB[i].n2 = 0.0f;
                pBQC->pB[i].d1 = 0.0f;
                pBQC->pB[i].d2 = 0.0f;
                pBQC->pB[i].s1 = 0.0f;
                pBQC->pB[i].s2 = 0.0f;
        }
        return 0;
}

//! deletes a cascade of biquad filters
void BQF_BQFcascadeDelete (
                TBQF_BiquadCascade *    pBQC    //!< (in) pointer to a struct to be freed
        ) {
        free (pBQC->pB);
        pBQC->pB = NULL;
        pBQC->num = 0;
}

//! convolut the input with the biquad filter cascade and generate the output (y = BQFc * x)
void BQF_BQFcascadeConvolute (
                TBQF_BiquadCascade *    pBQC,   //!< (in) the biqaud cascade
                float *                 px,             //!< (in) input
```

```
                        signal vector
                    unsigned int                                xNum,   //!< (in) amount of elements at the input vector
                    float *     py                  //!< (out) output signal vector (size must be more or equal to px)
        ) {
        int                                 i, j;
        float                               d1, y;
        TBQF_BiquadDF2 *         pF;

        for (i = 0; i < xNum; i++) {
                y = *px;
                pF = pBQC->pB;
                for (j = 0; j < pBQC->num; j++) {
                        // filter
                        d1 = - (pF->d2 * pF->s2 + pF->d1 * pF->s1 + y); // y = x
                        y = pF->n2 * pF->s2 + pF->n1 * pF->s1 +  d1;
                        y *= pF->k;

                        pF->s2 = pF->s1;
                        pF->s1 = d1;
                        // output is input for the next filter
                        pF++;
                }

                // save result to the output
                *py = y;
                px++;
                py++;
        }
}

//! initializes a element of the biquad cascade as high-pass(HP) filter
int BQF_BQFcascadeInitHP (
                TBQF_BiquadCascade *    pBQC,   //!< (in/out) the biqaud cascade
                unsigned int                                index,  //!< (in) the index of the filter witch should be used as HP
                float       fs,                 //!< (in) sample frequency
                float       fc                  //!< (in) cut off frequency
        ) {
        if (index >= pBQC->num) {
                return -1;
```

```
	}
	float k=tan(M_PI*fc/fs);
	float k2 = k * k;
	float sqrtTwo = M_SQRT2;
	float dn = (1+sqrtTwo*k+k2);

	float numerator[3];
	float denumerator[3];

	numerator[0]=1/dn;
	numerator[1]=-2/dn;
	numerator[2]=1/dn;
	denumerator[0]=1;
	denumerator[1]=(2*(k2-1))/dn;
	denumerator[2]=(1-sqrtTwo*k+k2)/dn;

	BQF_BQFinitFromCoefficents(pBQC,index, numerator, denumerator);

	return 0;
}

//! initializes a element of the biquad cascade as low-pass(LP) filter
int BQF_BQFcascadeInitLP (
		TBQF_BiquadCascade *	pBQC,	//!< (in/out) the biqaud cascade
		unsigned int			index,	//!< (in) the index of the filter witch should be used as LP
		float	fs,		//!< (in) sample frequency
		float	fc		//!< (in) cut off frequency
	) {
	if (index >= pBQC->num) {
		return -1;
	}
	float k=tan(M_PI*fc/fs);
	float k2 = k * k;
	float sqrtTwo = M_SQRT2;
	float dn = (1+sqrtTwo*k+k2);

	float numerator[3];
	float denumerator[3];

	numerator[0]=k2/dn;
	numerator[1]=2*k2/dn;
	numerator[2]=k2/dn;
	denumerator[0]=1;
	denumerator[1]=(2*(k2-1))/dn;
	denumerator[2]=(1-sqrtTwo*k+k2)/dn;

```

```c
        BQF_BQFinitFromCoefficents(pBQC,index, numerator, denumerator);
        return 0;
}

//! initializes a element of the biquad cascade as peak boost/cut filter
int BQF_BQFcascadeInitPeak (
                TBQF_BiquadCascade *    pBQC,   //!< (in/out) the biqaud cascade
                unsigned int                    index,  //!< (in) the index of the filter witch should be used as peak filter
                float                   fs,             //!< (in) sample frequency
                float                   fc,             //!< (in) center frequency
                float                   q,              //!< (in) quality (Q infinty)
                float                   g               //!< (in) gain
        ) {
// some info:
// @zlzer: g is in DB and is transformed via V0 = 10^(G/20) into v0 (DAFX p.55)
// @my function:
//              1. g is not in dB! g = V0
//              2. g < 1 => peak boost otherwise peak cut

        if (index >= pBQC->num) {
                return -1;
        }
        if (q == 0.0f) {
                return -2;
        }
        float k=tan(M_PI*fc/fs);
        float k2 = k * k;
        float dn;

        float numerator[3];
        float denumerator[3];

        if (g < 1) {
                // boost
                dn = 1.f+1.f/q*k+k2;
                numerator[0] = (1.f+g/q*k+k2) / dn;
                numerator[2] = (1.f-g/q*k+k2) / dn;
                denumerator[2] = (1.0f-1.0f/q*k+k2) / dn;
        } else {
                // cut
```

```c
                dn = (1.f+g/q*k+k2);
                numerator[0] = (1.f+1.f/q*k+k2) / dn;
                numerator[2] = (1.f-1.f/q*k+k2) / dn;
                denumerator[2] = (1.0f-g/q*k+k2) / dn;

        }
        numerator[1]=2.0f*(k2-1.0f)/dn;
        denumerator[0]=1.f;
        denumerator[1]=(2.f*(k2-1.f))/dn;

        BQF_BQFinitFromCoefficents(pBQC,index, numerator, denumerator);
        return 0;
}

//! initializes a element of the biquad cascade as low frequency shelving filter
int BQF_BQFcascadeInitLowFreqShelving (
                TBQF_BiquadCascade *    pBQC,    //!< (in/out) the biqaud cascade
                unsigned int                    index,    //!< (in) the index of the filter witch should be used as peak filter
                float    fs,                //!< (in) sample frequency
                float    f,                 //!< (in) cut/boost frequency
                float    q,                 //!< (in) quality (Q infinty)
                float    g                  //!< (in) gain
        ) {
// some info:
// @zlzer: g is in DB and is transformed via V0 = 10^(G/20) into v0 (DAFX p.55)
// @my function:
//              1. g is not in dB! g = V0
//              2. g < 1 => peak boost otherwise peak cut

        if (index >= pBQC->num) {
                return -1;
        }
        if (q == 0.0f) {
                return -2;
        }
        float k=tan(M_PI*f/fs);
        float k2 = k * k;
        float sqrtTwo = M_SQRT2;
        float sqrt2V0 = sqrtf(2.f * g);

        float dn;
```

```

        float numerator[3];
        float denumerator[3];

        if (g < 1) {
                // boost
                dn = 1.f+sqrtTwo*k+k2;
                numerator[0] = (1.f+sqrt2V0*k+g*k2) / dn;
                numerator[1] = (2.f * (g *k2 -1.f)) / dn;
                numerator[2] = (1.f-sqrt2V0*k+g*k2) / dn;
                denumerator[1] = (2.f * (k2 -1.f)) / dn;
                denumerator[2] = (1.f-sqrtTwo*k+k2) / dn;
        } else {
                // cut
                dn = 1.f+sqrt2V0*k+g*k2;
                numerator[0] = (1.f+sqrtTwo*k+k2) / dn;
                numerator[1] = (2.f * (k2 -1.f)) / dn;
                numerator[2] = (1.f-sqrtTwo*k+k2) / dn;
                denumerator[1] = (2.f * (g*k2 -1.f)) / dn;
                denumerator[2] = (1.f-sqrt2V0*k+g*k2) / dn;
        }
        denumerator[0]=1.f;

        BQF_BQFinitFromCoefficents(pBQC,index, numerator, denumerator);
        return 0;
}

//! initializes a element of the biquad cascade as high frequency shelving filter
int BQF_BQFcascadeInitHighFreqShelving (
                TBQF_BiquadCascade *    pBQC,   //!< (in/out) the biqaud cascade
                unsigned int                    index,  //!< (in) the index of the filter witch should be used as peak filter
                float   fs,             //!< (in) sample frequency
                float   f,              //!< (in) cut/boost frequency
                float   q,              //!< (in) quality (Q infinty)
                float   g               //!< (in) gain
        ) {
// some info:
```

```
// @zlzer: g is in DB and is transformed via V0 = 10^(G/20) into v0 (DAFX p.55)
// @my function:
//              1. g is not in dB! g = V0
//              2. g < 1 => peak boost otherwise peak cut

        if (index >= pBQC->num) {
                return -1;
        }
        if (q == 0.0f) {
                return -2;
        }
        float k=tan(M_PI*f/fs);
        float k2 = k * k;
        float sqrtTwo = M_SQRT2;
        float sqrt2V0 = sqrtf(2.f * g);
        float sqrt2divV0 = sqrtf(2.f / g);

        float dn;
        float dn2;

        float numerator[3];
        float denumerator[3];

        if (g < 1) {
                // boost
                dn = 1.f+sqrtTwo*k+k2;
                numerator[0] = (g+sqrt2V0*k+k2) / dn;
                numerator[1] = (2.f * (k2 -g)) / dn;
                numerator[2] = (g-sqrt2V0*k+k2) / dn;
                denumerator[1] = (2.f * (k2 -1.f)) / dn;
                denumerator[2] = (1.f-sqrtTwo*k+k2) / dn;
        } else {
                // cut
                dn = g+sqrt2V0*k+k2;
                dn2 = 1 + sqrt2divV0 * k + k2/g;
                numerator[0] = (1.f+sqrtTwo*k+k2) / dn;
                numerator[1] = (2.f * (k2 -1.f)) / dn;
                numerator[2] = (1.f-sqrtTwo*k+k2) / dn;
                denumerator[1] = (2.f * (k2/g -1.f)) / dn2;
                denumerator[2] = (1.f-sqrt2divV0*k+k2/g) / dn2;
        }
        denumerator[0]=1.f;

        BQF_BQFinitFromCoefficents(pBQC,index, numerator, denumerator);
        return 0;

}

//! initializes a element of the biquad cascade from a
```

```
   array with the coefficients
// input
// H(z) = (n0*z0 + n1*z-1 + n2 * z-2) / (d0*z0 + d1*z-1 + d2 * z-2)
// output
//        1/n0 * (n0/n0*z0 + n1/n0*z-1 + n2/n0 * z-2)
// H(z) =-----------------------------------------
//        1/d0 * (d0/d0*z0 + d1/d0*z-1 + d2/d0 * z-2)
// with now
// k = d0/n0
// n0 = 1
// n1 = n1/n0
// n2 = n2/n0
// d0 = 1
// d1 = d1/d0
// d2 = d2/d0
void BQF_BQFinitFromCoefficents (
                TBQF_BiquadCascade *    pBQC,             //!< (in/out) the biquad cascade
                int                     index,            //!< (in) the index of the filter witch should be used as peak filter
                float                   numerator [3],  //!< (in) n0 to n2
                float                   denominator [3] //!< (in) d0 to d2
        ) {
        TBQF_BiquadDF2 * pBQ = pBQC->pB + index;
        pBQ->n1 = numerator[1] / numerator[0];
        pBQ->n2 = numerator[2] / numerator[0];
        pBQ->d1 = denominator[1] / denominator[0];
        pBQ->d2 = denominator[2] / denominator[0];
        pBQ->k =  denominator[0] / numerator[0];
        pBQ->s1 = 0.f;
        pBQ->s2 = 0.f;
}

//! gets from an element of the biquad cascade the coefficients
// H1 is the internal form
// H1(z) = k (1*z0 + n1*z-1 + n2 * z-2) / (1*z0 + d1*z-1 + d2 * z-2)
// H2 is the external form
// H2(z) = (n0*z0 + n1*z-1 + n2 * z-2) / (d0*z0 + d1*z-1 + d2 * z-2)
// Formula:
// n0 = k
// n1 = k*n1
// n2 = k*n2
// d0 = 1
// d1 = d1
// d2 = d2
void BQF_BQFgetNumAndDenom (
```

```c
                TBQF_BiquadCascade *    pBQC,           //!< (in/out) the biquad cascade
                int                     index,          //!< (in) the index of the filter witch should be used as peak filter
                float *                 pNumerator,     //!< (out) n0 to n2
                float *                 pDenominator    //!< (out) d0 to d2
        ) {
        TBQF_BiquadDF2 * pBQ = pBQC->pB + index;
        pNumerator[0] = pBQ->k;
        pNumerator[1] = pBQ->k*pBQ->n1;
        pNumerator[2] = pBQ->k*pBQ->n2;
        pDenominator[0] = 1.0f;
        pDenominator[1] = pBQ->d1;
        pDenominator[2] = pBQ->d2;
}

#ifdef dBQF_implementPrintFunctions

// a simple print function (k, n1, n2, d1, d2)
void BQF_printBiquadDF2(TBQF_BiquadCascade *pBC, FILE * stream) {
        int i;
        TBQF_BiquadDF2 * pB;

        fprintf(stream,"BQC:\n");
        pB = pBC->pB;
        for (i = 0; i < pBC->num; i++) {
                fprintf(stream,"[%i]:\n",i);
                fprintf(stream,"\tk = %f\n",pB->k);
                fprintf(stream,"\tn1 = %f\n",pB->n1);
                fprintf(stream,"\tn2 = %f\n",pB->n2);
                fprintf(stream,"\td1 = %f\n",pB->d1);
                fprintf(stream,"\td2 = %f\n",pB->d2);
                pB++;
        }
        fflush(stream);
}

// a simple print function (n0, n1, n2, d0, d1, d2)
void BQF_PrintBiquad(TBQF_BiquadCascade *pBC, FILE * stream) {
        int i,j;
        float num[3];
        float denom[3];

        fprintf(stream,"BQC:\n");
        for (i = 0; i < pBC->num; i++) {
                BQF_BQFgetNumAndDenom(pBC, i, num, denom);
```

```
                    fprintf(stream,"[%i]:\n",i);
                    for (j = 0; j < 3; j++) {
                            fprintf(stream,"\tn%i = %f\n",j,
                               num[j]);
                    }
                    for (j = 0; j < 3; j++) {
                            fprintf(stream,"\td%i = %f\n",j,
                               denom[j]);
                    }

          }
          fflush(stream);
}

#endif
```

3.1.7 boost

code:

```
```

3.1.8 fftw3 & complex

includes:

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

code:

```
//   ==================================
// fftw3 libis used for complex and fft & ifft functions
//   ==================================
```

3.1.9 generic delay

includes:

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

code:

```
//
   ===============================================================
```

```c
// generic delay implementation (start)
// ===========================================================================

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

//! a delay
typedef struct SgenDelay {
        float * pStart; //!< starting pointer
        float * pEnd;   //!< end pointer
        int             amount; //!< amount of elements
        float * pR;             //!< read pointer
        float * pW;             //!< write pointer
} TgenDelay;

//! a enum type for the read / write flags
typedef enum EgenDelayRWflag {
        EgenDelayRWflag_readPointer = 0,            //!< only read pointer
        EgenDelayRWflag_writePointer = 1,           //!< only write pointer
        EgenDelayRWflag_readAndWritePointer = 2     //!< read & write pointer
} TgenDelayRWflag;

//! inits the delay
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)
```

```
                    amount of elements at the memory
                 int
                    clearBufferFlag          //!< if not
                    zero the buffer get overwritten with
                    0.0f
        ) {
        // set pointers
        pD->pStart = pStart;
        pD->amount = amount;
        pD->pEnd = pStart + amount;
        pD->pW = pStart;
        pD->pR = pStart;
        if (clearBufferFlag) {
                // and the buffer to 0.0f
                float * pB = pStart;
                while (amount) {
                        *pB = 0.0f;
                        pB++;
                        amount--;
                }
        }
}

//! creates the structure and the buffer for a delay
TgenDelay * genDelay_create (int amountOfSamples) {
        // malloc structure
        TgenDelay * pD = malloc(sizeof(TgenDelay)*
           amountOfSamples);
        if (!pD) return NULL;
        if (amountOfSamples > 0) {
                // malloc data buffer
                pD->pStart = malloc(sizeof(float)*
                   amountOfSamples);
                if (!pD->pStart) {
                        free (pD);
                        return NULL;
                }
                // reset structure
                genDelay_init(pD,pD->pStart,
                   amountOfSamples, 1);
        } else {
                genDelay_init(pD,NULL,0, 0);
        }
        return pD;
}

//! deletes the delay structure and the contained data
void genDelay_delete (TgenDelay * pD) {
        if (!pD) return;
        if (pD->pStart) free(pD->pStart);
        free(pD);
}

//!
int genDelay_resize (TgenDelay * pD, int newBufferSize)
```

```
{
        // null pointer exception
        if (!pD) return -1;
        // have we realy to alloc new memory?
        if (pD->amount == newBufferSize) goto init;
        // ok lets free the old one
        free (pD->pStart);
        // and alloc a new one
        pD->pStart = malloc(sizeof(float)*newBufferSize);
        // check if it worked
        if (!pD->pStart) {
                // no - ok lets make stable state and report an error
                genDelay_init(pD,NULL,0, 0);
                return -2;
        }
        // it worked lets init
init:
        genDelay_init(pD,pD->pStart,newBufferSize, 1);
        return 0;

}

// shuffle the read and/or write pointer
void genDelay_shuffle (
                TgenDelay *     pD,     //!< (in/out) pointer to a delay struct
                TgenDelayRWflag what,           //!< what defines what pointer is affected
                int                     offset          //!< the offset of the R/W pointer
        ) {
        int modf = offset % pD->amount;
        // read pointer
        if ((what == EgenDelayRWflag_readPointer) || (what == EgenDelayRWflag_readAndWritePointer)) {
                pD->pR += modf;
                if (pD->pR > pD->pEnd) {
                        pD->pR = pD->pStart + (pD->pR - pD->pEnd);
                } else {
                        if (pD->pR < pD->pStart) {
                                pD->pR = pD->pEnd - (pD->pStart - pD->pR);
                        }
                }
        }

        // write pointer
        if ((what == EgenDelayRWflag_writePointer) || (what == EgenDelayRWflag_readAndWritePointer))
```

```
			{
				pD->pW += modf;
				if (pD->pW > pD->pEnd) {
					pD->pW = pD->pStart + (pD->pW - pD->pEnd);
				} else {
					if (pD->pW < pD->pStart) {
						pD->pW = pD->pEnd - (pD->pStart - pD->pW);
					}
				}
		}

}

//! writes a grain of values to the buffer
void genDelay_write (
				TgenDelay *	pD,		//!< (in/out) pointer to a delay struct
				float *			pSrc,	//!< (in) pointer to the values put into the delay
				int				N		//!< (in) amount of values to be stored
		) {
		while (N) {
				// copy value
				*pD->pW = *pSrc;
				// inc src pointer
				pSrc++;
				// inc write pointer
				pD->pW++;
				// does we extends the border?
				if (pD->pW >= pD->pEnd) {
					// yes - ok lets wrap
					pD->pW = pD->pStart;
				}
				N--;
		}
}

//! reads a grain of values form the buffer
void genDelay_read (
				TgenDelay *	pD,		//!< (in) pointer to a delay struct
				float *			pDest,	//!< (in/out) pointer to which the values are stored
				int				N		//!< (in) amount of values to be stored
		) {
		while (N) {
				// copy value
```

```
		*pDest = *pD->pR;
		// inc src pointer
		pDest++;
		// inc read pointer
		pD->pR++;
		// does we extends the border?
		if (pD->pR >= pD->pEnd) {
			// yes - ok lets wrap
			pD->pR = pD->pStart;
		}
		N--;
	}
}

//! reads a grain of values form the buffer and copies the same amount into the buffer
void genDelay_readWrite (
		TgenDelay *	pD,			//!< (in) pointer to a delay struct
		float *			pSrc,	//!< (in) a pointer from which the values should be copied
		float *			pDest,	//!< (in/out) pointer to which the values are stored
		int						N	//!< (in) amount of values to be stored
	) {
	// 3 state approach to minimize the amount of i/o operations
	// 1st state read till we reach the write pointer or N is 0
	// 2nd state read & write till we read all data needed
	// 3th state write till all data is written

	// wrap counter
	int wc;

	// read number
	int rn, rn2;
	// lets calculate how many read operations are needed to reach the write pointer
	// first check the position of the write pointer
	if (pD->pW >= pD->pR) {
		// the write pointer is in front of the read pointer
		rn = pD->pW - pD->pR;
	} else {
		// the write pointer stands after the read pointer
		rn = pD->amount - (pD->pR - pD->pW);
	}
	// trim
```

```
        if (rn > N) rn = N;
        rn2 = N - rn;

        // 1st state
        // 1.1. read till it wraps
        wc = pD->pEnd - pD->pR;
        if (wc > rn) wc = rn;
        rn -= wc;
        while (wc) {
                wc--;
                *pDest = *pD->pR;
                pDest++;
                pD->pR++;
        }
        // wrap if needed
        if (pD->pR >= pD->pEnd) pD->pR = pD->pStart;
        // 1.2 and read on
        while (rn) {
                rn --;
                *pDest = *pD->pR;
                pDest++;
                pD->pR++;
        }

        // 2nd state
        // the read/write operation
        // we use only the pD->pW pointer for the operation and increment the pD->pR only once
        // in rn2 are the remaining numbers stored
        pD->pR += rn2;
        N -= rn2;
        // wrap pR if needed
        if (pD->pR >= pD->pEnd) pD->pR = pD->pStart + (pD->pR - pD->pEnd);

        // get the wrap counter
        wc = pD->pEnd - pD->pW;
        if (wc > rn2) wc = rn2;

        // get the remaining number
        rn = rn2 - wc;

        // write & read
        while (wc) {
                wc--;
                *pDest = *pD->pW;
                *pD->pW = *pSrc;
                pDest++;
                pSrc++;
                pD->pW++;
        }
        // warp
        if (pD->pW >= pD->pEnd) pD->pW = pD->pStart;
        // write & read
        while (rn) {
```

```
                rn--;
                *pDest = *pD->pW;
                *pD->pW = *pSrc;
                pDest++;
                pSrc++;
                pD->pW++;
        }

        // 3th state
        // write the data
        // at N are the number of the remaining data

        // get the wrap counter
        wc = pD->pEnd - pD->pW;
        if (wc > N) wc = N;
        N -= wc;
        // write the data
        while (wc) {
                wc--;
                *pD->pW = *pSrc;
                pSrc++;
                pD->pW++;
        }
        // wrap
        if (pD->pW >= pD->pEnd) pD->pW = pD->pStart;
        // write the remaining data
        while (N) {
                N--;
                *pD->pW = *pSrc;
                pSrc++;
                pD->pW++;
        }

}

//
   ====================================================================

// generic delay implementation (end)
//
   ====================================================================
```

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:

```
// -------------------------------------------
// AP gtk helper
// -------------------------------------------

PangoFontDescription * gAPgtkMonoSpaceFont = NULL;


// a helper for destroying "undestroyed" UI's
void APgtk_helper_destroy (GtkWidget * pWG) {
        if (pWG) {
                if (pWG->object.parent_instance.
                   ref_count) {
                        gtk_widget_destroy(pWG);
                }
        }
}

// -------------------------------------------
// AP panel
// -------------------------------------------

// struct for a panel
typedef struct SAPgtkPanel {
        GtkWidget *             pWnd;           //!<
           pointer to the window
        GtkWidget *             pBox;           //!<
           placement box
} TAPgtkPanel;


// create a panel
int gtkAP_local_PanelCreate (
                TAPgtkPanel *   pP,             //!<(in/
```

```
                        out) structure to be filled
                int                              x,          //!<(in) x position of the window
                int                              y,          //!<(in) y position of the window
                int                              xle,        //!<(in) x length of the window
                int                              yle         //!<(in) x length of the window
        ) {
        // Create a new window
        pP->pWnd = gtk_window_new (GTK_WINDOW_TOPLEVEL);
        if (!pP->pWnd) return -1;

        // UI's are at fixed positions
        pP->pBox = gtk_fixed_new();
        if (!pP->pBox) return -2;

        gtk_container_add(GTK_CONTAINER(pP->pWnd), pP->pBox);

        // setup main wnd
        gtk_window_set_title (GTK_WINDOW (pP->pWnd), "AP Panel");
        gtk_window_set_default_size (GTK_WINDOW (pP->pWnd),xle, yle);
        gtk_window_move(GTK_WINDOW (pP->pWnd),x, y);

        // It's a good idea to do this for all windows
        //gtk_signal_connect (GTK_OBJECT (pP->pWnd), "destroy", GTK_SIGNAL_FUNC (gtk_exit), NULL);
        gtk_signal_connect (GTK_OBJECT (pP->pWnd), "destroy", GTK_SIGNAL_FUNC (gtk_widget_destroy), NULL);

        //gtk_signal_connect (GTK_OBJECT (pP->pWnd), "delete_event", GTK_SIGNAL_FUNC (gtk_exit), NULL);
        gtk_signal_connect (GTK_OBJECT (pP->pWnd), "delete_event", GTK_SIGNAL_FUNC (gtk_widget_destroy), NULL);

        // Sets the border width of the window
        gtk_container_set_border_width (GTK_CONTAINER (pP->pWnd), 1);

        gtk_widget_realize(pP->pWnd);

        gtk_widget_show_all(pP->pWnd);

        return(0);
}
```

```


// destroys a panel
void gtkAP_local_PanelDestroy (
                TAPgtkPanel *    pP
        ) {
        APgtk_helper_destroy(pP->pWnd);
        APgtk_helper_destroy(pP->pBox);
        pP->pWnd = NULL;
        pP->pBox = NULL;
}

// -------------------------------------------
// AP button
// -------------------------------------------

// struct for a AP button
typedef struct SAPgtkButton {
        GtkWidget *             pB;         //!< pointer to the button
        int                          pressCounter;   //!< counter of the press event
} TAPgtkButton;

// call back to handle click calls
void gtkAP_cb_ButtonClicked (GtkWidget *widget, gpointer data) {
        TAPgtkButton * pB = (TAPgtkButton *) data;
        pB->pressCounter++;
}

// creates a AP button
int gtkAP_local_ButtonCreate (
                TAPgtkButton *  pB,             //!<(in/out) structure to be filled
                TAPgtkPanel *   pP,             //!<(in) panel witch holds the button
                int                             x,              //!<(in) x position of the window
                int                             y,              //!<(in) y position of the window
                int                             xle,    //!<(in) x length of the window
                int                             yle             //!<(in) x length of the window
        ) {


        pB->pressCounter = 0;
        // create button
        pB->pB = gtk_button_new_with_label ("");
        if (!pB->pB) return -1;
```

```

        // bind it to the window
        gtk_fixed_put(GTK_FIXED(pP->pBox), pB->pB, x, y);
        gtk_widget_set_size_request(pB->pB, xle, yle);

        gtk_widget_modify_font(gtk_bin_get_child (GTK_BIN(pB->pB)),gAPgtkMonoSpaceFont);

        gtk_widget_show (pB->pB);

        // set msg handler
        g_signal_connect(G_OBJECT(pB->pB), "clicked", G_CALLBACK(gtkAP_cb_ButtonClicked), pB);


        return 0;
}

// destroys a button
void gtkAP_local_ButtonDestroy (
                TAPgtkButton *  pB
        ) {
        APgtk_helper_destroy(pB->pB);

        pB->pB = NULL;
        pB->pressCounter = 0;
}

// ----------------------------------------------
// AP LED
// ----------------------------------------------

// struct for a AP led
typedef struct SAPgtkLED {
        GtkWidget *             pLED;           //!< pointer to the LED
        int                             onFlag;
        double                  onColor[3];
        double                  offColor[3];
} TAPgtkLED;


static gboolean APgtk_cb_LEDexpose (
                GtkWidget *                     da,
                GdkEventExpose *        event,
                gpointer                        data
        )
{
  cairo_t *             cr;
  TAPgtkLED     *       pLED = (TAPgtkLED       *) data;
  double                r;
  cr = gdk_cairo_create (da->window);
  gdk_cairo_rectangle (cr, &event->area);
  cairo_clip (cr);
```

```c
  if (pLED->onFlag) {
          cairo_set_source_rgb (cr, pLED->onColor[0], pLED->onColor[1], pLED->onColor[2]);
  } else {
          cairo_set_source_rgb (cr, pLED->offColor[0], pLED->offColor[1], pLED->offColor[2]);
  }
  //cairo_rectangle(cr, event->area.x, event->area.y, event->area.width, event->area.height);
  r = (event->area.width > event->area.height) ? event->area.width : event->area.height;
  r /= 2.f;
  cairo_arc (cr,event->area.x+r, event->area.y+r,r, 0.f, 2.f * M_PI);

  cairo_fill(cr);
  cairo_destroy (cr);
  return TRUE;
}

// creates a AP button
int gtkAP_local_LEDCreate (
                TAPgtkLED *     pLED,   //!<(in/out) structure to be filled
                TAPgtkPanel *   pP,             //!<(in) panel witch holds the button
                int                             x,              //!<(in) x position of the window
                int                             y,              //!<(in) y position of the window
                int                             xle,    //!<(in) x length of the window
                int                             yle             //!<(in) x length of the window
        ) {

        pLED->onFlag = 0;
        pLED->offColor[0] = 0.0;
        pLED->offColor[1] = 0.0;
        pLED->offColor[2] = 0.0;
        pLED->onColor[0] = 1.0;
        pLED->onColor[1] = 1.0;
        pLED->onColor[2] = 1.0;


        pLED->pLED = gtk_drawing_area_new ();
        if (!pLED->pLED) return -1;
        gtk_widget_set_size_request (pLED->pLED, xle, yle);

        gtk_fixed_put(GTK_FIXED(pP->pBox), pLED->pLED, x, y);
```

```
        g_signal_connect (pLED->pLED, "expose-event",
           G_CALLBACK (APgtk_cb_LEDexpose), pLED);

        gtk_widget_show (pLED->pLED);

        return 0;
}

// destroys a button
void gtkAP_local_LEDDestroy (
                TAPgtkLED *      pLED
        ) {
        APgtk_helper_destroy(pLED->pLED);

        pLED->pLED = NULL;
        pLED->onFlag = 0;
        pLED->offColor[0] = 0.0;
        pLED->offColor[1] = 0.0;
        pLED->offColor[2] = 0.0;
        pLED->onColor[0] = 0.0;
        pLED->onColor[1] = 0.0;
        pLED->onColor[2] = 0.0;
}


// -------------------------------------------
// AP Display
// -------------------------------------------

// struct for a AP led
typedef struct SAPgtkDisplay {
        GtkWidget *             pD;
                   //!< pointer to the display
        GtkWidget *             pF;
                   //!< frame around the label
        int                             charsPerLine;
        int                             lineCount;
} TAPgtkDisplay;


// creates a AP display
int gtkAP_local_DisplayCreate (
                TAPgtkDisplay * pD,             //!<(in/
                   out) structure to be filled
                TAPgtkPanel *   pP,             //!<(in)
                    panel witch holds the button
                int                             x,
                           //!<(in) x position of the
                   window
                int                             y,
                           //!<(in) y position of the
                   window
                int                             xle,
                   //!<(in) x length of the window
```

```
                int                                    yle
                          //!<(in) x length of the
                   window
        ) {

        pD->pD = NULL;
        pD->charsPerLine = 10;
        pD->lineCount = 4;

        pD->pD = gtk_label_new ("");
        if (!pD->pD) return -1;
        pD->pF = gtk_frame_new(NULL);
        if (!pD->pF) return -2;

        //gtk_widget_set_size_request (pD->pD, xle-2,
           yle-2);
        gtk_widget_set_size_request (pD->pF, xle, yle);

        //gtk_fixed_put(GTK_FIXED(pP->pBox), pD->pD, x,
           y);
        gtk_fixed_put(GTK_FIXED(pP->pBox), pD->pF, x, y)
           ;
        gtk_container_add(GTK_CONTAINER(pD->pF), pD->pD)
           ;

        //gtk_widget_modify_font(gtk_bin_get_child (
           GTK_BIN(pD->pD)),gAPgtkMonoSpaceFont);
        gtk_widget_modify_font(pD->pD,
           gAPgtkMonoSpaceFont);

        /*
        // change colors
        GdkColor color;
        gdk_color_parse ("yellow", &color);
        gtk_widget_modify_fg (pD->pD, GTK_STATE_NORMAL,
           &color);
        // background only works for the main wnd!
        gdk_color_parse ("green", &color);
        gtk_widget_modify_bg(pP->pWnd, GTK_STATE_NORMAL,
            &color);
         */

        gtk_widget_show (pD->pD);
        gtk_widget_show (pD->pF);

        return 0;
}

// destroys a display
void gtkAP_local_DisplayDestroy (
                TAPgtkDisplay *         pD
        ) {
        APgtk_helper_destroy(pD->pD);

        pD->pD = NULL;
```

```
        pD->charsPerLine = 10;
        pD->lineCount = 2;
}


// destroys a button
void gtkAP_local_DisplaySetText(
                TAPgtkDisplay *         pD,
                char *                          szTxt
        ) {
        const char const cLF = 0xA;
        char szdummyBuffer[pD->lineCount*pD->
           charsPerLine+1];
        char * pC = szdummyBuffer;
        int x = 0;
        int y = 0;

        while (*szTxt) {
                // check if we have a line break
                if (*szTxt == cLF) {
                        y++;
                        x = 0;
                        // check if we reached the end
                        if (y == pD->lineCount) {
                                *pC = 0;
                                goto printLines;
                        }
                }

                // copy char
                *pC = *szTxt;
                szTxt++;
                pC++;
                x++;

                // check if we reached the end of the
                   line
                if (x == pD->charsPerLine) {
                        y++;
                        x=0;
                        // check if we reached the end
                        if (y < pD->lineCount) {
                                *pC = cLF;
                                pC++;
                        } else {
                                *pC = 0;
                                goto printLines;
                        }
                }

        }
        *pC = 0;
printLines:
        gtk_label_set_text(GTK_LABEL(pD->pD),
           szdummyBuffer);
```

```
}

// ============================================
// AP gtk+ interface
// ============================================

enum eAPgtkActionID {
        eAPgtkActionID_exit = 1,
        eAPgtkActionID_redrawUI                  = 10,
        eAPgtkActionID_setDisplayText    = 11,
        eAPgtkActionID_createPanel               = 20,
        eAPgtkActionID_deletePanel               = 21,
        eAPgtkActionID_createButton              = 30,
        eAPgtkActionID_deleteButton              = 31,
        eAPgtkActionID_createLED                 = 40,
        eAPgtkActionID_deleteLED                 = 41,
        eAPgtkActionID_createDisplay     = 50,
        eAPgtkActionID_deleteDisplay     = 51,

};

typedef struct SAPgtk_command {
        sem_t           ps;                     //!< process finsh semaphore
        GMutex *        gm;                     //!< guarding mutex
        int                     newCmdFlag;     //!< is set to indicate that there is a new command

        int             cmd;            //!< command
        void *          pData1;
        void *          pData2;
        int             x;
        int                     y;
        int                     xle;
        int                     yle;


} TAPgtk_command;


TAPgtk_command                  gAPgtkMsg;
GThread *                               gAPgtkThread;

void APgtk_setMsg (
                int             id,
                void *  pD1,
                void *  pD2,
                int             x,
                int             y,
                int             xle,
                int             yle
) {
        g_mutex_lock(gAPgtkMsg.gm);
```

```
        // set the command
        gAPgtkMsg.cmd = id;
        gAPgtkMsg.pData1 = pD1;
        gAPgtkMsg.pData2 = pD2;
        gAPgtkMsg.x = x;
        gAPgtkMsg.y = y;
        gAPgtkMsg.xle = xle;
        gAPgtkMsg.yle = yle;
        gAPgtkMsg.newCmdFlag = 1;
        g_mutex_unlock(gAPgtkMsg.gm);

        sem_wait(&gAPgtkMsg.ps);
}

gpointer APgtk_threadFunc (gpointer Args) {
        // init gtk and bind it to the thread
        if (FALSE ==  gtk_init_check(NULL, NULL)) return ((gpointer)-1);

        gAPgtkMonoSpaceFont = pango_font_description_from_string("monospace");

        // process messages & user actions
        for (;;) {
                // check for message
                g_mutex_lock(gAPgtkMsg.gm);
                if (gAPgtkMsg.newCmdFlag) {
                        switch (gAPgtkMsg.cmd) {
                                case eAPgtkActionID_exit:
                                        goto exit;
                                        break;
                                case eAPgtkActionID_redrawUI:
                                        gtk_widget_queue_draw(GTK_WIDGET(gAPgtkMsg.pData1));
                                        break;

                                case eAPgtkActionID_setDisplayText:
                                        gtkAP_local_DisplaySetText(gAPgtkMsg.pData1, gAPgtkMsg.pData2);
                                        break;

                                case eAPgtkActionID_createPanel:
```

```
gtkAP_local_PanelCreate
    (
        gAPgtkMsg
        .
        pData1
        ,
        gAPgtkMsg
        .
        x
        ,
        gAPgtkMsg
        .
        y
        ,
        gAPgtkMsg
        .
        xle
        ,
        gAPgtkMsg
        .
        yle
    );
    break;
case
    eAPgtkActionID_deletePanel
    :
        gtkAP_local_PanelDestroy
            (gAPgtkMsg.
            pData1);
        break;

case
    eAPgtkActionID_createButton
    :
        gtkAP_local_ButtonCreate
            (
                gAPgtkMsg
                .
                pData1
                ,
                gAPgtkMsg
                .
                pData2
                ,
                gAPgtkMsg
                .
                x
                ,
```

```
                    gAPgtkMsg.y,
                    gAPgtkMsg.xle,
                    gAPgtkMsg.yle
                );
            break;
        case eAPgtkActionID_deleteButton:
            gtkAP_local_ButtonDestroy(gAPgtkMsg.pData1);
            break;

        case eAPgtkActionID_createLED:
            gtkAP_local_LEDCreate(
                    gAPgtkMsg.pData1,
                    gAPgtkMsg.pData2,
                    gAPgtkMsg.x,
                    gAPgtkMsg.y,
                    gAPgtkMsg.xle,
```

```
                                        gAPgtkMsg.yle
                                    );
                    break;
                case eAPgtkActionID_deleteLED:
                    gtkAP_local_LEDDestroy(gAPgtkMsg.pData1);
                    break;

                case eAPgtkActionID_createDisplay:
                    gtkAP_local_DisplayCreate(
                                        gAPgtkMsg.pData1,
                                        gAPgtkMsg.pData2,
                                        gAPgtkMsg.x,
                                        gAPgtkMsg.y,
                                        gAPgtkMsg.xle,
                                        gAPgtkMsg.yle
                                    );
                    break;
                case eAPgtkActionID_deleteDisplay:
                    gtkAP_local_DisplayDestroy(gAPgtkMsg.
```

```
                                                    pDatal);
                                                break;
                                }
                                gAPgtkMsg.newCmdFlag = 0;
                                sem_post(&gAPgtkMsg.ps);
                        }
                        g_mutex_unlock(gAPgtkMsg.gm);

                        // message handling
                        if (TRUE == gtk_events_pending ()) {
                                gtk_main_iteration ();
                        }

                }
                goto end;
exit:
                sem_post(&gAPgtkMsg.ps);
                g_mutex_unlock(gAPgtkMsg.gm);
end:
                pango_font_description_free(gAPgtkMonoSpaceFont)
                   ;
                gAPgtkMonoSpaceFont = NULL;
                return NULL;
}

int APgtk_start () {

                if (!g_thread_supported ()) g_thread_init (NULL)
                   ;

                gAPgtkMsg.newCmdFlag = 0; // no cmd ready
                gAPgtkMsg.gm = g_mutex_new ();
                sem_init(&gAPgtkMsg.ps,0,0);

                gAPgtkThread = g_thread_create(APgtk_threadFunc,
                    NULL,TRUE, NULL);
                return 0;
}


void APgtk_end () {
                APgtk_setMsg(eAPgtkActionID_exit, NULL, NULL, 0,
                    0, 0, 0);
                g_thread_join(gAPgtkThread);
                g_mutex_free(gAPgtkMsg.gm);
                sem_destroy(&gAPgtkMsg.ps);
}

// ----------------------------
// AP interface
// ----------------------------

// create a panel
int gtkAP_PanelCreate (
                        TAPgtkPanel *   pP,             //!<(in/
```

```
out) structure to be filled
                int                           x,                    //!<(in) x position of the window
                int                           y,                    //!<(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_createPanel, pP, NULL, x, y, xle, yle);

        return(0);
}


// destroys a panel
void gtkAP_PanelDestroy (
                TAPgtkPanel *   pP
        ) {
        APgtk_setMsg (eAPgtkActionID_deletePanel, pP, NULL, 0,0,0,0);
}


// create a button
int gtkAP_ButtonCreate (
                TAPgtkButton *  pB,             //!<(in/out) structure to be filled
                TAPgtkPanel *   pP,             //!<(in) panel witch holds the button
                int                           x,                    //!<(in) x position of the window
                int                           y,                    //!<(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_createButton, pB, pP, x, y, xle, yle);

        return(0);
}

```

```

// destroys a button
void gtkAP_ButtonDestroy (
                TAPgtkButton *  pB
        ) {
        APgtk_setMsg (eAPgtkActionID_deleteButton, pB,
           NULL, 0,0,0,0);
}

// create a LED
int gtkAP_LEDCreate (
                TAPgtkLED *     pLED,   //!<(in/out)
                   structure to be filled
                TAPgtkPanel *   pP,             //!<(in)
                    panel witch holds the button
                int                             x,
                        //!<(in) x position of the
                   window
                int                             y,
                        //!<(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_createLED, pLED, pP
           , x, y, xle, yle);

        return(0);
}


// destroys a LED
void gtkAP_LEDDestroy (
                TAPgtkLED *     pLED
        ) {
        APgtk_setMsg (eAPgtkActionID_deleteLED, pLED,
           NULL, 0,0,0,0);
}


// create a display
int gtkAP_DisplayCreate (
                TAPgtkDisplay * pD,             //!<(in/
                   out) structure to be filled
                TAPgtkPanel *   pP,             //!<(in)
                    panel witch holds the button
                int                             x,
                        //!<(in) x position of the
                   window
                int                             y,
                        //!<(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 {
```

```
                TAPgtkPanel                 panel;
                TAPgtkButton        button;
                TAPgtkDisplay       display;
                TAPgtkLED                   led;
        } ui;
} TAPgtkUI;

// set the coordinates of the ui
void APgtkUI_setCoordinates (
                TAPgtkUI *          pUI,
                int                 x,
                int                         y,
                int                         xle,
                int                         yle
        ) {
        pUI->x = x;
        pUI->y = y;
        pUI->xle = xle;
        pUI->yle = yle;
}

// creates visible ui
int APgtkUI_createUI (
                TAPgtkUI *          pUI,
                TAPgtkUI *          pParentUI,
                int                         uuid,
                int                         typeID
        ) {
        pUI->typeID = typeID;
        pUI->uuid = uuid;
        switch (typeID) {
                case eAPgtkUItype_panel:
                        return gtkAP_PanelCreate(
                                    &(pUI->ui.panel)
                                        ,
                                    pUI->x,
                                    pUI->y,
                                    pUI->xle,
                                    pUI->yle
                                );
                case eAPgtkUItype_button:
                        return gtkAP_ButtonCreate(
                                    &(pUI->ui.button
                                        ),
                                    &(pParentUI->ui.
                                        panel),
                                    pUI->x,
                                    pUI->y,
                                    pUI->xle,
                                    pUI->yle
                                );
                case eAPgtkUItype_LED:
                        return gtkAP_LEDCreate(
                                    &(pUI->ui.led),
                                    &(pParentUI->ui.
```

```
                                           panel),
                                        pUI->x,
                                        pUI->y,
                                        pUI->xle,
                                        pUI->yle
                                );
                case eAPgtkUItype_display:
                        return gtkAP_DisplayCreate(
                                        &(pUI->ui.
                                           display),
                                        &(pParentUI->ui.
                                           panel),
                                        pUI->x,
                                        pUI->y,
                                        pUI->xle,
                                        pUI->yle
                                );
        }
        return -100;
}

// destroy visible ui
void APgtkUI_destroyUI (
                TAPgtkUI *      pUI
        ) {
        switch (pUI->typeID) {
                case eAPgtkUItype_panel:
                        gtkAP_PanelDestroy(&(pUI->ui.
                           panel));
                        break;
                case eAPgtkUItype_button:
                        gtkAP_ButtonDestroy(&(pUI->ui.
                           button));
                        break;
                case eAPgtkUItype_LED:
                        gtkAP_LEDDestroy(&(pUI->ui.led))
                           ;
                        break;
                case eAPgtkUItype_display:
                        gtkAP_DisplayDestroy(&(pUI->ui.
                           display));
                        break;
        }
        pUI->typeID = eAPgtkUItype_unknown;
        pUI->uuid = 0;
}

// redraws a ui
void APgtkUI_redrawUI (
                TAPgtkUI *      pUI
        ) {
        switch (pUI->typeID) {
                case eAPgtkUItype_panel:
                        APgtk_setMsg (
                           eAPgtkActionID_redrawUI, pUI
```

```
                    ->ui.panel.pWnd, NULL,
                    0,0,0,0);
                break;
            case eAPgtkUItype_button:
                APgtk_setMsg (
                    eAPgtkActionID_redrawUI, pUI
                    ->ui.button.pB, NULL,
                    0,0,0,0);
                break;
            case eAPgtkUItype_LED:
                APgtk_setMsg (
                    eAPgtkActionID_redrawUI, pUI
                    ->ui.led.pLED, NULL, 0,0,0,0)
                    ;
                break;
            case eAPgtkUItype_display:
                APgtk_setMsg (
                    eAPgtkActionID_redrawUI, pUI
                    ->ui.display.pD, NULL,
                    0,0,0,0);
                break;
        }
}


// AP UI array type
typedef struct SAPgtkUIvector {
        TAPgtkUI *      pUI;
        int                     number;
} TAPgtkUIvector;


TAPgtkUIvector * APgtkUI_createVector (
                int number,
                int uiType
        ) {
        TAPgtkUIvector *        pV;
        int                                     i;

        pV = malloc (sizeof(TAPgtkUIvector));
        if (!pV) return NULL;
        pV->pUI = malloc (sizeof(TAPgtkUI)*number);
        if (!pV->pUI) {
                free (pV);
                return NULL;
        }
        memset (pV->pUI,0,sizeof(TAPgtkUI)*number);

        pV->number = number;
        for (i = 0; i < number; i++) {
                pV->pUI[i].typeID = uiType;
        }

        return pV;
}
```

```
void APgtkUI_destroyVector (
                TAPgtkUIvector * pUIv
        ) {
        int i;
        for (i = 0; i < pUIv->number; i++) {
                APgtkUI_destroyUI(&(pUIv->pUI[i]));
        }
        free (pUIv->pUI);;
        free (pUIv);
}
```

3.1.11 libsndfile frame based

includes:

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

code:

```
// ###########################################
// block based processing without overlapped operations
// ###########################################

#define dStjFrameWAVinitialFrameAmount (1024)

typedef struct SStjFrameWAVOpenInfo {
        unsigned int    channelNumber;
           //!< logical channel
        char *                  szFileName;
                        //!< name of the file
        unsigned int    isInput;
                //!< if <>0 then this file is an
           input

        unsigned int    sampleRate;
                //!< the sample rate
        unsigned int    fileChannelNumber;
           //!< number of the channel at the file (
           starting at 1)
} TStjFrameWAVOpenInfo;

typedef struct SStjFrameWAVFile {
        unsigned int    channel;
                //!< the channel
        SNDFILE *               pSndF;
                        //!< the file
        unsigned int    isInput;
                //!< if 0 the channel is an input
           channel
```

```


        float *                 pFrameBuffer;               //!< the buffer witch is filled with the frames
        int                             fileChannels;               //!< the amount of sample channels at the file
        int                             activeChannel;              //!< the channel used to read
        int                             frameAmount;                //!< amount of frame
} TStjFrameWAVFile;

typedef struct SStjFrameWAVmodule {
        TStjFrameWAVFile *      pChannels;
        unsigned int            number;
} TStjFrameWAVmodule;

//! close the set of wav files
void FrameWAVmoduleExit (TStjFrameWAVmodule * pM);

//! initis a set of wav files for writing / reading
int FrameWAVmoduleInit (
                int                             number,          //!<(in) amount of files
                TStjFrameWAVOpenInfo *  pWAVFiles,      //!<(in) file description
                TStjFrameWAVmodule *    pModul          //!<(in/out) modul descriptor witch is filled
        );

//! looking for the right channel
TStjFrameWAVFile * FrameWAVmoduleFindChannel (
                TStjFrameWAVmodule *    pM,             //!< the module
                int                                     channel         //!< channel to search for
        );

//! reads a sample from a channel
int FrameWAVmoduleGetInput (
                TStjFrameWAVmodule *    pM,                     //!<(in) the module
                int                             channel,                //!<(in) the channel
                int                             sampleNumber,   //!<(in) number of samples to be read
                float *                 pFrame                  //!<(out) pointer to the sample buffer witch is
```

```
                        filled
        );

//! writes a sample to an output
int FrameWAVmoduleSetOutput (
                TStjFrameWAVmodule *    pM,
                                        //!<(in) the module
                int
                            channel,                    //!<(
                   in) the channel
                int
                            sampleNumber,   //!<(in)
                   number of samples to be write
                float *
                   pFrame                   //!<(in)
                   pointer to the sample buffer witch is
                    filled
        );

//! close the set of wav files
void FrameWAVmoduleExit (TStjFrameWAVmodule * pM) {
        //1. close all sound files
        int i;

        TStjFrameWAVFile * pWF = pM->pChannels;
        for (i = 0; i < pM->number;i++) {
                sf_close(pWF->pSndF);
                if (pWF->pFrameBuffer) free (pWF->
                   pFrameBuffer);
                pWF++;
        }
        //2. free array
        free (pM->pChannels);

        //3. set all vars of the struct to default
        pM->number = 0;
        pM->pChannels = NULL;
}


//! initis a set of wav files for writing / reading
int FrameWAVmoduleInit (
                int
                   number,          //!<(in) amount of
                   files
                TStjFrameWAVOpenInfo *  pWAVFiles,
                   //!<(in) file description
                TStjFrameWAVmodule *    pModul
                   //!<(in/out) modul descriptor witch
                   is filled
        ) {

        pModul->pChannels = malloc (sizeof(
           TStjFrameWAVFile) * number);
        if (!pModul->pChannels) {
```

```
        return -1;
    }
    memset (pModul->pChannels,0,sizeof(TStjFrameWAVFile) * number);
    pModul->number = number;

    int                             i;
    SF_INFO                         info;
    TStjFrameWAVFile *      pWF = pModul->pChannels;

    for (i = 0;i < number;i++) {
        // open sndfile interface
        memset (&info,0,sizeof(SF_INFO));
        if (pWAVFiles[i].isInput) {
            pWF->pSndF = sf_open (pWAVFiles[i].szFileName,SFM_READ, &info);
        } else {
            info.samplerate = pWAVFiles[i].sampleRate;
            info.channels = 1;
            info.format = SF_FORMAT_WAV | SF_FORMAT_FLOAT;
            pWF->pSndF = sf_open (pWAVFiles[i].szFileName,SFM_WRITE, &info);
        }
        if (!pWF->pSndF) {
            goto error;
        }
        pWF->isInput = (unsigned int) pWAVFiles[i].isInput;
        pWF->channel = (unsigned int) pWAVFiles[i].channelNumber;


        if (pWAVFiles[i].isInput) {
            pWF->pFrameBuffer = malloc (sizeof(float)*info.channels*dStjFrameWAVinitialFrameAmount);
            if (!pWF->pFrameBuffer) {
                goto error;
            }
            pWF->frameAmount = dStjFrameWAVinitialFrameAmount;
        }

        // the file channel starts at 1 - here we start at 0
        pWF->activeChannel = pWAVFiles[i].fileChannelNumber - 1;
        pWF->fileChannels = (int) info.channels;

```

```
		pWF++;
	}
	return 0;
error:
	FrameWAVmoduleExit(pModul);
	return -2;
}

//! looking for the right channel
TStjFrameWAVFile * FrameWAVmoduleFindChannel (
		TStjFrameWAVmodule *	pM,	//!< the module
		int			channel //!< channel to search for
		) {
	TStjFrameWAVFile * pWF = pM->pChannels;
	int i;
	for (i = 0; i < pM->number;i++) {
		if (pWF->channel == channel) return pWF;
		pWF++;
	}
	return NULL;
}

//! reads a sample from a channel
int FrameWAVmoduleGetInput (
		TStjFrameWAVmodule *	pM,		//!<(in) the module
		int			channel,		//!<(in) the channel
		int			sampleNumber,	//!<(in) number of samples to be read
		float *			pFrame		//!<(out) pointer to the sample buffer witch is filled
	) {
	TStjFrameWAVFile * pWF = FrameWAVmoduleFindChannel(pM,channel);
	// channel not found
	if (!pWF) {
		return -1;
	}
	// channel is not an input
	if (!pWF->isInput) {
		return -2;
	}

	// check the temp buffer size
	if (pWF->frameAmount < sampleNumber) {
		pWF->pFrameBuffer = realloc(pWF->pFrameBuffer,sizeof(float)*pWF->
```

```
                    fileChannels*sampleNumber);
                if (!pWF->pFrameBuffer) {
                        return -3;
                }
                pWF->frameAmount = sampleNumber;
        }

        // fill the buffer
        int am = (int) sf_readf_float (pWF->pSndF,pWF->
           pFrameBuffer, sampleNumber);

        if (am <= 0) {
                return -4;
        }

        // copy buffer
        int i;
        float * pD = pFrame;
        float * pS = pWF->pFrameBuffer;
        // add the offset so we start at the right pos
           at the frame
        pS += pWF->activeChannel;
        for (i = 0; i < am; i++) {
                *pD = *pS;
                pD++;
                pS+= pWF->fileChannels;
        }
        // and fill the rest with 0.
        for (;i < sampleNumber; i++) {
                *pD = 0.0f;
                pD++;
        }
        return 0;
}

//! writes a sample to an output
int FrameWAVmoduleSetOutput (
                TStjFrameWAVmodule *    pM,
                                   //!<(in) the module
                int
                           channel,                //!<(
                   in) the channel
                int
                           sampleNumber,   //!<(in)
                   number of samples to be write
                float *
                   pFrame                  //!<(in)
                   pointer to the sample buffer witch is
                    filled
        ) {
        TStjFrameWAVFile * pWF =
           FrameWAVmoduleFindChannel(pM,channel);
        // channel not found
        if (!pWF) {
                return -1;
```

```
        }
        // channel is not an output
        if (pWF->isInput) {
                return -2;
        }
        // the last samples at the frame are new
        if (sampleNumber != sf_write_float (pWF->pSndF,
           pFrame, sampleNumber)) return -3;
        return 0;
}
```

3.1.12 libsndfile overlapped frame based

includes:

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

code:

```
// ============================================================
// descr: some helpers for the lib-snd-file group
// author: Stefan Jaritz
//
// the lib is the "libsndfile-1"
// ============================================================

// defines

typedef struct SStjFrameOverlappedWAVOpenInfo {
        unsigned int    channelNumber;
           //!< logical channel
        char *                  szFileName;
                                //!< name of the file
        unsigned int    isInput;
                        //!< if <>0 then this file is an
           input

        unsigned int    sampleRate;
                        //!< the sample rate
        unsigned int    fileChannelNumber;
           //!< number of the channel at the file (
           starting at 1)

        unsigned int    frameSize;
                        //!< number of samples per frame
```

```
	unsigned int	overlappingNumber;	//!< the number of samples kept in the frame at reading/writing
} TStjFrameOverlappedWAVOpenInfo;

typedef struct SStjFrameOverlappedWAVFile {
	unsigned int	channel;		//!< the channel
	SNDFILE *			pSndF;		//!< the file
	unsigned int	isInput;		//!< if 0 the channel is an input channel

	float *				pRB;		//!< the ring buffer storing/writing the samples
	unsigned int	rbFrameSize;	//!< the size of the frames at the ring buffer
	unsigned int	FrameSize;		//!< amount of samples at the frame
	unsigned int	nMax;			//!< number of elements at the ring buffer
	unsigned int	n;				//!< actual element (for inputs)/ the offset (output)
} TStjFrameOverlappedWAVFile;

typedef struct SStjFrameOverlappedWAVmodule {
	TStjFrameOverlappedWAVFile *	pChannels;
	unsigned int			number;
} TStjFrameOverlappedWAVmodule;

//! close the set of wav files
void FrameOverlappedWAVmoduleExit ( TStjFrameOverlappedWAVmodule * pM);

//! initis a set of wav files for writing / reading
int FrameOverlappedWAVmoduleInit (
				int				number,		//!<(in) amount of files
				TStjFrameOverlappedWAVOpenInfo *	pWAVFiles,	//!<(in) file description
				TStjFrameOverlappedWAVmodule *	pModul		//!<(in/out) modul descriptor witch is filled
	);

//! looking for the right channel
TStjFrameOverlappedWAVFile * FrameOverlappedWAVmoduleFindChannel (
```

```
                TStjFrameOverlappedWAVmodule *  pM,           //!< the module
                int                             channel       //!< channel to search for
        );

//! reads a sample from a channel
int FrameOverlappedWAVmoduleGetInput (
                TStjFrameOverlappedWAVmodule *  pM,           //!<(in) the module
                int                             channel,      //!<(in) the channel
                float *                         pFrame        //!<(out) pointer to the sample buffer witch is filled
        );

//! writes a sample to an output
int FrameOverlappedWAVmoduleSetOutput (
                TStjFrameOverlappedWAVmodule *  pM,           //!<(in) the module
                int                             channel,      //!<(in) the channel
                float *                         pFrame        //!<(in) pointer to the sample buffer witch is filled
        );

// functions

//! close the set of wav files
void FrameOverlappedWAVmoduleExit ( TStjFrameOverlappedWAVmodule * pM) {
        //1. close all sound files
        int i;

        TStjFrameOverlappedWAVFile * pWF = pM->pChannels;
        for (i = 0; i < pM->number;i++) {
                sf_close(pWF->pSndF);
                free (pWF->pRB);
                pWF++;
        }
        //2. free array
        free (pM->pChannels);

        //3. set all vars of the struct to default
        pM->number = 0;
        pM->pChannels = NULL;
}


//! initis a set of wav files for writing / reading
```

```
int FrameOverlappedWAVmoduleInit (
                int number,          //!<(in) amount of files
                TStjFrameOverlappedWAVOpenInfo * pWAVFiles,       //!<(in) file description
                TStjFrameOverlappedWAVmodule *  pModul          //!<(in/out) modul descriptor witch is filled
        ) {

        pModul->pChannels = malloc (sizeof(TStjFrameOverlappedWAVFile) * number);
        if (!pModul->pChannels) {
                return -1;
        }
        memset (pModul->pChannels,0,sizeof(TStjFrameOverlappedWAVFile) * number);
        pModul->number = number;

        int                                     i;
        SF_INFO                                 info;
        TStjFrameOverlappedWAVFile *    pWF = pModul->pChannels;

        for (i = 0;i < number;i++) {
                // open sndfile interface
                memset (&info,0,sizeof(SF_INFO));
                if (pWAVFiles[i].isInput) {
                        pWF->pSndF = sf_open (pWAVFiles[i].szFileName,SFM_READ, &info) ;
                } else {
                        info.samplerate = pWAVFiles[i].sampleRate;
                        info.channels = 1;
                        info.format = SF_FORMAT_WAV | SF_FORMAT_FLOAT;
                        pWF->pSndF = sf_open (pWAVFiles[i].szFileName,SFM_WRITE, &info) ;
                }
                if (!pWF->pSndF) {
                        goto error;
                }
                pWF->isInput = (unsigned int) pWAVFiles[i].isInput;
                pWF->channel = (unsigned int) pWAVFiles[i].channelNumber;

                // prepare buffers
                pWF->FrameSize = pWAVFiles[i].frameSize;
                if (pWAVFiles[i].frameSize % pWAVFiles[i].overlappingNumber) {
```

```c
                        goto error;
                }
                pWF->nMax = pWAVFiles[i].frameSize / pWAVFiles[i].overlappingNumber;
                pWF->rbFrameSize = pWAVFiles[i].overlappingNumber;
                // if it's an input we need a memory for the old frames witch are read before
                if (pWAVFiles[i].isInput) {
                        pWF->pRB = malloc(sizeof(float)*pWAVFiles[i].frameSize);
                        if (!pWF->pRB){
                                goto error;
                        }
                        memset(pWF->pRB,0,sizeof(float)*pWAVFiles[i].frameSize);
                        pWF->n = 0;
                } else {
                        pWF->pRB = NULL;
                        // the n is used as offset
                        pWF->n = pWAVFiles[i].frameSize - pWAVFiles[i].overlappingNumber;
                }
                pWF++;
        }
        return 0;
error:
        FrameOverlappedWAVmoduleExit(pModul);
        return -2;
}

//! looking for the right channel
TStjFrameOverlappedWAVFile * FrameOverlappedWAVmoduleFindChannel (
                TStjFrameOverlappedWAVmodule *  pM,         //!< the module
                int                             channel     //!< channel to search for
                ) {
        TStjFrameOverlappedWAVFile * pWF = pM->pChannels;
        int i;
        for (i = 0; i < pM->number;i++) {
                if (pWF->channel == channel) return pWF;
                pWF++;
        }
        return NULL;
}

//! reads a sample from a channel
int FrameOverlappedWAVmoduleGetInput (
                TStjFrameOverlappedWAVmodule *  pM,                 //!<(in) the module
                int
```

```
                        channel,            //!<(in) the channel
                float *
                        pFrame              //!<(out) pointer to the sample buffer witch is filled
        ) {
        TStjFrameOverlappedWAVFile * pWF = FrameOverlappedWAVmoduleFindChannel(pM, channel);
        // channel not found
        if (!pWF) {
                return -1;
        }
        // channel is not an input
        if (!pWF->isInput) {
                return -2;
        }

        // check the offset at the ringbuffer
        if (pWF->n >= pWF->nMax) pWF->n = 0;

        int offset;
        int idx = pWF->rbFrameSize * pWF->n;
        // fill the buffer

        sf_count_t am = sf_readf_float (pWF->pSndF,&pWF->pRB[idx], (sf_count_t) pWF->rbFrameSize);

        if (am == 0) return -3;
        if (am != (sf_count_t) pWF->rbFrameSize) {
                offset = (sf_count_t) am;
                // calc the missing samples
                am = (sf_count_t) pWF->rbFrameSize - am;
                // set the buffer to 0
                memset(&pWF->pRB[idx+offset],0,sizeof(float)*am);
        }
        // build the frame
        // on pos n is the newest frame
        // n+1 is oldest frame
        int nStart = pWF->n + 1;
        int amount;

        // copy all frames left from the newest block till the wrapping
        amount = pWF->nMax - nStart;
        if (amount) {
                memcpy(pFrame,&pWF->pRB[nStart*pWF->rbFrameSize],sizeof(float)*amount*pWF->rbFrameSize);
        }
        nStart = amount;
        // wrap and copy the rest of it
        amount = pWF->nMax - amount;
        memcpy(&pFrame[nStart*pWF->rbFrameSize],pWF->pRB
```

```
    ,sizeof(float)*amount*pWF->rbFrameSize);
        // remember that we received a frame(with wrapping)
        pWF->n++;
        return 0;
}

//! writes a sample to an output
int FrameOverlappedWAVmoduleSetOutput (
                TStjFrameOverlappedWAVmodule *  pM,                     //!<(in) the module
                int                         channel,        //!<(in) the channel
                float *                     pFrame          //!<(in) pointer to the sample buffer witch is filled
        ) {
        TStjFrameOverlappedWAVFile * pWF = FrameOverlappedWAVmoduleFindChannel(pM, channel);
        // channel not found
        if (!pWF) {
                return -1;
        }
        // channel is not an output
        if (pWF->isInput) {
                return -2;
        }
        // the last samples at the frame are new
        if (pWF->rbFrameSize != sf_write_float (pWF->pSndF,&pFrame[pWF->n] , pWF->rbFrameSize)) return -3;

        return 0;
}

// ======================================================================
```

3.1.13 libsndfile sample based

includes:

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

code:

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

```
// author: Stefan Jaritz
//
// the lib is the "libsndfile-1"
// ================================

typedef struct SStjWAVOpenInfo {
        unsigned int    channelNumber;          //!< logical channel
        char *                  szFileName;                     //!< name of the file
        unsigned int    isInput;                //!< if <>0 then this file is an input
        unsigned int    sampleRate;             //!< the sample rate
        unsigned int    fileChannelNumber;      //!< number of the channel at the file (starting at 1)
        unsigned int    sampleAmountCacheSize;  //!< amount of samples cached before writing or/at reading
} TStjWAVOpenInfo;

typedef struct SStjWAVFile {
        unsigned int    channel;        //!< the channel
        SNDFILE *                       pSndF;          //!< the file
        unsigned int    isInput;        //!< if <> 0 the channel is an input channel
        unsigned int    channelsAmount;         //!< the amount of channels
        unsigned int    sampleAmount;           //!< the amount of samples at the cache
        unsigned int    maxSampleAmount;        //!< the maximum sample amount
        unsigned int    dataOffset;     //!< the offset of the channel at the sample cache
        float *                 pCache;         //!< the sample cache
        float *                 pD;                     //!< pointer to the data
} TStjWAVFile;


typedef struct SStjWAVmodule {
        TStjWAVFile *   pChannels;
        unsigned int    number;
} TStjWAVmodule;

// fill the wav sample cache
int WAVmoduleFillCache (TStjWAVFile * pWF) {
        sf_count_t amount = (sf_count_t) pWF->maxSampleAmount;
```

```
        // read a data frame(s)
        amount = sf_readf_float (pWF->pSndF,pWF->pCache,
            amount);
        if (amount < 1) return -1;
        // reset data pointers and indexes
        pWF->pD = pWF->pCache;
        pWF->sampleAmount = (unsigned int) amount;
        return 0;
}

// flushes the wav sample cache to disk
int WAVmoduleFlushCache (TStjWAVFile * pWF) {
        if (!pWF->sampleAmount) return 0;
        sf_count_t amount = (sf_count_t) pWF->
           sampleAmount;
        if (amount != sf_write_float (pWF->pSndF,pWF->
           pCache , amount)) return -1;
        pWF->pD = pWF->pCache;
        pWF->sampleAmount = 0;
        return 0;
}


//! close the set of wav files
void WAVmoduleExit (TStjWAVmodule * pM) {
        //1. close all sound files
        int i;

        TStjWAVFile * pWF = pM->pChannels;
        for (i = 0; i < pM->number;i++) {
                // if it is an output flush the samples
                   to disk
                if (!pWF->isInput) {
                        WAVmoduleFlushCache(pWF);
                }
                sf_close(pWF->pSndF);
                if (pWF->pCache) {
                        free (pWF->pCache);
                }
                pWF++;
        }
        //2. free array
        free (pM->pChannels);

        //3. set all vars of the struct to default
        pM->number = 0;
        pM->pChannels = NULL;
}


//! initis a set of wav files for writing / reading
int WAVmoduleInit (
                int                                 number,
                           //!<(in) amount of files
                TStjWAVOpenInfo *       pWAVFiles,
```

```
                    //!<(in) file description
                TStjWAVmodule *          pModul
                    //!<(in/out) modul descriptor witch
                    is filled
        ) {

        pModul->pChannels = malloc (sizeof(TStjWAVFile)
            * number);
        if (!pModul->pChannels) {
                return -1;
        }
        memset (pModul->pChannels,0,sizeof(TStjWAVFile)
            * number);
        pModul->number = number;

        int                         i;
        SF_INFO                     info;
        TStjWAVFile *   pWF = pModul->pChannels;
        size_t                      cacheSize;

        for (i = 0;i < number;i++) {
                memset (&info,0,sizeof(SF_INFO));
                if (pWAVFiles[i].isInput) {
                        pWF->pSndF = sf_open (pWAVFiles[
                            i].szFileName,SFM_READ, &info
                            ) ;
                } else {
                        info.samplerate = pWAVFiles[i].
                            sampleRate;
                        info.channels = 1;
                        info.format = SF_FORMAT_WAV |
                            SF_FORMAT_FLOAT;
                        pWF->pSndF = sf_open (pWAVFiles[
                            i].szFileName,SFM_WRITE, &
                            info) ;
                }
                if (!pWF->pSndF) {
                        goto error;
                }
                pWF->isInput = (unsigned int) pWAVFiles[
                    i].isInput;
                pWF->channel = (unsigned int) pWAVFiles[
                    i].channelNumber;
                pWF->channelsAmount = (unsigned int)
                    info.channels;
                pWF->dataOffset = (unsigned int)
                    pWAVFiles[i].fileChannelNumber;
                pWF->maxSampleAmount = (unsigned int)
                    pWAVFiles[i].sampleAmountCacheSize;
                cacheSize =  pWF->channelsAmount * pWF->
                    maxSampleAmount * sizeof(float);
                pWF->pCache = malloc (cacheSize);
                if (!pWF->pCache) goto error;
                memset (pWF->pCache,0,cacheSize);
                pWF->sampleAmount = 0;
```

```
			// now fill the cache with data
			if (pWF->isInput) {
				if (WAVmoduleFillCache(pWF)) goto error;
			} else {
				pWF->pD = pWF->pCache;
			}
			pWF++;
		}
		return 0;
error:
		WAVmoduleExit(pModul);
		return -2;
}

//! looking for the right channel
TStjWAVFile * WAVmoduleFindChannel (
			TStjWAVmodule * pM,		//!< the module
			int				channel		//!< channel to search for
			) {
		TStjWAVFile * pWF = pM->pChannels;
		int i;
		for (i = 0; i < pM->number;i++) {
			if (pWF->channel == channel) return pWF;
			pWF++;
		}
		return NULL;
}

//! reads a sample from a channel
int WAVmoduleGetInput (
			TStjWAVmodule * pM,		//!<(in) the module
			int				channel,		//!<(in) the channel
			float *			pSample		//!<(out) pointer to the sample buffer witch is filled
		) {
		TStjWAVFile * pWF = WAVmoduleFindChannel(pM, channel);
		// channel not found
		if (!pWF) {
			return -1;
		}
		// channel is not an input
		if (!pWF->isInput) {
			return -2;
		}
		// if the cache is empty then fill it
		if (!pWF->sampleAmount) {
			if (WAVmoduleFillCache(pWF)) return -3;
		}
```

```
        // copy sample
        *pSample = pWF->pD[pWF->dataOffset];
        pWF->pD += pWF->channelsAmount;
        // now there is one sample less at the buffer
        pWF->sampleAmount--;
        return 0;
}

//! writes a sample to an output
int WAVmoduleSetOutput (
                TStjWAVmodule * pM,
                   //!<(in) the module
                int                                 channel,
                           //!<(in) the channel
                float                       sample
                   //!<(in) pointer to the sample buffer
                    witch is filled
        ) {
        TStjWAVFile * pWF = WAVmoduleFindChannel(pM,
           channel);
        // channel not found
        if (!pWF) {
                return -1;
        }
        // channel is not an output
        if (pWF->isInput) {
                return -2;
        }
        // if the cache is full then flush it
        if (pWF->sampleAmount == pWF->maxSampleAmount) {
                if (WAVmoduleFlushCache(pWF)) return -3;
        }
        // now store the value
        pWF->pD[pWF->dataOffset] = sample;
        pWF->pD += pWF->channelsAmount;
        pWF->sampleAmount++;
        return 0;
}
```

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:

```
// ===============================================
// generic vector interface for integer and
// rational types based on the std. C lib
// ===============================================

// -----------------------------------------------
// generic integer type
// -----------------------------------------------

// transforms the integer to a 'unique form'
inline uint32_t APgenericInteger_ToUniform (uint32_t v)
   {
        uint32_t u;
        u = v & 0xFF;
        u <<= 8;
        v >>= 8;
        u |= (v & 0xFF);
        u <<= 8;
        v >>= 8;
        u |= (v & 0xFF);
        u <<= 8;
        v >>= 8;
        u |= (v & 0xFF);
        return u;
}

// inverse transformation of the unique integer to the '
   local form'
inline uint32_t APgenericInteger_FromUniform (uint32_t v
   ) {
        uint32_t u;
        u = v & 0xFF;
        u <<= 8;
        v >>= 8;
        u |= v & 0xFF;
        u <<= 8;
        v >>= 8;
        u |= v & 0xFF;
        u <<= 8;
        v >>= 8;
        u |= v & 0xFF;
        return u;
}
```

```c



// type for a generic vector based int type
typedef struct SAPgenericIntegerVector {
        int32_t *       pVal;
        int                     num;
} TAPgenericIntegerVector;

// create int vector
TAPgenericIntegerVector * APgenericIntegerVector_create
   (int num) {
        int i;
        // alloc memory
        TAPgenericIntegerVector * pI = malloc (sizeof(
           TAPgenericIntegerVector));
        if (!pI) return NULL;
        pI->pVal = malloc (sizeof(int32_t)*num);
        if (!pI->pVal) {
                free (pI);
                return NULL;
        }
        pI->num = num;
        // set memory
        for (i = 0; i< num; i++) {
                pI->pVal[i] = 0;
        }
        return pI;
}

// destroy int vector
void APgenericIntegerVector_destroy (
   TAPgenericIntegerVector * pV) {
        if (pV) {
                free(pV->pVal);
                free(pV);
        }
}

// resize int vector
int APgenericIntegerVector_resize (
   TAPgenericIntegerVector * pV, int newNum) {
        int i;

        if (pV->num == newNum) return 0;
        free (pV->pVal);
        pV->pVal = malloc (sizeof(int32_t)*newNum);
        if (!pV->pVal) return -1;
        pV->num = newNum;
        // set memory
        for (i = 0; i< newNum; i++) {
                pV->pVal[i] = 0;
        }
        return 0;
}
```

```
// assign a = values
int APgenericIntegerVector_assignConst (
    TAPgenericIntegerVector * pa, int iStart, int num,
    int32_t * pVals) {
        int i = iStart;
        int imax = i + num;
        int32_t * pD = pa->pVal + iStart;
        if (imax > pa->num) imax = pa->num;

        for (; i < imax; i++) {
                *pD = *pVals;
                pD++;
                pVals++;
        }
        return 0;
}

// assign a = b
int APgenericIntegerVector_assign (
    TAPgenericIntegerVector * pa, TAPgenericIntegerVector
     * pb) {
        int imax = pa->num;
        int i;
        for (i = 0; i < imax; i++) {
                pa->pVal[i] = pb->pVal[i];
        }
        return 0;
}


// add c = a + b
int APgenericIntegerVector_add (TAPgenericIntegerVector
    * pa, TAPgenericIntegerVector * pb,
    TAPgenericIntegerVector * pc) {
        int imax = pc->num;
        int i;
        for (i = 0; i < imax; i++) {
                pc->pVal[i] = pa->pVal[i] + pb->pVal[i];
        }
        return 0;
}

// sub c = a - b
int APgenericIntegerVector_sub (TAPgenericIntegerVector
    * pa, TAPgenericIntegerVector * pb,
    TAPgenericIntegerVector * pc) {
        int imax = pc->num;
        int i;
        for (i = 0; i < imax; i++) {
                pc->pVal[i] = pa->pVal[i] - pb->pVal[i];
        }
        return 0;
}
```

```
// mul c = a * b
int APgenericIntegerVector_mul (TAPgenericIntegerVector * pa, TAPgenericIntegerVector * pb, TAPgenericIntegerVector * pc) {
        int imax = pc->num;
        int i;
        for (i = 0; i < imax; i++) {
                pc->pVal[i] = pa->pVal[i] * pb->pVal[i];
        }
        return 0;
}

// div c = a * b
int APgenericIntegerVector_div (TAPgenericIntegerVector * pa, TAPgenericIntegerVector * pb, TAPgenericIntegerVector * pc) {
        int imax = pc->num;
        int i;
        for (i = 0; i < imax; i++) {
                pc->pVal[i] = pa->pVal[i] / pb->pVal[i];
        }
        return 0;
}

// modulo c = a mod b
int APgenericIntegerVector_mod (TAPgenericIntegerVector * pa, TAPgenericIntegerVector * pb, TAPgenericIntegerVector * pc) {
        int imax = pc->num;
        int i;
        for (i = 0; i < imax; i++) {
                pc->pVal[i] = pa->pVal[i] % pb->pVal[i];
        }
        return 0;
}

// cmp Element cmp(a[i],b[j]):
//              res = -1: a < b
//              res = 1 : a > b
//              res = 0: a = b
int APgenericIntegerVector_cmpEle (TAPgenericIntegerVector * pa, TAPgenericIntegerVector * pb, int ia, int ib) {
        int zwv = pa->pVal[ia] - pb->pVal[ib];
        if (!zwv) return 0;
        if (zwv < 0) return -1;
        return 1;
}


// -----------------------------------------------
// generic rational type
// -----------------------------------------------

// type for a generic vector based float type
```

```
typedef struct SAPgenericRationalVector {
        float *         pVal;
        int                     num;
} TAPgenericRationalVector;

// create float vector
TAPgenericRationalVector *
   APgenericRationalVector_create (int num) {
        int i;
        // alloc memory
        TAPgenericRationalVector * pR = malloc (sizeof(
           TAPgenericRationalVector));
        if (!pR) return NULL;
        pR->pVal = malloc (sizeof(float)*num);
        if (!pR->pVal) {
                free (pR);
                return NULL;
        }
        pR->num = num;
        // set memory
        for (i = 0; i< num; i++) {
                pR->pVal[i] = 0.f;
        }
        return pR;
}

// destroy float vector
void APgenericRationalVector_destroy (
   TAPgenericRationalVector * pV) {
        if (pV) {
                free(pV->pVal);
                free(pV);
        }
}

// resize float vector
int APgenericRationalVector_resize (
   TAPgenericRationalVector * pV, int newNum) {
        int i;

        if (pV->num == newNum) return 0;
        free (pV->pVal);
        pV->pVal = malloc (sizeof(float)*newNum);
        if (!pV->pVal) return -1;
        pV->num = newNum;
        // set memory
        for (i = 0; i< newNum; i++) {
                pV->pVal[i] = 0.f;
        }
        return 0;
}

// assign a = values
int APgenericRationalVector_assignConst (
   TAPgenericRationalVector * pa, int iStart, int num,
```

```
    float * pVals) {
        int i = iStart;
        int imax = i + num;
        float * pD = pa->pVal + iStart;
        if (imax > pa->num) imax = pa->num;

        for (; i < imax; i++) {
                *pD = *pVals;
                pD++;
                pVals++;
        }
        return 0;
}

// assign a = b
int APgenericRationalVector_assign (
    TAPgenericRationalVector * pa,
    TAPgenericRationalVector * pb) {
        int imax = pa->num;
        int i;
        for (i = 0; i < imax; i++) {
                pa->pVal[i] = pb->pVal[i];
        }
        return 0;
}


// add c = a + b
int APgenericRationalVector_add (
    TAPgenericRationalVector * pa,
    TAPgenericRationalVector * pb,
    TAPgenericRationalVector * pc) {
        int imax = pc->num;
        int i;
        for (i = 0; i < imax; i++) {
                pc->pVal[i] = pa->pVal[i] + pb->pVal[i];
        }
        return 0;
}

// sub c = a - b
int APgenericRationalVector_sub (
    TAPgenericRationalVector * pa,
    TAPgenericRationalVector * pb,
    TAPgenericRationalVector * pc) {
        int imax = pc->num;
        int i;
        for (i = 0; i < imax; i++) {
                pc->pVal[i] = pa->pVal[i] - pb->pVal[i];
        }
        return 0;
}

// mul c = a * b
int APgenericRationalVector_mul (
```

```
    TAPgenericRationalVector * pa,
    TAPgenericRationalVector * pb,
    TAPgenericRationalVector * pc) {
        int imax = pc->num;
        int i;
        for (i = 0; i < imax; i++) {
                pc->pVal[i] = pa->pVal[i] * pb->pVal[i];
        }
        return 0;
}

// div c = a * b
int APgenericRationalVector_div (
    TAPgenericRationalVector * pa,
    TAPgenericRationalVector * pb,
    TAPgenericRationalVector * pc) {
        int imax = pc->num;
        int i;
        for (i = 0; i < imax; i++) {
                pc->pVal[i] = pa->pVal[i] / pb->pVal[i];
        }
        return 0;
}

// modulo c = a mod b
int APgenericRationalVector_mod (
    TAPgenericRationalVector * pa,
    TAPgenericRationalVector * pb,
    TAPgenericRationalVector * pc) {
        int imax = pc->num;
        int i;
        for (i = 0; i < imax; i++) {
                pc->pVal[i] = fmodf(pa->pVal[i],pb->pVal
                    [i]);
        }
        return 0;
}

// cmp Element cmp(a[i],b[j]):
//              res = -1: a < b
//              res = 1 : a > b
//              res = 0: a = b
int APgenericRationalVector_cmpEle (
    TAPgenericRationalVector * pa,
    TAPgenericRationalVector * pb, int ia, int ib) {
        float zwv = pa->pVal[ia] - pb->pVal[ib];
        if (zwv == 0.f) return 0;
        if (zwv < 0.f) return -1;
        return 1;
}
```

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:

```
// ================================
// variable implementation for a rational number(var id
   = 1)
// ================================

// updates a variable the AP
int HALimpl_1_recvUpdate (void * pVarData, void *
   pMsgData) {
        uint32_t * pD = (uint32_t *) pMsgData;
        TAPgenericRationalVector * pIV = pVarData;

        // at the first possion at the message is the
           global var index
        pD++; // skip it (it's the varindex)
        // and now we are at amount of values
        APendianConversation32Bit(pD,eAP_littleEndian);
        int imax = (int) *((int32_t *)pD);
        int i;

        if (APgenericRationalVector_resize(pIV,imax)) {
                return -1;
        }

        pD++;

        for (i = 0; i < imax; i++) {
                pIV->pVal[i] = *((float *)pD);
                pD++;
        }
        return 0;
}
// create a new variable
void * HALimpl_1_create (unsigned int numberOfElements)
```

```
{
        return APgenericRationalVector_create((int) numberOfElements);
}
// updates the vars at the other APs
int HALimpl_1_sendUpdate (void * pVarData, const void * pDrv, uint32_t receiver, uint32_t mNum, int32_t i) {
        TAPgenericRationalVector * pRV = pVarData;
        TAPMsgDrv * pMD = (TAPMsgDrv *) pDrv;
        int32_t dataAmount = (int32_t) 1 + pRV->num;
        int32_t dummy;
        // send header
        pMD->pfkt_updateVariable(pMD->pDrvData, receiver, mNum, i, dataAmount);
        // send amount of elements
        dummy = (int32_t) pRV->num;
        pMD->pfkt_sendInteger32(pMD,1,&dummy);
        // send elements
        pMD->pfkt_sendFloat32(pMD,pRV->num,pRV->pVal);
        return 0;
}
// decode data for the HAL functions
void * HALimpl_1_decodeData (void * pVarData) {
        return pVarData;
}
// delete the variable
void HALimpl_1_delete (void * pVarData) {
        APgenericRationalVector_destroy(pVarData);
}
```

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:

```
// ===============================
// variable implementation for a integer number(var id = 2)
// ===============================

// updates a variable the AP
```

```c
int HALimpl_2_recvUpdate (void * pVarData, void * pMsgData) {
        uint32_t * pD = (uint32_t *) pMsgData;
        TAPgenericIntegerVector * pIV = pVarData;

        // at the first possion at the message is the global var index
        pD++; // skip it (it's the varindex)
        // and now we are at amount of values
        APendianConversation32Bit(pD,eAP_littleEndian);
        int imax = (int) *((int32_t *)pD);

        int i;

        if (APgenericIntegerVector_resize(pIV,imax)) {
                return -1;
        }

        pD++;

        for (i = 0; i < imax; i++) {
                APendianConversation32Bit(pD, eAP_littleEndian);
                pIV->pVal[i] = *((int32_t *)pD);
                pD++;
        }
        return 0;
}

// create a new variable
void * HALimpl_2_create (unsigned int numberOfElements) {
        return APgenericIntegerVector_create((int) numberOfElements);
}
// updates the vars at the other APs
int HALimpl_2_sendUpdate (void * pVarData, const void * pDrv, uint32_t receiver, uint32_t mNum, int32_t i) {
        TAPgenericIntegerVector * pIV = pVarData;
        TAPMsgDrv * pMD = (TAPMsgDrv *) pDrv;
        int32_t dataAmount = 1 + pIV->num;
        int32_t dummy;
        int indx;
        // send header
        pMD->pfkt_updateVariable(pMD->pDrvData, receiver, mNum, i, dataAmount);
        // send amount of elements
        dummy = (int32_t) pIV->num;
        pMD->pfkt_sendInteger32(pMD,1,&dummy);
        // send elements
        for (indx = 0; indx < pIV->num; indx++) {
                pMD->pfkt_sendInteger32(pMD,1,&pIV->pVal[indx]);
        }
        return 0;
```

```
}
// decode data for the HAL functions
void * HALimpl_2_decodeData (void * pVarData) {
        return pVarData;
}
// delete the variable
void HALimpl_2_delete (void * pVarData) {
        APgenericIntegerVector_destroy(pVarData);
}
```

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 libary

includes:

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

code:

```
// ===============================
// variable implementation for a complex number(var id =
   4)
// ===============================

// the type for the complex data struct
typedef struct SAPvarComplex {
        fftwf_complex * pC;
        int                             num;
} TAPvarComplex;

// transmit the data
void HALimpl_4_init (void * pVarData, int frameNumber,
   void * pData) {

}

// start the init process
void HALimpl_4_initStart (void * pVarData, int
   frameNumber, int bigEndian) {

}

// get the frame Number and size
int HALimpl_4_frameGetNumber (void * pVarData, int *
   pframeSize) {
        TAPvarComplex * pC = (TAPvarComplex *) pVarData;
```

```
        *pframeSize = pC->num * sizeof(fftwf_complex);
        return 1;
}

// end init process
void HALimpl_4_initEnd (void * pVarData) {

}

// create a new variable
void * HALimpl_4_create (unsigned int numberOfElements) {
        if (!numberOfElements) return NULL;

        TAPvarComplex * pC;
        pC = malloc(sizeof(TAPvarComplex));
        if (!pC) return NULL;

        pC->pC = fftwf_malloc(sizeof(fftwf_complex)*numberOfElements);
        if (!(pC->pC)) {
                free (pC);
                return NULL;
        }
        pC->num = numberOfElements;

        memset (pC->pC,0,sizeof(fftwf_complex)*numberOfElements);
        return pC;
}

// fill the frame with data
void HALimpl_4_frameFill (void * pVarData, int frameNumber, void * pFrame) {

}

// decode data for the HAL functions
void * HALimpl_4_decodeData (void * pVarData) {
        TAPvarComplex * pC = (TAPvarComplex *) pVarData;
        return pC->pC;
}

// delete the variable
void HALimpl_4_delete (void * pVarData) {
        TAPvarComplex * pC = (TAPvarComplex *) pVarData;
        if (pC) {
                if (pC->pC) fftwf_free(pC->pC);
                free (pC);
        }
}
```

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:

```
// ================================
// variable implementation for FFT or IFFT sturcture(var id = 30)
// ================================

// transmit the data
void HALimpl_30_init (void * pVarData, int frameNumber, void * pData) {

}
// start the init process
void HALimpl_30_initStart (void * pVarData, int frameNumber, int bigEndian) {

}
// get the frame Number and size
int HALimpl_30_frameGetNumber (void * pVarData, int * pframeSize) {
return 0;
}
// end init process
void HALimpl_30_initEnd (void * pVarData) {

}
// create a new variable
void * HALimpl_30_create (unsigned int numberOfElements) {
        return NULL;
}
// fill the frame with data
void HALimpl_30_frameFill (void * pVarData, int frameNumber, void * pFrame) {

}
// decode data for the HAL functions
void * HALimpl_30_decodeData (void * pVarData) {
        return pVarData;
}
// delete the variable
```

```
void HALimpl_30_delete (void * pVarData) {
        if (pVarData) {
                fftwf_destroy_plan(pVarData);
        }
}
```

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:

```
// ================================
// variable implementation for panel UI(var id = 100)
// ================================

typedef struct SMSP430_panel {
        unsigned char dummy;
} TMSP430_panel;

typedef struct SMSP430_panelVec {
        TMSP430_panel * pP;
        int num;
} TMSP430_panelVec;

// updates a variable the AP
int HALimpl_100_recvUpdate (void * pVarData, void *
    pMsgData) {
return -1;
}
// create a new variable
void * HALimpl_100_create (unsigned int numberOfElements
    ) {
        TMSP430_panelVec * pPV;

        pPV = malloc(sizeof(TMSP430_panelVec));
        if (!pPV) return NULL;

        pPV->pP = malloc(sizeof(TMSP430_panel)*
            numberOfElements);
        if (!pPV->pP) {
                free(pPV);
                return NULL;
```

```
        }
        pPV->num = (int) numberOfElements;
        return pPV;
}
// updates the vars at the other APs
int HALimpl_100_sendUpdate (void * pVarData, const void * pDrv, uint32_t receiver, uint32_t mNum, int32_t i) {
return -1;
}
// decode data for the HAL functions
void * HALimpl_100_decodeData (void * pVarData) {
        return pVarData;
}
// delete the variable
void HALimpl_100_delete (void * pVarData) {
        TMSP430_panelVec * pPV = (TMSP430_panelVec *) pVarData;
        if (pPV) {
                if (pPV->pP) free(pPV->pP);
                free (pPV);
        }
}
```

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:

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

typedef struct SMSP430_button {
        unsigned char port1Mask;
} TMSP430_button;

typedef struct SMSP430_buttonVec {
        TMSP430_button * pB;
        int num;
} TMSP430_buttonVec;

```

```
// updates a variable the AP
int HALimpl_101_recvUpdate (void * pVarData, void *
    pMsgData) {
return -1;
}
// create a new variable
void * HALimpl_101_create (unsigned int numberOfElements
    ) {
        TMSP430_buttonVec * pBV;

        pBV = malloc(sizeof(TMSP430_buttonVec));
        if (!pBV) return NULL;

        pBV->pB = malloc(sizeof(TMSP430_button)*
            numberOfElements);
        if (!pBV->pB) {
                free(pBV);
                return NULL;
        }

        pBV->num = (int) numberOfElements;
        memset (pBV->pB, 0, sizeof(TMSP430_button)*
            numberOfElements);
        return pBV;
}
// updates the vars at the other APs
int HALimpl_101_sendUpdate (void * pVarData, const void
    * pDrv, uint32_t receiver, uint32_t mNum, int32_t i)
    {
return -1;
}
// decode data for the HAL functions
void * HALimpl_101_decodeData (void * pVarData) {
return pVarData;
}
// delete the variable
void HALimpl_101_delete (void * pVarData) {
        TMSP430_buttonVec * pBV = (TMSP430_buttonVec *)
            pVarData;
        if (pBV) {
                if (pBV->pB) free(pBV->pB);
                free (pBV);
        }
}
```

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:

```
// ================================
// variable implementation for led UI(var id = 102)
// ================================

typedef struct SMSP430_LED {
        unsigned char port3mask;
} TMSP430_LED;

typedef struct SMSP430_LEDvec {
        TMSP430_LED * pL;
        int num;
} TMSP430_LEDvec;

// updates a variable the AP
int HALimpl_102_recvUpdate (void * pVarData, void *
    pMsgData) {
return -1;
}
// create a new variable
void * HALimpl_102_create (unsigned int numberOfElements
    ) {
        TMSP430_LEDvec * pLV;

        pLV = malloc(sizeof(TMSP430_LEDvec));
        if (!pLV) return NULL;

        pLV->pL = malloc(sizeof(TMSP430_LED)*
           numberOfElements);
        if (!pLV->pL) {
                free(pLV);
                return NULL;
        }

        pLV->num = (int) numberOfElements;
        memset (pLV->pL, 0, sizeof(TMSP430_LED)*
           numberOfElements);
        return pLV;
}
// updates the vars at the other APs
```

```
int HALimpl_102_sendUpdate (void * pVarData, const void
    * pDrv, uint32_t receiver, uint32_t mNum, int32_t i)
    {
return -1;
}
// decode data for the HAL functions
void * HALimpl_102_decodeData (void * pVarData) {
return pVarData;
}
// delete the variable
void HALimpl_102_delete (void * pVarData) {
        TMSP430_LEDvec * pLV = (TMSP430_LEDvec *)
            pVarData;
        if (pLV) {
                if (pLV->pL) free(pLV->pL);
                free (pLV);
        }

}
```

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:

```
// ================================
// variable implementation for display UI(var id = 103)
// ================================

typedef struct SMSP430_display {
        unsigned char dummy;
} TMSP430_display;

typedef struct SMSP430_displayVec {
        TMSP430_display * pD;
        int num;
} TMSP430_displayVec;

// updates a variable the AP
int HALimpl_103_recvUpdate (void * pVarData, void *
    pMsgData) {
return -1;
```

```
}
// create a new variable
void * HALimpl_103_create (unsigned int numberOfElements
   ) {
        TMSP430_displayVec * pDV;

        pDV = malloc(sizeof(TMSP430_displayVec));
        if (!pDV) return NULL;

        pDV->pD = malloc(sizeof(TMSP430_display)*
           numberOfElements);
        if (!pDV->pD) {
                free(pDV);
                return NULL;
        }

        pDV->num = (int) numberOfElements;
        memset (pDV->pD, 0, sizeof(TMSP430_display)*
           numberOfElements);
        return pDV;
}
// updates the vars at the other APs
int HALimpl_103_sendUpdate (void * pVarData, const void
   * pDrv, uint32_t receiver, uint32_t mNum, int32_t i)
   {
return -1;
}
// decode data for the HAL functions
void * HALimpl_103_decodeData (void * pVarData) {
return pVarData;
}
// delete the variable
void HALimpl_103_delete (void * pVarData) {
        TMSP430_displayVec * pDV = (TMSP430_displayVec
           *) pVarData;
        if (pDV) {
                if (pDV->pD) free(pDV->pD);
                free (pDV);
        }
}
```

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:

```
// ================================
// variable implementation for panel UI(var id = 100)
// ================================

// updates a variable the AP
int HALimpl_100_recvUpdate (void * pVarData, void * pMsg
   ) {
        return -1;
}
// create a new variable
void * HALimpl_100_create (unsigned int numberOfElements
   ) {
        return APgtkUI_createVector(numberOfElements,
           eAPgtkUItype_panel);
}
// updates the vars at the other APs
int HALimpl_100_sendUpdate (void * pVarData, const void
   * pDrv, uint32_t receiver, uint32_t mNum, int32_t i)
   {
        return -1;
}
// decode data for the HAL functions
void * HALimpl_100_decodeData (void * pVarData) {
        return pVarData;
}
// delete the variable
void HALimpl_100_delete (void * pVarData) {
        APgtkUI_destroyVector(pVarData);
}
```

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:

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

```c
// ================================

// updates a variable the AP
int HALimpl_101_recvUpdate (void * pVarData, void * pMsg
    ) {
return -1;
}
// create a new variable
void * HALimpl_101_create (unsigned int numberOfElements
    ) {
        return APgtkUI_createVector(numberOfElements,
            eAPgtkUItype_button);
}

// updates the vars at the other APs
int HALimpl_101_sendUpdate (void * pVarData, const void
    * pDrv, uint32_t receiver, uint32_t mNum, int32_t i)
    {
return -1;
}
// decode data for the HAL functions
void * HALimpl_101_decodeData (void * pVarData) {
        return pVarData;
}
// delete the variable
void HALimpl_101_delete (void * pVarData) {
        APgtkUI_destroyVector(pVarData);
}
```

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:

```c
// ================================
// variable implementation for led UI(var id = 102)
// ================================

// updates a variable the AP
int HALimpl_102_recvUpdate (void * pVarData, void * pMsg
    ) {
return -1;
```

```
}
// create a new variable
void * HALimpl_102_create (unsigned int numberOfElements ) {
        return APgtkUI_createVector(numberOfElements , eAPgtkUItype_LED);
}
// updates the vars at the other APs
int HALimpl_102_sendUpdate (void * pVarData , const void * pDrv , uint32_t receiver , uint32_t mNum , int32_t i) {
return -1;
}
// decode data for the HAL functions
void * HALimpl_102_decodeData (void * pVarData) {
        return pVarData;
}
// delete the variable
void HALimpl_102_delete (void * pVarData) {
        APgtkUI_destroyVector(pVarData);
}
```

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:

```
// ===============================
// variable implementation for display UI(var id = 103)
// ===============================

// updates a variable the AP
int HALimpl_103_recvUpdate (void * pVarData , void * pMsg ) {
return -1;
}
// create a new variable
void * HALimpl_103_create (unsigned int numberOfElements ) {
        return APgtkUI_createVector(numberOfElements , eAPgtkUItype_display);
}
```

```
// updates the vars at the other APs
int HALimpl_103_sendUpdate (void * pVarData, const void
    * pDrv, uint32_t receiver, uint32_t mNum, int32_t i)
    {
return -1;
}
// decode data for the HAL functions
void * HALimpl_103_decodeData (void * pVarData) {
        return pVarData;
}
// delete the variable
void HALimpl_103_delete (void * pVarData) {
        APgtkUI_destroyVector(pVarData);
}
```

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:

```
// ================================
// variable implementation for a string(var id = 3)
// ================================

// updates a variable the AP
int HALimpl_3_recvUpdate (void * pVarData, void * pMsg)
    {
        int32_t *                        pD = (int32_t *)
            pMsg;
        TAPstringVector *        pSV = pVarData;
        TAPstring *                      ps;
        int                                      i, num,
            imax;

        // at the first possion at the message is the
            global var index
        pD++; // skip it (it's the varindex)
        //1. get amount of strings at the string vector
        APendianConversation32Bit((uint32_t *)pD,
            eAP_littleEndian);
        num = (int) *pD;

```

```
        if (APstringVector_resizeVector(pSV,num)) {
                return -1;
        }

        pD++;
        ps = pSV->sv;
        //2. resize and fill the strings
        for (i = 0; i < num; i++) {
                // get length
                APendianConversation32Bit((uint32_t *)pD,eAP_littleEndian);
                if (APstringVector_resize(pSV,i,(size_t)*pD)) {
                        return -2;
                }
                imax = (int) *pD;
                pD++;
                for (i = 0;i <imax; i++)  {
                        ps->szTxt[i] = (char) *pD;
                        pD++;
                }
                ps->szTxt[i] = 0;
                ps++;
        }
        return 0;
}
// create a new variable
void * HALimpl_3_create (unsigned int numberOfElements) {
        return APstringVector_create((int)numberOfElements);
}
// updates the vars at the other APs
int HALimpl_3_sendUpdate (void * pVarData, const void * pDrv, uint32_t receiver, uint32_t mNum, int32_t i) {
        // msgdata = (stringAmount)[(strlen)(string + fill byes), ...]
        // some vars
        unsigned int            indx;
        int                             stl;
        TAPstring *             ps;
        TAPstringVector *       pSV = (TAPstringVector *)pVarData;
        TAPMsgDrv *             pMD = (TAPMsgDrv *) pDrv;
        int32_t                         uint32Amount = 1; // vector size
        int32_t                         sendVal;
        char *                          psz;
        // ok get the size of the message
        ps = pSV->sv;
        for (indx = 0;indx < pSV->num; indx++) {
                // one element for the length
                uint32Amount++;
                // the string data
```

```
                stl = ps->szTxt ? strlen(ps->szTxt) : 0;
                uint32Amount+=stl;
                // next string
                ps++;
        }
        // send header
        pMD->pfkt_updateVariable(pMD->pDrvData, receiver, mNum, i, uint32Amount);
        // send amount of elements
        sendVal = (int32_t) pSV->num;
        pMD->pfkt_sendInteger32(pMD,1,&sendVal);
        ps = pSV->sv;
        for (indx = 0; indx < pSV->num; indx++) {
                if (ps->szTxt) {
                        // get length
                        sendVal = (int32_t) strlen(ps->szTxt);
                        // send length
                        pMD->pfkt_sendInteger32(pMD,1,&sendVal);
                        // send string
                        psz = ps->szTxt;
                        while (*psz) {
                                sendVal = (int32_t) *psz;
                                pMD->pfkt_sendInteger32(pMD,1,&sendVal);
                                psz++;
                        }
                } else {
                        sendVal = 0;
                        // send length
                        pMD->pfkt_sendInteger32(pMD,1,&sendVal);
                }
                ps++;
        }
        return 0;
}
// decode data for the HAL functions
void * HALimpl_3_decodeData (void * pVarData) {
        return pVarData;
}
// delete the variable
void HALimpl_3_delete (void * pVarData) {
        APstringVector_free(pVarData);
}
```

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:

```
// ================================
// variable implementation for a biquad filter(var id = 10)
// ================================

// updates a variable the AP
int HALimpl_10_recvUpdate (void * pVarData, void * pMsgData) {
        uint32_t * pD = (uint32_t *) pMsgData;
        TBQF_BiquadCascade * pV =(TBQF_BiquadCascade *) pVarData;
        TBQF_BiquadDF2 * pE;
        int i,j, imax;
        float * pNum;
        float * pDenom;


        // at the first possion at the message is the global var index
        pD++; // skip it (it's the varindex)
        // and now we are at amount of values
        if (gAPendianFlag != eAP_littleEndian) {
                APendianConversation32Bit(pD, eAP_littleEndian);
        }
        imax = (int) *((int32_t *)pD);
        pD++;
        // 1. resize vector if needed
        if (pV->num != imax) {
                if (BQF_BQFcascadeCreate(pV,imax)) {
                        return -1;
                }
        }

        // 2. fill parameters
        pE = pV->pB;

        for (i = 0; i < imax; i++) {
                if (gAPendianFlag != eAP_littleEndian) {
```

```
                        for (j = 0; j < 6; j++) {
                                APendianConversation32Bit
                                   (pD+j,
                                   eAP_littleEndian);
                        }
                }
                // calc pointers
                pNum = ((float *)pD);
                pD += 3;
                pDenom = ((float *)pD);
                pD += 3;
                //
                BQF_BQFinitFromCoefficents(pV, i, pNum,
                   pDenom);
                pE++;
        }
#ifdef dBQF_implementPrintFunctions
        fprintf(stdout,"rx biquad:\n")
        BQF_PrintBiquad(pV,stdout);
        fflush(stdout);
#endif
        return 0;
}
// create a new variable
void * HALimpl_10_create (unsigned int numberOfElements)
    {
        TBQF_BiquadCascade * pBC = malloc(sizeof(
           TBQF_BiquadCascade));
        if (!pBC) return NULL;
        pBC->num = 0;
        pBC->pB = NULL;
        if (BQF_BQFcascadeCreate(pBC, numberOfElements))
            {
                free(pBC);
                return NULL;
        }
        return pBC;
}

// updates the vars at the other APs
int HALimpl_10_sendUpdate (void * pVarData, const void *
    pDrv, uint32_t receiver, uint32_t mNum, int32_t i) {
        TBQF_BiquadCascade * pV =(TBQF_BiquadCascade *)
           pVarData;
        TBQF_BiquadDF2 * pE;
        TAPMsgDrv * pMD = (TAPMsgDrv *) pDrv;

        int32_t dataAmount;
        int32_t n;
        float num[3];
        float denom[3];

        // calc amount of transmission bytes
        dataAmount = (int32_t) 1 + pV->num * 6;

```

```
        // 1 send header
        pMD->pfkt_updateVariable(pMD->pDrvData, receiver
            , mNum, i, dataAmount);

        // 2 send vector elementwise

        // 2.1 send amount of elements at the vector
        pMD->pfkt_sendInteger32(pMD,1,&pV->num);

        // 2.2 send element
        pE = pV->pB;
        for (n = 0; n < pV->num; n++) {
                BQF_BQFgetNumAndDenom(pV, n, num, denom)
                    ;
                pMD->pfkt_sendFloat32(pMD,3,num);
                pMD->pfkt_sendFloat32(pMD,3,denom);
                // inc
                pE++;
        }
#ifdef dBQF_implementPrintFunctions
        fprintf(stdout,"tx biquad:\n")
        BQF_PrintBiquad(pV,stdout);
        fflush(stdout);
#endif

        return 0;
}
// decode data for the HAL functions
void * HALimpl_10_decodeData (void * pVarData) {
        return pVarData;
}
// delete the variable
void HALimpl_10_delete (void * pVarData) {
        TBQF_BiquadCascade * pBC = (TBQF_BiquadCascade
            *) pVarData;
        if (pBC) {
                BQF_BQFcascadeDelete(pBC);
                free (pBC);
        }
}
```

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:

```
// ================================
// variable implementation for a noisegate(var id = 11)
// ================================

// updates a variable the AP
int HALimpl_11_recvUpdate (void * pVarData, void *
   pMsgData) {
        return -1;
}
// create a new variable
void * HALimpl_11_create (unsigned int numberOfElements)
    {
        if (numberOfElements != 1) return NULL;
        return malloc(sizeof(TDynProc_Noisegate));
}
// updates the vars at the other APs
int HALimpl_11_sendUpdate (void * pVarData, const void *
    pDrv, uint32_t receiver, uint32_t mNum, int32_t i) {
        return -1;
}
// decode data for the HAL functions
void * HALimpl_11_decodeData (void * pVarData) {
        return pVarData;
}
// delete the variable
void HALimpl_11_delete (void * pVarData) {
        TDynProc_Noisegate * pNG = (TDynProc_Noisegate
           *) pVarData;
        free (pNG);
}
```

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:

```c
// ================================
// variable implementation for a expander(var id = 12)
// ================================

// updates a variable the AP
int HALimpl_12_recvUpdate (void * pVarData, void *
    pMsgData) {
return -1;
}
// create a new variable
void * HALimpl_12_create (unsigned int numberOfElements)
     {
        if (numberOfElements != 1) return NULL;
        return malloc(sizeof(TDynProc_Expander));

}
// updates the vars at the other APs
int HALimpl_12_sendUpdate (void * pVarData, const void *
     pDrv, uint32_t receiver, uint32_t mNum, int32_t i) {
        return -1;
}
// decode data for the HAL functions
void * HALimpl_12_decodeData (void * pVarData) {
        return pVarData;
}
// delete the variable
void HALimpl_12_delete (void * pVarData) {
        TDynProc_Expander * pExp = (TDynProc_Expander *)
            pVarData;
        free (pExp);

}
```

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:

```c
// ================================
// variable implementation for a compressor(var id = 13)
// ================================
```

```
// updates a variable the AP
int HALimpl_13_recvUpdate (void * pVarData, void *
    pMsgData) {
        return -1;
}
// create a new variable
void * HALimpl_13_create (unsigned int numberOfElements)
     {
        if (numberOfElements != 1) return NULL;
        return malloc(sizeof(TDynProc_Compressor));
}
// updates the vars at the other APs
int HALimpl_13_sendUpdate (void * pVarData, const void *
     pDrv, uint32_t receiver, uint32_t mNum, int32_t i) {
        return -1;
}
// decode data for the HAL functions
void * HALimpl_13_decodeData (void * pVarData) {
        return pVarData;
}
// delete the variable
void HALimpl_13_delete (void * pVarData) {
        TDynProc_Compressor * pCmp = (
            TDynProc_Compressor *) pVarData;
        free (pCmp);
}
```

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:

```
// ===============================
// variable implementation for a limiter(var id = 14)
// ===============================

// updates a variable the AP
int HALimpl_14_recvUpdate (void * pVarData, void *
    pMsgData) {
        return -1;
}
```

```
// create a new variable
void * HALimpl_14_create (unsigned int numberOfElements)
    {
        if (numberOfElements != 1) return NULL;
        return malloc(sizeof(TDynProc_Limiter));
}
// updates the vars at the other APs
int HALimpl_14_sendUpdate (void * pVarData, const void *
    pDrv, uint32_t receiver, uint32_t mNum, int32_t i) {
        return -1;
}
// decode data for the HAL functions
void * HALimpl_14_decodeData (void * pVarData) {
        return pVarData;
}
// delete the variable
void HALimpl_14_delete (void * pVarData) {
        TDynProc_Limiter * pLim = (TDynProc_Limiter *)
           pVarData;
        free (pLim);
}
```

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:

```
// ================================
// variable implementation for delay (varID=20)
// ================================

// updates a variable the AP
int HALimpl_20_recvUpdate ( void * pVarData, void *
   pMsgData ) {
 // rx function for delay
        uint32_t * pD = (uint32_t *) pMsgData;
        TgenDelay * pV =(TgenDelay *) pVarData;

        int imax, i;

        // at the first possion at the message is the
           global var index
```

```c
		pD++; // skip it (it's the varindex)
		// and now we are at amount of values
		APendianConversation32Bit(pD,eAP_littleEndian);
		imax = (int) *((int32_t *)pD);
		pD++;

		// 1. resize
		if (genDelay_resize(pV, imax)) {
			return -1;
		}

		// 2. fill parameters
		for (i = 0; i < pV->amount; i++) {
			// no endian conversation
			pV->pStart[i]=*((float *)pD);
			pD++;
		}
		return 0;
}
// create a new variable
void * HALimpl_20_create ( unsigned int numberOfElements ) {
		// only one element is allowed
		if (numberOfElements != 1) {
			return NULL;
		}
		// cerate with now data inside
		return genDelay_create(0);
}
// updates the vars at the other APs
int HALimpl_20_sendUpdate ( void * pVarData, const void * pDrv, uint32_t receiver, uint32_t mNum, int32_t i ) {
  // tx function for delay
		TgenDelay * pV =(TgenDelay *) pVarData;
		TAPMsgDrv * pMD = (TAPMsgDrv *) pDrv;

		int32_t dataAmount;

		// calc amount of transmission bytes
		// = size of the delay + amount of the elements at the delay
		dataAmount = (int32_t) 1 + pV->amount;

		// 1 send header
		pMD->pfkt_updateVariable(pMD->pDrvData, receiver, mNum, i, dataAmount);

		// 2 send delay

		// 2.1 send amount of elements at the vector
		pMD->pfkt_sendInteger32(pMD,1,&pV->amount);

		// 2.2 send elements
		pMD->pfkt_sendFloat32(pMD,pV->amount,pV->pStart)
```

```
                ;
        return 0;
}
// decode data for the HAL functions
void * HALimpl_20_decodeData ( void * pVarData ) {
 return pVarData;
}
// delete the variable
void HALimpl_20_delete ( void * pVarData ) {
        TgenDelay * pD = (TgenDelay *) pVarData;
        genDelay_delete(pD);
}
```

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:

```
// ===============================
// function implementation for jump(var id = 50)
// description: jumps n instructions
// ===============================

void HALfunc_ID50_jump(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {
// parameter number descr: amount of instructions to
   jump
//      int32_t* pnumber = &( pParams[0].fp_integer);
        ((TAPInterpreterCPU *)pIPcpu)->pIP += pParams
           [0].fp_integer;
};
```

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:

```
// ================================
// function implementation for jumpCF(var id = 51)
// description: jumps if the carry flag is set n
   instructions
// ================================

void HALfunc_ID51_jumpCF(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {
        if (((TAPInterpreterCPU *)pIPcpu)->CF) {
                ((TAPInterpreterCPU *)pIPcpu)->pIP +=
                   pParams[0].fp_integer;
        } else {
                ((TAPInterpreterCPU *)pIPcpu)->pIP++;
        }
};
```

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:

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

```
// description: jumps if the carry flag is not set n
   instructions
// ================================

void HALfunc_ID52_jumpNCF(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {
        if (!((TAPInterpreterCPU *)pIPcpu)->CF) {
                ((TAPInterpreterCPU *)pIPcpu)->pIP +=
                   pParams[0].fp_integer;
        } else {
                ((TAPInterpreterCPU *)pIPcpu)->pIP++;
        }
};
```

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:

```
// ================================
// function implementation for setCF(var id = 55)
// description: sets the carry flag
// ================================

void HALfunc_ID55_setCF(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {
// parameter value descr: the value of the CF
        if (pParams[0].fp_integer) {
                ((TAPInterpreterCPU *)pIPcpu)->CF = 1;
        } else {
                ((TAPInterpreterCPU *)pIPcpu)->CF = 0;
        }
        ((TAPInterpreterCPU *)pIPcpu)->pIP++;
};
```

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:

```
// ================================
// function implementation for update(var id = 56)
// description: updates a global variable
// ================================

void HALfunc_ID56_update(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {
        if (TX_updateVariable(APInterpreterGetAPfromIP
           (((TAPInterpreterCPU *)pIPcpu)->IP),(uint32_t
           )pParams[0].fp_VarIndex)) {
                ((TAPInterpreterCPU *)pIPcpu)->EF =
                   -100;
        } else {
                ((TAPInterpreterCPU *)pIPcpu)->pIP++;
        }
};
```

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:

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

```
// description: a = values
// ================================

void HALfunc_ID20_assignConstInteger(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {
// parameter iv descr: integer vector
        TAPgenericIntegerVector * piv = (
           TAPgenericIntegerVector *) pParams[0].fp_pD;
// parameter iStart descr: start index
        int32_t iStart = pParams[1].fp_integer;
// parameter num descr: number of values
        int32_t num = pParams[2].fp_integer;

        int32_t * pSrc =(int32_t *) &pParams[3].fp_raw;

        APgenericIntegerVector_assignConst(piv, (int)
           iStart, (int)num, pSrc);

        ((TAPInterpreterCPU *)pIPcpu)->pIP++;
};
```

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:

```
// ================================
// function implementation for assignInteger(var id =
   21)
// description: a = b
// ================================

void HALfunc_ID21_assignInteger(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {
// parameter a descr: vector a
        TAPgenericIntegerVector * pa = (
           TAPgenericIntegerVector *)pParams[0].fp_pD;
// parameter b descr: vector b
        TAPgenericIntegerVector* pb = (
           TAPgenericIntegerVector *)pParams[1].fp_pD;

        APgenericIntegerVector_assign(pa, pb);
```

```

        ((TAPInterpreterCPU *)pIPcpu)->pIP++;
};
```

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:

```
// ================================
// function implementation for addInteger(var id = 22)
// description: c = a + b
// ================================

void HALfunc_ID22_addInteger(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {
// parameter a descr: vector a
        TAPgenericIntegerVector * pa = (
           TAPgenericIntegerVector *)pParams[0].fp_pD;
// parameter b descr: vector b
        TAPgenericIntegerVector* pb = (
           TAPgenericIntegerVector *)pParams[1].fp_pD;
// parameter c descr: vector c
        TAPgenericIntegerVector* pc = (
           TAPgenericIntegerVector *)pParams[2].fp_pD;

        APgenericIntegerVector_add(pa, pb, pc);

        ((TAPInterpreterCPU *)pIPcpu)->pIP++;
};
```

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:

```
// ================================
// function implementation for subInteger(var id = 23)
// description: c = a - b
// ================================

void HALfunc_ID23_subInteger(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {
// parameter a descr: vector a
        TAPgenericIntegerVector * pa = (
           TAPgenericIntegerVector *)pParams[0].fp_pD;
// parameter b descr: vector b
        TAPgenericIntegerVector* pb = (
           TAPgenericIntegerVector *)pParams[1].fp_pD;
// parameter c descr: vector c
        TAPgenericIntegerVector* pc = (
           TAPgenericIntegerVector *)pParams[2].fp_pD;

        APgenericIntegerVector_sub(pa, pb, pc);

        ((TAPInterpreterCPU *)pIPcpu)->pIP++;
};
```

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:

```
// ================================
// function implementation for mulInteger(var id = 24)
// description: c = a * b
// ================================

void HALfunc_ID24_mulInteger(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {
```

```
// parameter a descr: vector a
        TAPgenericIntegerVector * pa = (
           TAPgenericIntegerVector *)pParams[0].fp_pD;
// parameter b descr: vector b
        TAPgenericIntegerVector* pb = (
           TAPgenericIntegerVector *)pParams[1].fp_pD;
// parameter c descr: vector c
        TAPgenericIntegerVector* pc = (
           TAPgenericIntegerVector *)pParams[2].fp_pD;

        APgenericIntegerVector_mul(pa, pb, pc);

        ((TAPInterpreterCPU *)pIPcpu)->pIP++;
};
```

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:

```
// ==============================
// function implementation for divInteger(var id = 25)
// description: c = a / b
// ==============================

void HALfunc_ID25_divInteger(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {
// parameter a descr: vector a
        TAPgenericIntegerVector * pa = (
           TAPgenericIntegerVector *)pParams[0].fp_pD;
// parameter b descr: vector b
        TAPgenericIntegerVector* pb = (
           TAPgenericIntegerVector *)pParams[1].fp_pD;
// parameter c descr: vector c
        TAPgenericIntegerVector* pc = (
           TAPgenericIntegerVector *)pParams[2].fp_pD;

        APgenericIntegerVector_div(pa, pb, pc);

        ((TAPInterpreterCPU *)pIPcpu)->pIP++;
};
```

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:

```
// ================================
// function implementation for modInteger(var id = 26)
// description: c = mod(a,b)
// ================================

void HALfunc_ID26_modInteger(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {
// parameter a descr: vector a
        TAPgenericIntegerVector * pa = (
           TAPgenericIntegerVector *)pParams[0].fp_pD;
// parameter b descr: vector b
        TAPgenericIntegerVector* pb = (
           TAPgenericIntegerVector *)pParams[1].fp_pD;
// parameter c descr: vector c
        TAPgenericIntegerVector* pc = (
           TAPgenericIntegerVector *)pParams[2].fp_pD;

        APgenericIntegerVector_mod(pa, pb, pc);

        ((TAPInterpreterCPU *)pIPcpu)->pIP++;
};
```

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:

```
// ================================
// function implementation for assignConstRational(var
   id = 27)
// description: a = values
// ================================
void HALfunc_ID27_assignConstRational(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {
// parameter iv descr: rational vector
        TAPgenericRationalVector * piv = (
           TAPgenericRationalVector *) pParams[0].fp_pD;
// parameter iStart descr: start index
        int32_t iStart = pParams[1].fp_integer;
// parameter num descr: amount of values used
        int32_t num = pParams[2].fp_integer;

        float * pSrc =(float *) &pParams[3].fp_raw;

        APgenericRationalVector_assignConst (piv, (int)
           iStart, (int)num, pSrc);

        ((TAPInterpreterCPU *)pIPcpu)->pIP++;
};
```

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:

```
// ================================
// function implementation for assignRational(var id =
   28)
// description: a = b
// ================================
void HALfunc_ID28_assignRational(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {
// parameter a descr: vector a
        TAPgenericRationalVector * pa = (
           TAPgenericRationalVector *)pParams[0].fp_pD;
// parameter b descr: vector b
```

```
        TAPgenericRationalVector* pb = (
           TAPgenericRationalVector *)pParams[1].fp_pD;

        APgenericRationalVector_assign(pa, pb);

        ((TAPInterpreterCPU *)pIPcpu)->pIP++;

};
```

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:

```
// ===============================
// function implementation for addRational(var id = 29)
// description: c = a + b
// ===============================
void HALfunc_ID29_addRational(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {
// parameter a descr: vector a
        TAPgenericRationalVector * pa = (
           TAPgenericRationalVector *)pParams[0].fp_pD;
// parameter b descr: vector b
        TAPgenericRationalVector * pb = (
           TAPgenericRationalVector *)pParams[1].fp_pD;
// parameter c descr: vector c
        TAPgenericRationalVector * pc = (
           TAPgenericRationalVector *)pParams[2].fp_pD;

        APgenericRationalVector_add(pa, pb, pc);

        ((TAPInterpreterCPU *)pIPcpu)->pIP++;
};
```

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:

```
// ================================
// function implementation for subRational(var id = 30)
// description: c = a - b
// ================================
void HALfunc_ID30_subRational(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {
// parameter a descr: vector a
        TAPgenericRationalVector * pa = (
           TAPgenericRationalVector *)pParams[0].fp_pD;
// parameter b descr: vector b
        TAPgenericRationalVector * pb = (
           TAPgenericRationalVector *)pParams[1].fp_pD;
// parameter c descr: vector c
        TAPgenericRationalVector * pc = (
           TAPgenericRationalVector *)pParams[2].fp_pD;

        APgenericRationalVector_sub(pa, pb, pc);

        ((TAPInterpreterCPU *)pIPcpu)->pIP++;
};
```

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:

```
// ================================
// function implementation for mulRational(var id = 31)
// description: c = a * b
// ================================
void HALfunc_ID31_mulRational(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {
// parameter a descr: vector a
        TAPgenericRationalVector * pa = (
           TAPgenericRationalVector *)pParams[0].fp_pD;
// parameter b descr: vector b
        TAPgenericRationalVector * pb = (
           TAPgenericRationalVector *)pParams[1].fp_pD;
// parameter c descr: vector c
        TAPgenericRationalVector * pc = (
           TAPgenericRationalVector *)pParams[2].fp_pD;

        APgenericRationalVector_mul(pa, pb, pc);

        ((TAPInterpreterCPU *)pIPcpu)->pIP++;
};
```

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:

```
// ================================
// function implementation for divRational(var id = 32)
// description: c = a / b
// ================================
void HALfunc_ID32_divRational(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {
// parameter a descr: vector a
        TAPgenericRationalVector * pa = (
           TAPgenericRationalVector *)pParams[0].fp_pD;
// parameter b descr: vector b
        TAPgenericRationalVector * pb = (
           TAPgenericRationalVector *)pParams[1].fp_pD;
// parameter c descr: vector c
        TAPgenericRationalVector * pc = (
           TAPgenericRationalVector *)pParams[2].fp_pD;
```

```
        APgenericRationalVector_div(pa, pb, pc);

        ((TAPInterpreterCPU *)pIPcpu)->pIP++;
};
```

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:

```
// ===============================
// function implementation for modRational(var id = 33)
// description: c = mod(a,b)
// ===============================
void HALfunc_ID33_modRational(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {
// parameter a descr: vector a
        TAPgenericRationalVector * pa = (
           TAPgenericRationalVector *)pParams[0].fp_pD;
// parameter b descr: vector b
        TAPgenericRationalVector * pb = (
           TAPgenericRationalVector *)pParams[1].fp_pD;
// parameter c descr: vector c
        TAPgenericRationalVector * pc = (
           TAPgenericRationalVector *)pParams[2].fp_pD;

        APgenericRationalVector_mod(pa, pb, pc);

        ((TAPInterpreterCPU *)pIPcpu)->pIP++;
};
```

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:

```
// ================================
// function implementation for compareRationalLess(var id = 34)
// description: a < b ? CF = 1 : CF = 0
// ================================
void HALfunc_ID34_compareRationalLess(void * pIPcpu, TuAPInterpreterFunctionParameter * pParams) {
// parameter a descr: vector a
        TAPgenericRationalVector * pa = (TAPgenericRationalVector *)pParams[0].fp_pD;
// parameter ia descr: a index
        int32_t ia = pParams[1].fp_integer;
// parameter b descr: vector b
        TAPgenericRationalVector* pb = (TAPgenericRationalVector *)pParams[2].fp_pD;
// parameter ib descr: b index
        int32_t ib = pParams[3].fp_integer;

        if (APgenericRationalVector_cmpEle(pa, pb, ia, ib) < 0) {
                ((TAPInterpreterCPU *)pIPcpu)->CF = 1;
        } else {
                ((TAPInterpreterCPU *)pIPcpu)->CF = 0;
        }
        ((TAPInterpreterCPU *)pIPcpu)->pIP++;


};
```

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:

```
// ================================
// function implementation for compareRationalMore(var
    id = 35)
// description: a > b ? CF = 1 : CF = 0
// ================================
void HALfunc_ID35_compareRationalMore(void * pIPcpu,
    TuAPInterpreterFunctionParameter * pParams) {
// parameter a descr: vector a
        TAPgenericRationalVector * pa = (
           TAPgenericRationalVector *)pParams[0].fp_pD;
// parameter ia descr: a index
        int32_t ia = pParams[1].fp_integer;
// parameter b descr: vector b
        TAPgenericRationalVector* pb = (
           TAPgenericRationalVector *)pParams[2].fp_pD;
// parameter ib descr: b index
        int32_t ib = pParams[3].fp_integer;

        if (APgenericRationalVector_cmpEle(pa, pb, ia,
           ib) > 0) {
                ((TAPInterpreterCPU *)pIPcpu)->CF = 1;
        } else {
                ((TAPInterpreterCPU *)pIPcpu)->CF = 0;
        }
        ((TAPInterpreterCPU *)pIPcpu)->pIP++;
};
```

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:

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

```
// parameter ia descr: a index
        int32_t ia = pParams[1].fp_integer;
// parameter b descr: vector b
        TAPgenericRationalVector* pb = (TAPgenericRationalVector *)pParams[2].fp_pD;
// parameter ib descr: b index
        int32_t ib = pParams[3].fp_integer;

        if (APgenericRationalVector_cmpEle(pa, pb, ia, ib) == 0) {
                ((TAPInterpreterCPU *)pIPcpu)->CF = 1;
        } else {
                ((TAPInterpreterCPU *)pIPcpu)->CF = 0;
        }
        ((TAPInterpreterCPU *)pIPcpu)->pIP++;
};
```

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:

```
// ================================
// function implementation for compareRationalNEqual(var id = 37)
// description: a != b ? CF = 1 : CF = 0
// ================================
void HALfunc_ID37_compareRationalNEqual(void * pIPcpu, TuAPInterpreterFunctionParameter * pParams) {
// parameter a descr: vector a
        TAPgenericRationalVector * pa = (TAPgenericRationalVector *)pParams[0].fp_pD;
// parameter ia descr: a index
        int32_t ia = pParams[1].fp_integer;
// parameter b descr: vector b
        TAPgenericRationalVector* pb = (TAPgenericRationalVector *)pParams[2].fp_pD;
// parameter ib descr: b index
        int32_t ib = pParams[3].fp_integer;

```

```
        if (APgenericRationalVector_cmpEle(pa, pb, ia, ib) != 0) {
                ((TAPInterpreterCPU *)pIPcpu)->CF = 1;
        } else {
                ((TAPInterpreterCPU *)pIPcpu)->CF = 0;
        }
        ((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:

```
// ===============================
// function implementation for compareRationalLessEqual(var id = 38)
// description: a <= b ? CF = 1 : CF = 0
// ===============================
void HALfunc_ID38_compareRationalLessEqual(void * pIPcpu, TuAPInterpreterFunctionParameter * pParams) {
// parameter a descr: vector a
        TAPgenericRationalVector * pa = (TAPgenericRationalVector *)pParams[0].fp_pD;
// parameter ia descr: a index
        int32_t ia = pParams[1].fp_integer;
// parameter b descr: vector b
        TAPgenericRationalVector* pb = (TAPgenericRationalVector *)pParams[2].fp_pD;
// parameter ib descr: b index
        int32_t ib = pParams[3].fp_integer;

        if (APgenericRationalVector_cmpEle(pa, pb, ia, ib) > 0) {
                ((TAPInterpreterCPU *)pIPcpu)->CF = 0;
        } else {
                ((TAPInterpreterCPU *)pIPcpu)->CF = 1;
        }
        ((TAPInterpreterCPU *)pIPcpu)->pIP++;
};
```

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:

```
// ================================
// function implementation for compareRationalMoreEqual(
    var id = 39)
// description: a >= b ? CF = 1 : CF = 0
// ================================
void HALfunc_ID39_compareRationalMoreEqual(void * pIPcpu
    , TuAPInterpreterFunctionParameter * pParams) {
// parameter a descr: vector a
        TAPgenericRationalVector * pa = (
           TAPgenericRationalVector *)pParams[0].fp_pD;
// parameter ia descr: a index
        int32_t ia = pParams[1].fp_integer;
// parameter b descr: vector b
        TAPgenericRationalVector* pb = (
           TAPgenericRationalVector *)pParams[2].fp_pD;
// parameter ib descr: b index
        int32_t ib = pParams[3].fp_integer;

        if (APgenericRationalVector_cmpEle(pa, pb, ia,
           ib)  < 0) {
                ((TAPInterpreterCPU *)pIPcpu)->CF = 0;
        } else {
                ((TAPInterpreterCPU *)pIPcpu)->CF = 1;
        }
        ((TAPInterpreterCPU *)pIPcpu)->pIP++;
};
```

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:

```
// ================================
// function implementation for comparaIntegerLess(var id
   = 40)
// description: a < b ? CF = 1 : CF = 0
// ================================

void HALfunc_ID40_compareIntegerLess(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {
// parameter a descr: a vector
        TAPgenericIntegerVector * pa = (
           TAPgenericIntegerVector *) pParams[0].fp_pD;
// parameter ia descr: a index
        int32_t ia = pParams[1].fp_integer;
// parameter b descr: b vector
        TAPgenericIntegerVector * pb = (
           TAPgenericIntegerVector *) pParams[2].fp_pD;
// parameter ib descr: b index
        int32_t ib = pParams[3].fp_integer;

        if (APgenericIntegerVector_cmpEle(pa, pb, ia, ib
           ) < 0) {
                ((TAPInterpreterCPU *)pIPcpu)->CF = 1;
        } else {
                ((TAPInterpreterCPU *)pIPcpu)->CF = 0;
        }
        ((TAPInterpreterCPU *)pIPcpu)->pIP++;
};
```

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:

```
// ================================
```

```
// function implementation for comparaIntegerMore(var id
     = 41)
// description: a > b ? CF = 1 : CF = 0
// ================================

void HALfunc_ID41_compareIntegerMore(void * pIPcpu,
    TuAPInterpreterFunctionParameter * pParams) {
// parameter a descr: a vector
        TAPgenericIntegerVector * pa = (
           TAPgenericIntegerVector *) pParams[0].fp_pD;
// parameter ia descr: a index
        int32_t ia = pParams[1].fp_integer;
// parameter b descr: b vector
        TAPgenericIntegerVector * pb = (
           TAPgenericIntegerVector *) pParams[2].fp_pD;
// parameter ib descr: b index
        int32_t ib = pParams[3].fp_integer;

        if (APgenericIntegerVector_cmpEle(pa, pb, ia, ib
           ) > 0) {
                ((TAPInterpreterCPU *)pIPcpu)->CF = 1;
        } else {
                ((TAPInterpreterCPU *)pIPcpu)->CF = 0;
        }
        ((TAPInterpreterCPU *)pIPcpu)->pIP++;
};
```

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:

```
// ================================
// function implementation for comparaIntegerEqual(var
    id = 42)
// description: a == b ? CF = 1 : CF = 0
// ================================

void HALfunc_ID42_compareIntegerEqual(void * pIPcpu,
    TuAPInterpreterFunctionParameter * pParams) {
// parameter a descr: a vector
```

```
        TAPgenericIntegerVector * pa = (TAPgenericIntegerVector *) pParams[0].fp_pD;
// parameter ia descr: a index
        int32_t ia = pParams[1].fp_integer;
// parameter b descr: b vector
        TAPgenericIntegerVector * pb = (TAPgenericIntegerVector *) pParams[2].fp_pD;
// parameter ib descr: b index
        int32_t ib = pParams[3].fp_integer;

        if (APgenericIntegerVector_cmpEle(pa, pb, ia, ib) == 0) {
                ((TAPInterpreterCPU *)pIPcpu)->CF = 1;
        } else {
                ((TAPInterpreterCPU *)pIPcpu)->CF = 0;
        }
        ((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:

```
// ==============================
// function implementation for comparaIntegerNEqual(var id = 43)
// description: a != b ? CF = 1 : CF = 0
// ==============================

void HALfunc_ID43_compareIntegerNEqual(void * pIPcpu, TuAPInterpreterFunctionParameter * pParams) {
// parameter a descr: a vector
        TAPgenericIntegerVector * pa = (TAPgenericIntegerVector *) pParams[0].fp_pD;
// parameter ia descr: a index
        int32_t ia = pParams[1].fp_integer;
// parameter b descr: b vector
        TAPgenericIntegerVector * pb = (TAPgenericIntegerVector *) pParams[2].fp_pD;
// parameter ib descr: b index
        int32_t ib = pParams[3].fp_integer;
```

```
        if (APgenericIntegerVector_cmpEle(pa, pb, ia, ib
           ) != 0) {
                ((TAPInterpreterCPU *)pIPcpu)->CF = 1;
        } else {
                ((TAPInterpreterCPU *)pIPcpu)->CF = 0;
        }
        ((TAPInterpreterCPU *)pIPcpu)->pIP++;

};
```

3.3.30 compareIntegerLessEqual (rational and integer generic)

Informations:

function HAL id:	44
variable type name:	compareIntegerLessEqual
group:	rational and integer generic
description:	a<=b

includes:

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

code:

```
// ===============================
// function implementation for comparaIntegerLessEqual(
   var id = 44)
// description: a <= b ? CF = 1 : CF = 0
// ===============================

void HALfunc_ID44_compareIntegerLessEqual(void * pIPcpu,
    TuAPInterpreterFunctionParameter * pParams) {
// parameter a descr: a vector
        TAPgenericIntegerVector * pa = (
           TAPgenericIntegerVector *) pParams[0].fp_pD;
// parameter ia descr: a index
        int32_t ia = pParams[1].fp_integer;
// parameter b descr: b vector
        TAPgenericIntegerVector * pb = (
           TAPgenericIntegerVector *) pParams[2].fp_pD;
// parameter ib descr: b index
        int32_t ib = pParams[3].fp_integer;

        if (APgenericIntegerVector_cmpEle(pa, pb, ia, ib
           ) > 0) {
                ((TAPInterpreterCPU *)pIPcpu)->CF = 0;
        } else {
                ((TAPInterpreterCPU *)pIPcpu)->CF = 1;
```

```
        }
        ((TAPInterpreterCPU *)pIPcpu)->pIP++;
};
```

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:

```
// ================================
// function implementation for comparaIntegerMoreEqual(
   var id = 45)
// description: a >= b ? CF = 1 : CF = 0
// ================================

void HALfunc_ID45_compareIntegerMoreEqual(void * pIPcpu,
    TuAPInterpreterFunctionParameter * pParams) {
// parameter a descr: a vector
        TAPgenericIntegerVector * pa = (
           TAPgenericIntegerVector *) pParams[0].fp_pD;
// parameter ia descr: a index
        int32_t ia = pParams[1].fp_integer;
// parameter b descr: b vector
        TAPgenericIntegerVector * pb = (
           TAPgenericIntegerVector *) pParams[2].fp_pD;
// parameter ib descr: b index
        int32_t ib = pParams[3].fp_integer;

        if (APgenericIntegerVector_cmpEle(pa, pb, ia, ib
           ) < 0) {
                ((TAPInterpreterCPU *)pIPcpu)->CF = 0;
        } else {
                ((TAPInterpreterCPU *)pIPcpu)->CF = 1;
        }
        ((TAPInterpreterCPU *)pIPcpu)->pIP++;
};
```

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:

```
// ================================
// function implementation for readSample(var id = 60)
// description: reading a sample from an input
// ================================

void HALfunc_ID60_readSample(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {
// parameter channel descr: the channel
        // int32_t* pchannel =  pParams[0].fp_integer;
// parameter resValue descr: the result of the action
        // int32_t* pSample =  pParams[1].fp_pD;

        // foreward declaration of the global variable
        extern TStjWAVmodule gWAVModule;

        if (WAVmoduleGetInput(&gWAVModule,pParams[0].
           fp_integer,((TAPvarRational *)pParams[1].
           fp_pD)->pR)){
                ((TAPInterpreterCPU *)pIPcpu)->EF = 1;
        }

        ((TAPInterpreterCPU *)pIPcpu)->pIP++;
};
```

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:

```
// ================================
// function implementation for writeSample(var id = 61)
// description: writes a sample to a output
// ================================

void HALfunc_ID61_writeSample(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {
// parameter channel descr: the channel
//       int32_t* pchannel = &( pParams[0].fp_integer);
// parameter value descr: the value to be written to the
    output
//       void ** pvalue = &( pParams[1].fp_pD);

        // foreward declaration of the global variable
        extern TStjWAVmodule gWAVModule;
        if (WAVmoduleSetOutput(&gWAVModule,pParams[0].
           fp_integer,*((TAPvarRational *)pParams[1].
           fp_pD)->pR)){
                ((TAPInterpreterCPU *)pIPcpu)->EF = 1;
        }

        ((TAPInterpreterCPU *)pIPcpu)->pIP++;
};
```

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:

```
// ================================
// function implementation for initRFFT(var id = 130)
// description: init FFT structure as real input FFT
// ================================

void HALfunc_ID130_initRFFT(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {
        // parameter N descr: FFT length
        int32_t N = pParams[0].fp_integer;

```

```
        // parameter x descr: input
        float * px = (float *) pParams[1].fp_pD;

        // parameter y descr: output
        fftwf_complex * py = (fftwf_complex *) pParams
           [2].fp_pD;

        // parameter fftStruct descr: FFT structure
        TAPInterpreterVariable * pFFTVar = pParams[3].
           fp_pV;

        if (pFFTVar->pData) fftwf_destroy_plan(pFFTVar->
           pData);
        pFFTVar->pData = fftwf_plan_dft_r2c_1d(N, px, py
           , FFTW_ESTIMATE);

        // increment IP
        ((TAPInterpreterCPU *)pIPcpu)->pIP++;
};
```

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:

```
// ================================
// function implementation for initIFFT(var id = 131)
// description: init inverse FFT structure
// ================================

void HALfunc_ID131_initIFFT(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {
        // parameter N descr: FFT length
        int32_t N = pParams[0].fp_integer;

        // parameter x descr: input
        fftwf_complex * px = (fftwf_complex *) pParams
           [1].fp_pD;

        // parameter y descr: output
        float * py = (float *) pParams[2].fp_pD;

```

```
        // parameter fftStruct descr: FFT structure
        TAPInterpreterVariable * pFFTVar = pParams[3].
           fp_pV;

        if (pFFTVar->pData) fftwf_destroy_plan(pFFTVar->
           pData);
        pFFTVar->pData = fftwf_plan_dft_c2r_1d(N, px, py
           , FFTW_ESTIMATE);

        // increment IP
        ((TAPInterpreterCPU *)pIPcpu)->pIP++;
};
```

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:

```
// ================================
// function implementation for processRFFT(var id = 132)
// description: processes the real input FFT
// ================================

void HALfunc_ID132_processRFFT(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {
        // parameter fftStruct descr: the fft structure
        fftwf_execute(pParams[0].fp_pV->pData);

        // increment IP
        ((TAPInterpreterCPU *)pIPcpu)->pIP++;
};
```

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:

```
// ================================
// function implementation for processIFFT(var id = 133)
// description: processes the IFFT
// ================================

void HALfunc_ID133_processIFFT(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {
        // parameter fftStruct descr: the fft structure
        fftwf_execute(pParams[0].fp_pV->pData);

        // increment IP
        ((TAPInterpreterCPU *)pIPcpu)->pIP++;
}
```

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:

```
// ================================
// function implementation for readSampleFrame(var id =
   62)
// description: reading a frames of sample from an input
// ================================

void HALfunc_ID62_readSampleFrame(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {
        // parameter channel descr: the channel
        int32_t channel = pParams[0].fp_integer;
        // parameter frameBuffer descr: the buffer witch
            receives the samples
        float * pFrame = (float *) pParams[1].fp_pD;
```

```
        // parameter waitForNewFrame descr: if not zero the function waits for a new sample frame
        int waitForNewFrame = (int)pParams[2].fp_integer;


        extern TStjFrameWAVmodule gFrameWAVModule;

        if (FrameWAVmoduleGetInput(&gFrameWAVModule, channel,pFrame)){
                ((TAPInterpreterCPU *)pIPcpu)->EF = 1;
        }
        ((TAPInterpreterCPU *)pIPcpu)->pIP++;
};
```

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:

```
// ==============================
// function implementation for writeSampleFrame(var id = 63)
// description: writes a frame of samples to a output
// ==============================

void HALfunc_ID63_writeSampleFrame(void * pIPcpu, TuAPInterpreterFunctionParameter * pParams) {
        // parameter channel descr: the channel
        int32_t channel = pParams[0].fp_integer;
        // parameter frameBuffer descr: the buffer which is writen to the channel
error rational vartype
        float * pFrame = (float *) pParams[1].fp_pD;

        extern TStjFrameWAVmodule gFrameWAVModule;

        if (FrameWAVmoduleSetOutput(&gFrameWAVModule, channel,pFrame)){
                ((TAPInterpreterCPU *)pIPcpu)->EF = 1;
```

```
        }
        ((TAPInterpreterCPU *)pIPcpu)->pIP++;
}
```

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:

```
// ================================
// function implementation for readSampleFrame(var id =
   62)
// description: reading a frames of sample from an input
// ================================

void HALfunc_ID62_readSampleFrame(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {
// parameter channel descr: the channel
        // pParams[0].fp_integer
// parameter resValue descr: the result of the action
        TAPgenericRationalVector * pRV = (
           TAPgenericRationalVector *) pParams[1].fp_pD;
// parameter waitForNewFrame descr: if not zero the
   function waits for a new sample frame
        int waitForNewFrame = (int)pParams[2].fp_integer
           ;

        TCodecChannel * pC;

        // the channels at the AP starts with 1; at the
           ADSP with 0
        pC = ADSP_getChannel((int)pParams[0].fp_integer
           -1);
        if (!pC) {
                ((TAPInterpreterCPU *)pIPcpu)->EF = -1;
                return;
        }

```

```
        ADSP_readSamplesFromChannel(pC,pRV->pVal,pRV->
           num, waitForNewFrame);

        ((TAPInterpreterCPU *)pIPcpu)->pIP++;
};
```

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:

```
// ==============================
// function implementation for writeSampleFrame(var id =
   63)
// description: writes a frame of samples to a output
// ==============================

void HALfunc_ID63_writeSampleFrame(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {
// parameter channel descr: the channel
        // pParams[0].fp_integer
// parameter resValue descr: the result of the action
        TAPgenericRationalVector * pRV = (
           TAPgenericRationalVector *) pParams[1].fp_pD;

        TCodecChannel * pC;

        // the channels at the AP starts with 1; at the
           ADSP with 0
        pC = ADSP_getChannel((int)pParams[0].fp_integer
           -1);
        if (!pC) {
                ((TAPInterpreterCPU *)pIPcpu)->EF = -1;
                return;
        }

        ADSP_writesSamplesToChannel(pC,pRV->pVal,pRV->
           num);

```

```
        ((TAPInterpreterCPU *)pIPcpu)->pIP++;
};
```

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:

```
// ================================
// function implementation for readSampleFrame(var id =
   62)
// description: reading a frames of sample from an input
// ================================

void HALfunc_ID62_readSampleFrame(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {
// parameter channel descr: the channel
        int32_t channel = pParams[0].fp_integer;
// parameter frameBuffer descr: the buffer witch
   receives the samples
        TAPvarRational * pRA = (TAPvarRational *)
           pParams[1].fp_pD;
// parameter waitForNewFrame descr: if not zero the
   function waits for a new sample frame
        int waitForNewFrame = (int)pParams[2].fp_integer
           ;

        extern TStjFrameWAVmodule gFrameWAVModule;

        if (FrameWAVmoduleGetInput(&gFrameWAVModule,(int
           )channel,pRA->num,pRA->pR)) {
                ((TAPInterpreterCPU *)pIPcpu)->EF = -1;
        }
        ((TAPInterpreterCPU *)pIPcpu)->pIP++;
};
```

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:

```
// ================================
// function implementation for writeSampleFrame(var id =
    63)
// description: writes a frame of samples to a output
// ================================

void HALfunc_ID63_writeSampleFrame(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {
// parameter channel descr: the channel
        int32_t channel = pParams[0].fp_integer;
// parameter frameBuffer descr: the buffer which is
   writen to the channel
        TAPvarRational * pRA = (TAPvarRational *)
           pParams[1].fp_pD;

        extern TStjFrameWAVmodule gFrameWAVModule;

        if (FrameWAVmoduleSetOutput(&gFrameWAVModule,(
           int)channel,pRA->num,pRA->pR)) {
                ((TAPInterpreterCPU *)pIPcpu)->EF = -1;
        }
        ((TAPInterpreterCPU *)pIPcpu)->pIP++;
};
```

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 (emty because of real hardware)

includes:

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

code:

```
// ================================
// function implementation for uiSetDim(var id = 200)
// description: sets the dimension of a UI
// ================================

void HALfunc_ID200_uiSetDim(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {
         ((TAPInterpreterCPU *)pIPcpu)->pIP++;
};
```

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:

```
// ================================
// function implementation for uiInitPanel(var id = 201)
// description: inits a panel
// ================================

void HALfunc_ID201_uiInitPanel(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {
         // do nothing
         ((TAPInterpreterCPU *)pIPcpu)->pIP++;
};
```

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:

```
// ================================
// function implementation for uiInitButton(var id =
   202)
// description: inits a button
// ================================

void HALfunc_ID202_uiInitButton(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {
// parameter uiUUID descr: uuid of the button
        int32_t UUID = pParams[0].fp_integer;
// parameter b descr: button
        TMSP430_buttonVec * pbv = (TMSP430_buttonVec *)
           pParams[1].fp_pD;
// parameter bIndex descr: button index
        int32_t bIndex = pParams[2].fp_integer;
// parameter p descr: panel
        TMSP430_panelVec * pp = (TMSP430_panelVec *)
           pParams[3].fp_pD;
// parameter pIndex descr: panel index
        int32_t pIndex = pParams[4].fp_integer;


        if ((bIndex < 0) || (bIndex >= pbv->num)) {
                ((TAPInterpreterCPU *)pIPcpu)->EF =
                   -202;
                return;
        }
        switch (UUID) {
                case 21:
                        pbv->pB[bIndex].port1Mask = BIT5
                           ;
                        break;
                case 22:
                        pbv->pB[bIndex].port1Mask = BIT6
                           ;
                        break;
                case 23:
                        pbv->pB[bIndex].port1Mask = BIT7
                           ;
                        break;
                default:
                        ((TAPInterpreterCPU *)pIPcpu)->
                           EF = -202;
                        return;
        }
        ((TAPInterpreterCPU *)pIPcpu)->pIP++;
};
```

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:

```
// ================================
// function implementation for uiInitDisplay(var id =
    203)
// description: inits a display
// ================================

void HALfunc_ID203_uiInitDisplay(void * pIPcpu,
    TuAPInterpreterFunctionParameter * pParams) {
// parameter uiUUID descr: uuid of the display
        int32_t UUID = pParams[0].fp_integer;
// parameter b descr: button
        TMSP430_displayVec * pdv = (TMSP430_displayVec
            *) pParams[1].fp_pD;
// parameter dIndex descr: display index
        int32_t dIndex = pParams[2].fp_integer;
// parameter p descr: panel
        TMSP430_panelVec * ppv = pParams[3].fp_pD;
// parameter pIndex descr: panel index
        int32_t pIndex = pParams[4].fp_integer;

        if ((dIndex < 0) || (dIndex >= pdv->num)) {
                ((TAPInterpreterCPU *)pIPcpu)->EF =
                    -203;
                return;
        }
        if (UUID != 11) {
                ((TAPInterpreterCPU *)pIPcpu)->EF =
                    -203;
                return;
        }
        // do nothing
        ((TAPInterpreterCPU *)pIPcpu)->pIP++;
};
```

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:

```
// ================================
// function implementation for uiInitLED(var id = 204)
// description: inits a LED
// ================================

void HALfunc_ID204_uiInitLED(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {
// parameter uiUUID descr: uuid of the LED
        int32_t UUID = pParams[0].fp_integer;
// parameter l descr: led
        TMSP430_LEDvec * plv = (TMSP430_LEDvec *)
           pParams[1].fp_pD;
// parameter lIndex descr: LED index
        int32_t lIndex = pParams[2].fp_integer;
// parameter p descr: panel
        TMSP430_panelVec * ppv = (TMSP430_panelVec *)
           pParams[3].fp_pD;
// parameter pIndex descr: panel index
        int32_t pIndex = pParams[4].fp_integer;

        if ((lIndex < 0) || (lIndex >= plv->num)) {
                ((TAPInterpreterCPU *)pIPcpu)->EF =
                   -204;
                return;
        }
        switch (UUID) {
                case 31:
                        plv->pL[lIndex].port3mask = BIT6
                           ;
                        break;
                case 32:
                        plv->pL[lIndex].port3mask = BIT7
                           ;
                        break;
                default:
                        ((TAPInterpreterCPU *)pIPcpu)->
                           EF = -204;
                        return;
        }
```

```
        ((TAPInterpreterCPU *)pIPcpu)->pIP++;
};
```

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:

```
// ===============================
// function implementation for uiCheckButtonPressed(var
   id = 210)
// description: if the button was pressed the CF is set
// ===============================

void HALfunc_ID210_uiCheckButtonPressed(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {
// parameter b descr: button
        TMSP430_buttonVec * pbv = (TMSP430_buttonVec *)
           pParams[0].fp_pD;
// parameter bIndex descr: button index
        int32_t bIndex = pParams[1].fp_integer;

        if ((bIndex < 0) || (bIndex >= pbv->num)){
                ((TAPInterpreterCPU *)pIPcpu)->EF =
                   -210;
                return;
        }
        unsigned char r = P1IN & pbv->pB[bIndex].
           port1Mask;
        ((TAPInterpreterCPU *)pIPcpu)->CF = r ? 1 : 0;
        ((TAPInterpreterCPU *)pIPcpu)->pIP++;
};
```

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:

```
// ================================
// function implementation for uiSetLED(var id = 211)
// description: set the LED state
// ================================

void HALfunc_ID211_uiSetLED(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {
// parameter l descr: LED
        TMSP430_LEDvec * plv = pParams[0].fp_pD;
// parameter lIndex descr: LED index
        int32_t lIndex = pParams[1].fp_integer;
// parameter onFlag descr: if the flag is not zero the
   LED is turned on
        int32_t onFlag = pParams[2].fp_integer;

        if ((lIndex < 0) || (lIndex >= plv->num)) {
                ((TAPInterpreterCPU *)pIPcpu)->EF =
                   -211;
                return;
        }

        if (onFlag) {
                P3OUT &= ~(plv->pL[lIndex].port3mask);
        } else {
                P3OUT |= plv->pL[lIndex].port3mask;
        }
        ((TAPInterpreterCPU *)pIPcpu)->pIP++;
};
```

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:

```
// ================================
// function implementation for uiSetDisplay(var id =
   212)
// description: set the text of a display
// ================================

void HALfunc_ID212_uiSetDisplay(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {
// parameter d descr: display
        TMSP430_display * pd = pParams[0].fp_pD;
// parameter dIndex descr: display index
        int32_t dIndex = pParams[1].fp_integer;
// parameter s descr: the string
        TAPstringVector * pSV = pParams[2].fp_pD;
// parameter iString descr: index of the string at the
   array
        int iString = (int) pParams[3].fp_integer;

        msp430_LCD_print(0,0,pSV->sv[iString].szTxt);
        ((TAPInterpreterCPU *)pIPcpu)->pIP++;
};
```

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:

```
// ================================
// function implementation for uiSetDim(var id = 200)
// description: sets the dimension of a UI
// ================================

void HALfunc_ID200_uiSetDim(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {
```

```
// parameter xPos descr: x position in pixel
        int32_t xPos = pParams[0].fp_integer;
// parameter yPos descr: y position in pixel
        int32_t yPos = pParams[1].fp_integer;
// parameter xLen descr: x width
        int32_t xLen = pParams[2].fp_integer;
// parameter yLen descr: y height
        int32_t yLen = pParams[3].fp_integer;
// parameter ui descr: UI element
        TAPgtkUIvector * puiVec = (TAPgtkUIvector *)
           pParams[4].fp_pV;
// parameter uiIndex descr: UI element index
        int32_t uiIndex = pParams[5].fp_integer;

        APgtkUI_setCoordinates (
                        &(puiVec->pUI[uiIndex]),
                        xPos,
                        yPos,
                        xLen,
                        yLen
                );
        ((TAPInterpreterCPU *)pIPcpu)->pIP++;
};
```

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:

```
// ==============================
// function implementation for uiInitPanel(var id = 201)
// description: inits a panel
// ==============================

void HALfunc_ID201_uiInitPanel(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {
// parameter uiUUID descr: uuid of the pannel
        int32_t uiUUID = pParams[0].fp_integer;
// parameter p descr: pannel
        TAPgtkUIvector * puiVec = (TAPgtkUIvector *)
           pParams[1].fp_pD;
// parameter pIndex descr: pannel index
```

```
        int32_t pIndex = pParams[2].fp_integer;

        if (APgtkUI_createUI (
                        &(puiVec->pUI[pIndex]),
                        NULL,
                        uiUUID,
                        eAPgtkUItype_panel
                )
        ) {
                ((TAPInterpreterCPU *)pIPcpu)->EF = -10;
        }

        ((TAPInterpreterCPU *)pIPcpu)->pIP++;
};
```

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:

```
// ================================
// function implementation for uiInitButton(var id =
   202)
// description: inits a button
// ================================

void HALfunc_ID202_uiInitButton(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {
// parameter uiUUID descr: uuid of the pannel
        int32_t uiUUID = pParams[0].fp_integer;
// parameter b descr: button
        TAPgtkUIvector * puiB = (TAPgtkUIvector *)
           pParams[1].fp_pD;
// parameter bIndex descr: button index
        int32_t bIndex = pParams[2].fp_integer;
// parameter p descr: panel
        TAPgtkUIvector * puiP = (TAPgtkUIvector *)
           pParams[3].fp_pD;
// parameter pIndex descr: panel index
        int32_t pIndex = pParams[4].fp_integer;

        if (APgtkUI_createUI (
```

```
                                &(puiB->pUI[bIndex]),
                                &(puiP->pUI[pIndex]),
                                uiUUID,
                                eAPgtkUItype_button
                        )
        ) {
                ((TAPInterpreterCPU *)pIPcpu)->EF = -10;
        }

        ((TAPInterpreterCPU *)pIPcpu)->pIP++;
};
```

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:

```
// ================================
// function implementation for uiInitDisplay(var id =
   203)
// description: inits a display
// ================================

void HALfunc_ID203_uiInitDisplay(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {
// parameter uiUUID descr: uuid of the panel
        int32_t uiUUID = pParams[0].fp_integer;
// parameter b descr: display
        TAPgtkUIvector * puiD = (TAPgtkUIvector *)
           pParams[1].fp_pD;
// parameter bIndex descr: display index
        int32_t dIndex = pParams[2].fp_integer;
// parameter p descr: panel
        TAPgtkUIvector * puiP = (TAPgtkUIvector *)
           pParams[3].fp_pD;
// parameter pIndex descr: panel index
        int32_t pIndex = pParams[4].fp_integer;

        if (APgtkUI_createUI (
                                &(puiD->pUI[dIndex]),
                                &(puiP->pUI[pIndex]),
                                uiUUID,
```

```
                            eAPgtkUItype_display
                    )
        ) {
                ((TAPInterpreterCPU *)pIPcpu)->EF = -10;
        }

        ((TAPInterpreterCPU *)pIPcpu)->pIP++;
};
```

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:

```
// ================================
// function implementation for uiInitLED(var id = 204)
// description: inits a LED
// ================================

void HALfunc_ID204_uiInitLED(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {
// parameter uiUUID descr: uuid of the LED
        int32_t uiUUID = pParams[0].fp_integer;
// parameter l descr: LED
        TAPgtkUIvector * puiL = (TAPgtkUIvector *)
           pParams[1].fp_pD;
// parameter lIndex descr: LED index
        int32_t lIndex = pParams[2].fp_integer;
// parameter p descr: panel
        TAPgtkUIvector * puiP = (TAPgtkUIvector *)
           pParams[3].fp_pD;
// parameter pIndex descr: panel index
        int32_t pIndex = pParams[4].fp_integer;

        if (APgtkUI_createUI (
                            &(puiL->pUI[lIndex]),
                            &(puiP->pUI[pIndex]),
                            uiUUID,
                            eAPgtkUItype_LED
                    )
        ) {
                ((TAPInterpreterCPU *)pIPcpu)->EF = -10;
```

```
        }

        ((TAPInterpreterCPU *)pIPcpu)->pIP++;
};
```

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:

```
// ================================
// function implementation for uiCheckButtonPressed(var
   id = 210)
// description: if the button was pressed the CF is set
// ================================

void HALfunc_ID210_uiCheckButtonPressed(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {
// parameter b descr: button
        TAPgtkUIvector * puiB = (TAPgtkUIvector *)
           pParams[0].fp_pD;
// parameter bIndex descr: button index
        int32_t bIndex = pParams[1].fp_integer;
// parameter p descr: panel

        if (puiB->pUI[bIndex].ui.button.pressCounter) {
                puiB->pUI[bIndex].ui.button.pressCounter
                   --;
                ((TAPInterpreterCPU *)pIPcpu)->CF = 1;
        } else {
                ((TAPInterpreterCPU *)pIPcpu)->CF = 0;
        }
        ((TAPInterpreterCPU *)pIPcpu)->pIP++;
};
```

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:

```c
// ================================
// function implementation for uiSetLED(var id = 211)
// description: set the LED state
// ================================

void HALfunc_ID211_uiSetLED(void * pIPcpu, TuAPInterpreterFunctionParameter * pParams) {
// parameter l descr: LED
	TAPgtkUIvector * puiL = (TAPgtkUIvector *) pParams[0].fp_pD;
// parameter lIndex descr: LED index
	int32_t lIndex = pParams[1].fp_integer;
// parameter onFlag descr: if the flag is not zero the LED is turned on
	int32_t  onFlag = pParams[2].fp_integer;

	puiL->pUI[lIndex].ui.led.onFlag = (!onFlag) ? 0 : 1;
	APgtkUI_redrawUI(&(puiL->pUI[lIndex]));
	((TAPInterpreterCPU *)pIPcpu)->pIP++;
};
```

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:

```
// ================================
// function implementation for uiSetDisplay(var id =
   212)
// description: set the text of a display
// ================================

void HALfunc_ID212_uiSetDisplay(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {
// parameter d descr: display
        TAPgtkUIvector * puiD = (TAPgtkUIvector *)
           pParams[0].fp_pD;
// parameter dIndex descr: display index
        int32_t dIndex = pParams[1].fp_integer;
// parameter s descr: the string
        TAPstringVector * pSV = (TAPstringVector *)
           pParams[2].fp_pD;
// parameter iString descr: index of the string at the
   array
        int32_t iString = pParams[3].fp_integer;

        gtkAP_DisplaySetText (
                        &(puiD->pUI[dIndex].ui.display),
                        pSV->sv[iString].szTxt
                );
        ((TAPInterpreterCPU *)pIPcpu)->pIP++;
};
```

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:

```
// ================================
// function implementation for setStringSize(var id = 5)
// description: resets the size of a string
// ================================

void HALfunc_ID5_setStringSize(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {
// parameter s descr: the string
```

```
        TAPstringVector * pSV = (TAPstringVector *)
           pParams[0].fp_pV;
// parameter i descr: index of the string at the array
        int32_t i = pParams[1].fp_integer;
// parameter length descr: length of the string
        int32_t length = pParams[2].fp_integer;

        if (APstringVector_resize(pSV,(int) i, length))
           {
                ((TAPInterpreterCPU *)pIPcpu)->EF = -1;
        } else {
                ((TAPInterpreterCPU *)pIPcpu)->pIP++;
        }
};
```

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:

```
// ================================
// function implementation for setStringValues(var id =
   6)
// description: set the string
// ================================

void HALfunc_ID6_setStringValues(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {
// parameter s descr: the string
        TAPstringVector * pSV = (TAPstringVector *)
           pParams[0].fp_pV;
// parameter i descr: index of the string at the array
        int i = (int) pParams[1].fp_integer;
// parameter p descr: position at the string where to
   start from
        int p = (int) pParams[2].fp_integer;

        char * pSrc =(char *) &pParams[3].fp_raw;

        int pe = p + (dAPInterpreterFuncMaxParams-3) *
           sizeof(int32_t);

```

```
        APstringVector_fill(pSV,i,p,pe,pSrc);

        ((TAPInterpreterCPU *)pIPcpu)->pIP++;
};
```

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:

```
// ================================
// function implementation for concatStrings(var id = 7)
// description: concat two strings
// ================================
void HALfunc_ID7_concatStrings(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {
// parameter s1 descr: the string at its end the other
   string is concated
        TAPstringVector * s1 = (TAPstringVector *)
           pParams[0].fp_pD;
// parameter i1 descr: index of the string 1
        int i1 = (int) pParams[1].fp_integer;
// parameter s2 descr: the concat string
        TAPstringVector * s2 = (TAPstringVector *)
           pParams[2].fp_pD;
// parameter i2 descr: index of the string 2
        int i2 = (int) pParams[3].fp_integer;

        if (APstringVector_concat(s1,i1,s2,i2)) {
                ((TAPInterpreterCPU *)pIPcpu)->EF = -1;
        } else {
                ((TAPInterpreterCPU *)pIPcpu)->pIP++;
        }
};
```

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 keept untouched)

includes:

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

code:

```
// ================================
// function implementation for rationalToString(var id = 8)
// description: converts a rational to a string
// ================================
void HALfunc_ID8_rationalToString(void * pIPcpu, TuAPInterpreterFunctionParameter * pParams) {
// parameter s descr: the string
        TAPstringVector * s = (TAPstringVector *) pParams[0].fp_pD;
// parameter sIndex descr: index of the string at the array
        int sIndex = (int) pParams[1].fp_integer;
// parameter r descr: rational vector
        TAPgenericRationalVector * r = ( TAPgenericRationalVector *) pParams[2].fp_pD;
// parameter rIndx descr: rational vector index
        int rIndx = (int) pParams[3].fp_integer;

        APstringVector_printFloat(s, sIndex, r->pVal[rIndx]);

        ((TAPInterpreterCPU *)pIPcpu)->pIP++;
};
```

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 keept untouched)

includes:

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

code:

```
// ================================
// function implementation for integerToString(var id =
   9)
// description: converts an integer to a string
// ================================
void HALfunc_ID9_integerToString(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {
// parameter s descr: the string
        TAPstringVector * s = (TAPstringVector *)
           pParams[0].fp_pD;
// parameter sIndex descr: index of the string at the
   array
        int sIndex = (int) pParams[1].fp_integer;
// parameter i descr: integer vector
        TAPgenericIntegerVector * i = (
           TAPgenericIntegerVector *) pParams[2].fp_pD;
// parameter iIndex descr: integer vector indx
        int iIndex = (int) pParams[3].fp_integer;

        APstringVector_printInt(s, sIndex, i->pVal[
           iIndex]);

        ((TAPInterpreterCPU *)pIPcpu)->pIP++;
};
```

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:

```
// ================================
// function implementation for assignString(var id = 10)
// description: assigns a string to an other
// ================================
```

```
void HALfunc_ID10_assignString(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {
// parameter s1 descr: the string at its end the other
   string is concated
        TAPstringVector * s1 = (TAPstringVector *)
           pParams[0].fp_pD;
// parameter i1 descr: index of the string 1
        int i1 = (int) pParams[1].fp_integer;
// parameter s2 descr: the concat string
        TAPstringVector * s2 = (TAPstringVector *)
           pParams[2].fp_pD;
// parameter i2 descr: index of the string 2
        int i2 = (int) pParams[3].fp_integer;

        if (APstringVector_assign(s1,i1,s2,i2)) {
                ((TAPInterpreterCPU *)pIPcpu)->EF = -1;
        } else {
                ((TAPInterpreterCPU *)pIPcpu)->pIP++;
        }

};
```

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:

```
// ================================
// function implementation for initBiquadAsHP(var id =
   100)
// description: inits a biquad filter as an high pass
   filter
// ================================

void HALfunc_ID100_initBiquadAsHP(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {
// parameter bq descr: biquad cascade
        TBQF_BiquadCascade * pbqc = ((
           TAPInterpreterVariable *) pParams[0].fp_pV)->
           pData;
// parameter index descr: index at the cascade
        int32_t index = pParams[1].fp_integer;
```

```
// parameter fs descr: sample frequency
        float fs = *((TAPgenericRationalVector *) pParams[2].fp_pD)->pVal;
// parameter fc descr: cut off frequency
        float fc = *((TAPgenericRationalVector *) pParams[3].fp_pD)->pVal;

        if (BQF_BQFcascadeInitHP(pbqc, (unsigned int) index, fs, fc)) {
                ((TAPInterpreterCPU *)pIPcpu)->EF = -100;
        } else {
                ((TAPInterpreterCPU *)pIPcpu)->pIP++;
        }
};
```

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:

```
// ================================
// function implementation for initBiquadAsLP(var id = 101)
// description: inits a biquad filter as a low pass filter
// ================================
void HALfunc_ID101_initBiquadAsLP(void * pIPcpu, TuAPInterpreterFunctionParameter * pParams) {
// parameter bq descr: biquad cascade
        TBQF_BiquadCascade * pbqc = ((TAPInterpreterVariable *) pParams[0].fp_pV)->pData;
// parameter index descr: index at the cascade
        int32_t index = pParams[1].fp_integer;
// parameter fs descr: sample frequency
        float fs = *((TAPgenericRationalVector *) pParams[2].fp_pD)->pVal;
// parameter fc descr: cut off frequency
        float fc = *((TAPgenericRationalVector *) pParams[3].fp_pD)->pVal;

```

```
        if (BQF_BQFcascadeInitLP(pbqc, (unsigned int)
           index, fs, fc)) {
                ((TAPInterpreterCPU *)pIPcpu)->EF =
                   -101;
        } else {
                ((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:

```
// ================================
// function implementation for initBiquadAsPeakFilter(
   var id = 102)
// description: inits a biquad filter as peak filter
// ================================
void HALfunc_ID102_initBiquadAsPeakFilter(void * pIPcpu,
    TuAPInterpreterFunctionParameter * pParams) {
// parameter bq descr: biquad cascade
        TBQF_BiquadCascade * pbqc = ((
           TAPInterpreterVariable *) pParams[0].fp_pV)->
           pData;
// parameter index descr: index at the cascade
        int32_t index = pParams[1].fp_integer;
// parameter fs descr: sample frequency
        float fs = *((TAPgenericRationalVector *)
           pParams[2].fp_pD)->pVal;
// parameter fc descr: center frequency
        float fc = *((TAPgenericRationalVector *)
           pParams[3].fp_pD)->pVal;
// parameter q descr: quality
        float q = *((TAPgenericRationalVector *) pParams
           [4].fp_pD)->pVal;
// parameter g descr: gain (not in dB)
        float g = *((TAPgenericRationalVector *) pParams
           [5].fp_pD)->pVal;

        if (BQF_BQFcascadeInitPeak(pbqc, (unsigned int)
           index, fs, fc, q, g)) {
```

```
                    ((TAPInterpreterCPU *)pIPcpu)->EF =
                        -102;
        } else {
                    ((TAPInterpreterCPU *)pIPcpu)->pIP++;
        }
};
```

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:

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

```
        if (BQF_BQFcascadeInitLowFreqShelving(pbqc, (
           unsigned int) index, fs, f, q, g)) {
                ((TAPInterpreterCPU *)pIPcpu)->EF =
                   -103;
        } else {
                ((TAPInterpreterCPU *)pIPcpu)->pIP++;
        }
};
```

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:

```
// ===============================
// function implementation for
   initBiquadAsHighFreqShelvFilter(var id = 104)
// description: inits a biquad filter as high ferquency
   shelving filter
// ===============================
void HALfunc_ID104_initBiquadAsHighFreqShelvFilter(void
   * pIPcpu, TuAPInterpreterFunctionParameter * pParams)
    {
// parameter bq descr: biquad cascade
        TBQF_BiquadCascade * pbqc = ((
           TAPInterpreterVariable *) pParams[0].fp_pV)->
           pData;
// parameter index descr: index at the cascade
        int32_t index = pParams[1].fp_integer;
// parameter fs descr: sample frequnecy
        float fs = *((TAPgenericRationalVector *)
           pParams[2].fp_pD)->pVal;
// parameter f descr: cut/boost frequency
        float f = *((TAPgenericRationalVector *) pParams
           [3].fp_pD)->pVal;
// parameter q descr: quality
        float q = *((TAPgenericRationalVector *) pParams
           [4].fp_pD)->pVal;
```

```
// parameter g descr: gain (not in dB)
        float g = *((TAPgenericRationalVector *) pParams
           [5].fp_pD)->pVal;

        if (BQF_BQFcascadeInitHighFreqShelving(pbqc, (
           unsigned int) index, fs, f, q, g)) {
                ((TAPInterpreterCPU *)pIPcpu)->EF =
                   -104;
        } else {
                ((TAPInterpreterCPU *)pIPcpu)->pIP++;
        }
};
```

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:

```
// ===============================
// function implementation for convoluteBiquad(var id =
   110)
// description: convolute biquad with an input and
   generate an output
// ===============================
void HALfunc_ID110_convoluteBiquad(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {
// parameter x descr: input
        TAPgenericRationalVector * pXvec = (
           TAPgenericRationalVector *) pParams[0].fp_pD;
// parameter bqa descr: biquad cascade
        TBQF_BiquadCascade * pbqc = (TBQF_BiquadCascade
           *) pParams[1].fp_pD;
// parameter y descr: output
        TAPgenericRationalVector * pYvec = (
           TAPgenericRationalVector *) pParams[2].fp_pD;

        BQF_BQFcascadeConvolute(pbqc,pXvec->pVal, pXvec
           ->num, pYvec->pVal);

        ((TAPInterpreterCPU *)pIPcpu)->pIP++;
};
```

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:

```
// ================================
// function implementation for initNoisegate(var id =
    111)
// description: initialize a noisegate
// ================================
void HALfunc_ID111_initNoisegate(void * pIPcpu,
    TuAPInterpreterFunctionParameter * pParams) {
// parameter ng descr: noisegate
        TDynProc_Noisegate * pNG = (TDynProc_Noisegate
           *) pParams[0].fp_pD;
// parameter rmsTAV descr: time average value for the
    rms
        float rmsTAV = pParams[1].fp_rational;
// parameter AT descr: attack value for the smoothing
        float AT = pParams[2].fp_rational;
// parameter RT descr: release value for the smoothing
        float RT = pParams[3].fp_rational;
// parameter NT descr: noise cut off threshold
        float NT = pParams[4].fp_rational;
// parameter NS descr: slope
        float NS = pParams[5].fp_rational;

        DynProc_InitNoisegate(pNG, rmsTAV, AT, RT, NT,
           NS);
        ((TAPInterpreterCPU *)pIPcpu)->pIP++;
};
```

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:

```
// ================================
// function implementation for initExpander(var id =
   112)
// description: initialize a expander
// ================================
void HALfunc_ID112_initExpander(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {
// parameter exp descr: expander
        TDynProc_Expander * pExp = (TDynProc_Expander *)
            pParams[0].fp_pD;
// parameter rmsTAV descr: time average value for the
   rms
        float rmsTAV = pParams[1].fp_rational;
// parameter AT descr: attack value for the smoothing
        float AT = pParams[2].fp_rational;
// parameter RT descr: release value for the smoothing
        float RT = pParams[3].fp_rational;
// parameter ET descr: expander threshold
        float ET = pParams[4].fp_rational;
// parameter ES descr: slope
        float ES = pParams[5].fp_rational;

        DynProc_InitExpander(pExp, rmsTAV, AT, RT, ET,
           ES);
        ((TAPInterpreterCPU *)pIPcpu)->pIP++;
};
```

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:

```
// ================================
// function implementation for initCompressor(var id =
   113)
```

```
// description: initialize a compressor
// ================================
void HALfunc_ID113_initCompressor(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {
// parameter comp descr: compressor
        TDynProc_Compressor * pComp = (
           TDynProc_Compressor *) pParams[0].fp_pD;
// parameter rmsTAV descr: time average value for the
   rms
        float rmsTAV = pParams[1].fp_rational;
// parameter AT descr: attack value for the smoothing
        float AT = pParams[2].fp_rational;
// parameter RT descr: release value for the smoothing
        float RT = pParams[3].fp_rational;
// parameter CT descr: compressor threshold
        float CT = pParams[4].fp_rational;
// parameter CS descr: slope
        float CS = pParams[5].fp_rational;

        DynProc_InitCompressor(pComp, rmsTAV, AT, RT, CT
           , CS);
        ((TAPInterpreterCPU *)pIPcpu)->pIP++;
};
```

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:

```
// ================================
// function implementation for initLimiter(var id = 114)
// description: initialize a limiter
// ================================
void HALfunc_ID114_initLimiter(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {
// parameter lim descr: limiter
        TDynProc_Limiter * pLim = (TDynProc_Limiter *)
           pParams[0].fp_pD;
// parameter ATpeak descr: attack value for the peak
   detection
        float ATpeak = pParams[1].fp_rational;
```

```
// parameter RTpeak descr: release value for the peak
   detection
        float RTpeak = pParams[2].fp_rational;
// parameter ATsmooth descr: attack value for the
   smoothing
        float ATsmooth = pParams[3].fp_rational;
// parameter RTsmooth descr: release value for the
   smoothing
        float RTsmooth = pParams[4].fp_rational;
// parameter LT descr: limiter threshold
        float LT = pParams[5].fp_rational;
// parameter LS descr: slope
        float LS = pParams[6].fp_rational;

        DynProc_InitLimiter(pLim, ATpeak, RTpeak,
           ATsmooth, RTsmooth, LT, LS);
        ((TAPInterpreterCPU *)pIPcpu)->pIP++;
};
```

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:

```
// ===============================
// function implementation for calcNoisegate(var id =
   115)
// description: sends a stream of samples through a
   noisegate
// ===============================
void HALfunc_ID115_calcNoisegate(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {
// parameter x descr: input
        TAPgenericRationalVector * x = (
           TAPgenericRationalVector *) pParams[0].fp_pD;
// parameter ng descr: noisegate
        TDynProc_Noisegate * pNG = (TDynProc_Noisegate
           *) pParams[1].fp_pD;
// parameter y descr: output
        TAPgenericRationalVector * y = (
           TAPgenericRationalVector *) pParams[2].fp_pD;

```

```
        int i;
        for (i = 0; i < x->num; i++) {
                y->pVal[i] = DynProc_calcNoisegate(pNG,
                   x->pVal[i]);
        }
        ((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:

```
// ===============================
// function implementation for calcExpander(var id =
   116)
// description: sends a stream of samples through a
   expander
// ===============================
void HALfunc_ID116_calcExpander(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {
// parameter x descr: input
        TAPgenericRationalVector * x = (
           TAPgenericRationalVector *) pParams[0].fp_pD;
// parameter exp descr: expander
        TDynProc_Expander * pExp = (TDynProc_Expander *)
            pParams[1].fp_pD;
// parameter y descr: output
        TAPgenericRationalVector * y = (
           TAPgenericRationalVector *) pParams[2].fp_pD;

        int i;
        for (i = 0; i < x->num; i++) {
                y->pVal[i] = DynProc_calcExpander(pExp,
                   x->pVal[i]);
        }
        ((TAPInterpreterCPU *)pIPcpu)->pIP++;
};
```

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:

```
// ================================
// function implementation for calcCompressor(var id = 117)
// description: sends a stream of samples through a compressor
// ================================
void HALfunc_ID117_calcCompressor(void * pIPcpu, TuAPInterpreterFunctionParameter * pParams) {
// parameter x descr: input
        TAPgenericRationalVector * x = (TAPgenericRationalVector *) pParams[0].fp_pD;
// parameter comp descr: compressor
        TDynProc_Compressor * pComp = (TDynProc_Compressor *) pParams[1].fp_pD;
// parameter y descr: output
        TAPgenericRationalVector * y = (TAPgenericRationalVector *) pParams[2].fp_pD;

        int i;
        for (i = 0; i < x->num; i++) {
                y->pVal[i] = DynProc_calcCompressor(pComp, x->pVal[i]);
        }
        ((TAPInterpreterCPU *)pIPcpu)->pIP++;
};
```

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:

```
// ================================
// function implementation for calcLimiter(var id = 118)
// description: sends a stream of samples through a
   limiter
// ================================
void HALfunc_ID118_calcLimiter(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {
// parameter x descr: input
        TAPgenericRationalVector * x = (
           TAPgenericRationalVector *) pParams[0].fp_pD;
// parameter lim descr: limiter
        TDynProc_Limiter * pLim = (TDynProc_Limiter *)
           pParams[1].fp_pD;
// parameter y descr: output
        TAPgenericRationalVector * y = (
           TAPgenericRationalVector *) pParams[2].fp_pD;

        int i;
        for (i = 0; i < x->num; i++) {
                y->pVal[i] = DynProc_calcLimiter(pLim, x
                   ->pVal[i]);
        }
        ((TAPInterpreterCPU *)pIPcpu)->pIP++;
};
```

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:

```
// ================================
// function implementation for calcDelay(var id = 150)
// description: shifts the content of the delay
// ================================
```

```
void HALfunc_ID150_calcDelay(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {
// parameter x descr: input
        TAPgenericRationalVector * x = (
           TAPgenericRationalVector *) pParams[0].fp_pD;
// parameter delay descr: the delay
        TgenDelay * delay = (TgenDelay *) pParams[1].
           fp_pD;
// parameter y descr: output
        TAPgenericRationalVector * y = (
           TAPgenericRationalVector *) pParams[2].fp_pD;

        genDelay_readWrite(delay, x->pVal, y->pVal, y->
           num);
        ((TAPInterpreterCPU *)pIPcpu)->pIP++;
};
```

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:

```
// ================================
// function implementation for initDelay(var id = 151)
// description: inits the delay
// ================================
void HALfunc_ID151_initDelay(void * pIPcpu,
   TuAPInterpreterFunctionParameter * pParams) {
// parameter d descr: the delay
        TgenDelay * d = (TgenDelay *) pParams[0].fp_pD;
// parameter N descr: number of samples which correspond
    to the delay time (Tdelay = N * Ta)
        TAPgenericRationalVector * N = pParams[1].fp_pD;
// parameter Nindex descr: the index at the vector of N
        int Nindex = (int) pParams[2].fp_integer;
// parameter readToWriteOffset descr: the distance in
   Samples between the read and write position
        int readToWriteOffset = (int) pParams[3].
           fp_integer;

        // resize delay
        if (genDelay_resize(d, N->pVal[Nindex])) {
```

```
                ((TAPInterpreterCPU *)pIPcpu)->EF = -1;
                return;
        }
        // set pointer
        genDelay_shuffle(d,EgenDelayRWflag_readPointer,
           readToWriteOffset);
        // inc IP
        ((TAPInterpreterCPU *)pIPcpu)->pIP++;
};
```

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:

```
// ===============================
// driver uuid =2
// ===============================

// INFO
// the ADSP internal dataformat is BIG endian
// the UART send function sends data in little endian
// thats why we use the BIG endian function

// ------------------------------
// drv own functions
// ------------------------------

// struct for the drv data
typedef struct SADSPuartDrv {
        TAPMsgHeader                            h;
           //!< transmit header template
        TAPReceiveStateMachine          rxSM;
           //!< receive state machine
} TADSPuartDrv;

TADSPuartDrv gADSPuartDrv;

// ------------------------------
// drv own functions
// ------------------------------
```

```c

void drv_2_sendHeader (
                uint32_t msgId,
                uint32_t recv,
                uint32_t num,
                uint32_t length
        ) {
        gADSPuartDrv.h[eAPMsgHeaderPosition_msgTypeID] =
            msgId;
        gADSPuartDrv.h[eAPMsgHeaderPosition_msgNumber] =
            num;
        gADSPuartDrv.h[eAPMsgHeaderPosition_receiver] =
           recv;
        gADSPuartDrv.h[eAPMsgHeaderPosition_length] =
           length;
        sendUARTuintVectorBigEndian(gADSPuartDrv.h,
           sizeof(TAPMsgHeader)/sizeof(uint32_t));
}

// call back function of the uart driver
int drv_2_cbAPClient (unsigned int d) {
        return gADSPuartDrv.rxSM.state (&(gADSPuartDrv.
           rxSM), &d, 1);
}

//open an existing driver and bind it to the AP
int drv_2_open (void * pAP, struct SAPMsgDrv *pDrv) {
        // init header
        gADSPuartDrv.h[eAPMsgHeaderPosition_endian] =
           gAPendianFlag;
        gADSPuartDrv.h[eAPMsgHeaderPosition_sender] = ((
           TAP *) pAP)->nodeID;
        gADSPuartDrv.h[eAPMsgHeaderPosition_receiver] =
           dAPNodeID_ALL;
        gADSPuartDrv.h[eAPMsgHeaderPosition_msgTypeID] =
            0;
        gADSPuartDrv.h[eAPMsgHeaderPosition_msgNumber] =
            0;
        gADSPuartDrv.h[eAPMsgHeaderPosition_length] = 0;

        // save driver data
        pDrv->pDrvData = &gADSPuartDrv;

        // init receive state machine
        APInitReceiveStateMachine(
                        &gADSPuartDrv.rxSM,
                        ((TAP *)pAP)->MS,
                        pDrv
                );
        return 0;
}

//close the driver
int drv_2_close (struct SAPMsgDrv *pDrv) {
        return 0;
```

```
}
//destroys the driver
int drv_2_destroy (struct SAPMsgDrv *pDrv) {
        return 0;
}

//send raw data
int drv_2_sendRaw (struct SAPMsgDrv *pDrv, int amount,
    uint32_t *pData) {
        sendUARTraw4ByteVector(pData,amount);
        return 0;
}

//sends integer data in int32_t chunks in the sequence (
    LB0,HB0,LB1,HB1,LB2,HB2,LB3,HB3)
int drv_2_sendInteger32 (struct SAPMsgDrv *pDrv, int
    amount, int32_t * pData) {
        sendUARTintVectorBigEndian(pData,amount);
        return 0;
}
//sends float data in float chunks
int drv_2_sendFloat32 (struct SAPMsgDrv *pDrv, int
    amount, float * pData) {
        sendUARTFloatVector(pData,amount);
        return 0;
}



//acknowledge
int drv_2_ACK (void *pDrvData, uint32_t receiver,
    uint32_t mNum) {
        drv_2_sendHeader(eAPMsgTypes_ACK, receiver, mNum
            , 0);
        return 0;
}
//not acknowledge
int drv_2_NACK (void *pDrvData, uint32_t receiver,
    uint32_t mNum) {
        drv_2_sendHeader(eAPMsgTypes_NACK, receiver,
            mNum, 0);
        return 0;
}
//start sending a program
int drv_2_startPrg (void *pDrvData, uint32_t receiver,
    uint32_t mNum, int32_t globalVariableNumber, int32_t
    localVariableNumber, int32_t instructionNumber) {
        drv_2_sendHeader(eAPMsgTypes_startPrg, receiver,
            mNum, 0);
        sendUARTintVectorBigEndian(globalVariableNumber
            ,1);
        sendUARTintVectorBigEndian(localVariableNumber
            ,1);
        sendUARTintVectorBigEndian(instructionNumber,1);
        return 0;
```

```
}

//sends a variable
int drv_2_sendVariable (void *pDrvData, uint32_t
   receiver, uint32_t mNum, int32_t index, int32_t
   varTypeID, int32_t num) {
         drv_2_sendHeader(eAPMsgTypes_sendVariable,
            receiver, mNum, 3);
         sendUARTintVectorBigEndian(&index,1);
         sendUARTintVectorBigEndian(&varTypeID,1);
         sendUARTintVectorBigEndian(&num,1);
         return 0;
}
//sends an instruction
int drv_2_sendInstruction (void *pDrvData, uint32_t
   receiver, uint32_t mNum, int32_t index, uint32_t *
   fbc) {
         drv_2_sendHeader(eAPMsgTypes_sendInstruction,
            receiver, mNum, dAPInterpreterFuncMaxParams +
             1);
         sendUARTintVectorBigEndian(&index,1);
         sendUARTraw4ByteVector(fbc,sizeof(int32_t)*
            dAPInterpreterFuncMaxParams);
         return 0;
}
//sings that the program transmission has completed
int drv_2_endPrg (void *pDrvData, uint32_t receiver,
   uint32_t mNum) {
         drv_2_sendHeader(eAPMsgTypes_endPrg, receiver,
            mNum, 0);
         return 0;
}
//stops the AP
int drv_2_stop (void *pDrvData, uint32_t receiver,
   uint32_t mNum) {
         drv_2_sendHeader(eAPMsgTypes_stop, receiver,
            mNum, 0);
         return 0;
}
//the AP executes one instruction
int drv_2_step (void *pDrvData, uint32_t receiver,
   uint32_t mNum) {
         drv_2_sendHeader(eAPMsgTypes_step, receiver,
            mNum, 0);
         return 0;
}
//the AP runs the program
int drv_2_run (void *pDrvData, uint32_t receiver,
   uint32_t mNum) {
         drv_2_sendHeader(eAPMsgTypes_run, receiver, mNum
            , 0);
         return 0;
}
//a variable going to be updated
int drv_2_updateVariable (void *pDrvData, uint32_t
```

```
    receiver, uint32_t mNum, int32_t gIndex, int32_t
    dataElements) {
        drv_2_sendHeader(eAPMsgTypes_updateVariable,
            receiver, mNum, dataElements + 1);
        sendUARTintVectorBigEndian(&gIndex,1);
        return 0;
}
//a AP is going to be logged in to the system
int drv_2_login (void *pDrvData, uint32_t receiver,
    uint32_t mNum) {
        drv_2_sendHeader(eAPMsgTypes_login, receiver,
            mNum, 0);
        return 0;
}
//a AP is going to be logged out of the system
int drv_2_logout (void *pDrvData, uint32_t receiver,
    uint32_t mNum) {
        drv_2_sendHeader(eAPMsgTypes_logout, receiver,
            mNum, 0);
        return 0;
}
```

3.4.2 driver 2 (MSP430-169STK)

Informations:

description:	MSP430uartDRV

includes:

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

code:

```
// ================================
// driver uuid =3
// ================================

// struct for the drv data
typedef struct SMSP430uartDrv {
        TAPMsgHeader                                h;
                    //!< transmit header template
        TAPReceiveStateMachine          rxSM;
            //!< receive state machine
} TMSP430uartDrv;

TMSP430uartDrv gMSP430uartDrv;

// ------------------------------
// drv own functions
// ------------------------------

void drv_3_sendHeader (
```

```
                uint32_t msgId,
                uint32_t recv,
                uint32_t num,
                uint32_t length
        ) {
        gMSP430uartDrv.h[eAPMsgHeaderPosition_msgTypeID] = msgId;
        gMSP430uartDrv.h[eAPMsgHeaderPosition_msgNumber] = num;
        gMSP430uartDrv.h[eAPMsgHeaderPosition_receiver] = recv;
        gMSP430uartDrv.h[eAPMsgHeaderPosition_length] = length;
        msp430_UART_send((unsigned char *)gMSP430uartDrv.h,sizeof(TAPMsgHeader));
}


// function for feeding the recv state machine
void drv_3_feedRecvStateM () {
        uint32_t d;
        while (!stjFIFO_readElement(&gMsp430_uartFIFO ,&d)) {
                gMSP430uartDrv.rxSM.state (&(gMSP430uartDrv.rxSM), &d, 1);
        }
}
//open an existing driver and bind it to the AP
int drv_3_open (void * pAP, struct SAPMsgDrv *pDrv) {
        // init header
        gMSP430uartDrv.h[eAPMsgHeaderPosition_endian] = gAPendianFlag;
        gMSP430uartDrv.h[eAPMsgHeaderPosition_sender] = ((TAP *) pAP)->nodeID;
        gMSP430uartDrv.h[eAPMsgHeaderPosition_receiver] = dAPNodeID_ALL;
        gMSP430uartDrv.h[eAPMsgHeaderPosition_msgTypeID] = 0;
        gMSP430uartDrv.h[eAPMsgHeaderPosition_msgNumber] = 0;
        gMSP430uartDrv.h[eAPMsgHeaderPosition_length] = 0;

        // save driver data
        pDrv->pDrvData = &gMSP430uartDrv;

        // init receive state machine
        APInitReceiveStateMachine(
                        &gMSP430uartDrv.rxSM,
                        ((TAP *)pAP)->MS,
                        pDrv
                );
        return 0;
}

```

```c
//close the driver
int drv_3_close (struct SAPMsgDrv *pDrv) {
	return 0;
}
//destroys the driver
int drv_3_destroy (struct SAPMsgDrv *pDrv) {
	return 0;
}

//sends raw data in uint32_t chunks
int drv_3_sendRaw (struct SAPMsgDrv *pDrv, int amount, uint32_t * pData) {
	msp430_UART_send((unsigned char *)pData,amount*sizeof(uint32_t));
	return 0;
}
//sends integer data in int32_t chunks in the sequence (LB0,HB0,LB1,HB1,LB2,HB2,LB3,HB3)
int drv_3_sendInteger32 (struct SAPMsgDrv *pDrv, int amount, int32_t * pData) {
	msp430_UART_send((unsigned char *)pData,amount*sizeof(uint32_t));
	return 0;
}
//sends float data in float chunks
int drv_3_sendFloat32 (struct SAPMsgDrv *pDrv, int amount, float * pData) {
	msp430_UART_send((unsigned char *)pData,amount*sizeof(uint32_t));
	return 0;
}

//acknowledge
int drv_3_ACK (void *pDrvData, uint32_t receiver, uint32_t mNum) {
	drv_3_sendHeader(eAPMsgTypes_ACK, receiver, mNum, 0);
	return 0;
}
//not acknowledge
int drv_3_NACK (void *pDrvData, uint32_t receiver, uint32_t mNum) {
	drv_3_sendHeader(eAPMsgTypes_NACK, receiver, mNum, 0);
	return 0;
}
//start sending a program
int drv_3_startPrg (void *pDrvData, uint32_t receiver, uint32_t mNum, int32_t globalVariableNumber, int32_t localVariableNumber, int32_t instructionNumber) {
	drv_3_sendHeader(eAPMsgTypes_startPrg, receiver, mNum, 0);
	msp430_UART_send((unsigned char *)&globalVariableNumber,sizeof(int32_t));
	msp430_UART_send((unsigned char *)&
```

```
            localVariableNumber,sizeof(int32_t));
        msp430_UART_send((unsigned char *)&
            instructionNumber,sizeof(int32_t));
        return 0;
}

//sends a variable
int drv_3_sendVariable (void *pDrvData, uint32_t
    receiver, uint32_t mNum, int32_t index, int32_t
    varTypeID, int32_t num) {
        drv_3_sendHeader(eAPMsgTypes_sendVariable,
            receiver, mNum, 3);
        msp430_UART_send((unsigned char *)&index,sizeof(
            int32_t));
        msp430_UART_send((unsigned char *)&varTypeID,
            sizeof(int32_t));
        msp430_UART_send((unsigned char *)&num,sizeof(
            int32_t));
        return 0;
}
//sends an instruction
int drv_3_sendInstruction (void *pDrvData, uint32_t
    receiver, uint32_t mNum, int32_t index, uint32_t *
    fbc) {
        drv_3_sendHeader(eAPMsgTypes_sendInstruction,
            receiver, mNum, dAPInterpreterFuncMaxParams +
             1);
        msp430_UART_send((unsigned char *)&index,sizeof(
            int32_t));
        msp430_UART_send((unsigned char *)fbc,sizeof(
            int32_t) * (dAPInterpreterFuncMaxParams+1));
        return 0;
}
//sings that the program transmission has completed
int drv_3_endPrg (void *pDrvData, uint32_t receiver,
    uint32_t mNum) {
        drv_3_sendHeader(eAPMsgTypes_endPrg, receiver,
            mNum, 0);
        return 0;
}
//stops the AP
int drv_3_stop (void *pDrvData, uint32_t receiver,
    uint32_t mNum) {
        drv_3_sendHeader(eAPMsgTypes_stop, receiver,
            mNum, 0);
        return 0;
}
//the AP executes one instruction
int drv_3_step (void *pDrvData, uint32_t receiver,
    uint32_t mNum) {
        drv_3_sendHeader(eAPMsgTypes_step, receiver,
            mNum, 0);
        return 0;
}
//the AP runs the program
```

```
int drv_3_run (void *pDrvData, uint32_t receiver,
   uint32_t mNum) {
        drv_3_sendHeader(eAPMsgTypes_run, receiver, mNum
            , 0);
        return 0;
}
//a variable going to be updated
int drv_3_updateVariable (void *pDrvData, uint32_t
   receiver, uint32_t mNum, int32_t gIndex, int32_t
   dataElements) {
        drv_3_sendHeader(eAPMsgTypes_updateVariable,
            receiver, mNum, dataElements + 1);
        msp430_UART_send((unsigned char *)&gIndex,sizeof
            (int32_t));
        return 0;
}
//a AP is going to be logged in to the system
int drv_3_login (void *pDrvData, uint32_t receiver,
   uint32_t mNum) {
        drv_3_sendHeader(eAPMsgTypes_login, receiver,
            mNum, 0);
        return 0;
}
//a AP is going to be logged out of the system
int drv_3_logout (void *pDrvData, uint32_t receiver,
   uint32_t mNum) {
        drv_3_sendHeader(eAPMsgTypes_logout, receiver,
            mNum, 0);
        return 0;
}
```

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:

```
// ================================
// driver uuid =1
// ================================
// -------------------------------
// drv own functions
// -------------------------------

#define dAPClientServerAdminPort (50000)
```

```
#define dAPClientInitBuffer (1024)

// struct for the drv data
typedef struct SAPTCPIPdrv {
        TAPClient cl;                    //!< tcp/ip client
        TAPMsgHeader txHeader;           //!< transmit header template
        TAPReceiveStateMachine rxSM;     //!< receive state machine
} TAPTCPIPdrv;

// ------------------------------
// drv own functions
// ------------------------------

int drv_1_sendHeader (
                TAPClient * pCl,
                uint32_t * pH,
                uint32_t msgId,
                uint32_t recv,
                uint32_t num,
                uint32_t length
        ) {
        pH[eAPMsgHeaderPosition_msgTypeID] = msgId;
        pH[eAPMsgHeaderPosition_msgNumber] = num;
        pH[eAPMsgHeaderPosition_receiver] = recv;
        pH[eAPMsgHeaderPosition_length] = length;
        if (APclient_send(pCl,sizeof(TAPMsgHeader),(uint8_t *)pH)) return -1;
        return 0;
}

// call back function of the TCP/IP driver
int drv_1_cbAPClient (void *pvDC, uint16_t number, uint8_t * pData) {
        TAPReceiveStateMachine * pSM = pvDC;
        return pSM->state (pSM, (uint32_t *) pData, number / sizeof(uint32_t));
}

//open an existing driver and bind it to the AP
int drv_1_open (void * pAP, struct SAPMsgDrv *pDrv) {
        TAPTCPIPdrv * pDC = NULL;

        pDC = malloc(sizeof(TAPTCPIPdrv));
        if (!pDC) return -1;
        if (APclient_create(&(pDC->cl),dAPClientServerAdminPort,dAPClientInitBuffer,&(pDC->rxSM),drv_1_cbAPClient)) return -2;
        // init header
        pDC->txHeader[eAPMsgHeaderPosition_endian] = (uint32_t) gAPendianFlag;
        pDC->txHeader[eAPMsgHeaderPosition_sender] = ((TAP *) pAP)->nodeID;
```

```
        pDC->txHeader[eAPMsgHeaderPosition_receiver] = dAPNodeID_ALL;
        pDC->txHeader[eAPMsgHeaderPosition_msgTypeID] = 0;
        pDC->txHeader[eAPMsgHeaderPosition_msgNumber] = 0;
        pDC->txHeader[eAPMsgHeaderPosition_length] = 0;

        // save driver data
        pDrv->pDrvData = pDC;

        // init receive state machine
        APInitReceiveStateMachine(
                        &pDC->rxSM,
                        ((TAP *)pAP)->MS,
                        pDrv
                );
        return 0;
}
//close the driver
int drv_1_close (struct SAPMsgDrv *pDrv) {
        TAPTCPIPdrv * pDC = (TAPTCPIPdrv *)(pDrv->pDrvData);
        if (pDC) {
                APclient_close(&pDC->cl);
        }
        return 0;
}
//destroys the driver
int drv_1_destroy (struct SAPMsgDrv *pDrv) {
        TAPTCPIPdrv * pDC = (TAPTCPIPdrv *)(pDrv->pDrvData);
        if (pDC) {
                free(pDC);
                pDrv->pDrvData = NULL;
        }
        return 0;
}

//sends raw data
int drv_1_sendRaw (struct SAPMsgDrv *pDrv, int amount, uint32_t * pData) {
        TAPTCPIPdrv * pDC = (TAPTCPIPdrv *)(pDrv->pDrvData);
        return APclient_send(&(pDC->cl),amount*sizeof(uint32_t),(uint8_t *)pData);
}

//a AP is going to be logged out of the system
int drv_1_sendInteger32 (struct SAPMsgDrv *pDrv, int amount, int32_t * pData) {
        TAPTCPIPdrv * pDC = (TAPTCPIPdrv *)(pDrv->pDrvData);

        if (gAPendianFlag == eAP_littleEndian) {
```

```
			return APclient_send(&(pDC->cl),amount*
			   sizeof(uint32_t),(uint8_t *)pData);
		} else {
			int i;
			int32_t v;
			for (i = 0; i < amount; i++) {
				v = *pData;
				APendianConversation32Bit((
				   uint32_t *)&v,
				   eAP_littleEndian);
				if (APclient_send(&(pDC->cl),
				   sizeof(int32_t),(uint8_t *)&v
				   )) {
					return -1;
				}
				pData++;
			}
		}
		return 0;
}

//a AP is going to be logged out of the system
int drv_1_sendFloat32 (struct SAPMsgDrv *pDrv, int
   amount, float * pData) {
		TAPTCPIPdrv * pDC = (TAPTCPIPdrv *)(pDrv->
		   pDrvData);

		return APclient_send(&(pDC->cl),amount*sizeof(
		   float),(uint8_t *)pData);
}



//acknowledge
int drv_1_ACK (void *pDrvData, uint32_t receiver,
   uint32_t mNum) {
		TAPClient * pCl = &((TAPTCPIPdrv *)(pDrvData))->
		   cl;
		uint32_t * pH = ((TAPTCPIPdrv *)(pDrvData))->
		   txHeader;

		return drv_1_sendHeader(pCl, pH, eAPMsgTypes_ACK
		   , receiver, mNum, 0);
}
//not acknowledge
int drv_1_NACK (void *pDrvData, uint32_t receiver,
   uint32_t mNum) {
		TAPClient * pCl = &((TAPTCPIPdrv *)(pDrvData))->
		   cl;
		uint32_t * pH = ((TAPTCPIPdrv *)(pDrvData))->
		   txHeader;

		return drv_1_sendHeader(pCl, pH,
		   eAPMsgTypes_NACK, receiver, mNum, 0);
}
```

```c
//start sending a program
int drv_1_startPrg (void *pDrvData, uint32_t receiver, uint32_t mNum, int32_t globalVariableNumber, int32_t localVariableNumber, int32_t instructionNumber) {
        TAPClient * pCl = &((TAPTCPIPdrv *)(pDrvData))->cl;
        uint32_t * pH = ((TAPTCPIPdrv *)(pDrvData))->txHeader;
        if (
                                drv_1_sendHeader(pCl, pH, eAPMsgTypes_startPrg, receiver, mNum, 0) ||
                                APclient_send(pCl,sizeof(int32_t),(uint8_t *)&globalVariableNumber) ||
                                APclient_send(pCl,sizeof(int32_t),(uint8_t *)&localVariableNumber) ||
                                APclient_send(pCl,sizeof(int32_t),(uint8_t *)&instructionNumber)
                ) return -10;
        return 0;
}

//sends a variable
int drv_1_sendVariable (void *pDrvData, uint32_t receiver, uint32_t mNum, int32_t index, int32_t varTypeID, int32_t num) {
        TAPClient * pCl = &((TAPTCPIPdrv *)(pDrvData))->cl;
        uint32_t * pH = ((TAPTCPIPdrv *)(pDrvData))->txHeader;
        if (
                                drv_1_sendHeader(pCl, pH, eAPMsgTypes_sendVariable, receiver, mNum, 3) ||
                                APclient_send(pCl,sizeof(int32_t),(uint8_t *)&index) ||
                                APclient_send(pCl,sizeof(int32_t),(uint8_t *)&varTypeID) ||
                                APclient_send(pCl,sizeof(int32_t),(uint8_t *)&num)
                ) return -10;
        return 0;
}
//sends an instruction
int drv_1_sendInstruction (void *pDrvData, uint32_t receiver, uint32_t mNum, int32_t index, uint32_t * fbc) {
        TAPClient * pCl = &((TAPTCPIPdrv *)(pDrvData))->cl;
        uint32_t * pH = ((TAPTCPIPdrv *)(pDrvData))->txHeader;
        if (
```

```
                        drv_1_sendHeader(pCl, pH, eAPMsgTypes_sendInstruction, receiver, mNum, dAPInterpreterFuncMaxParams + 1) ||
                        APclient_send(pCl,sizeof(int32_t),(uint8_t *)&index) ||
                        APclient_send(pCl,sizeof(int32_t)*dAPInterpreterFuncMaxParams,(uint8_t *)fbc)
                ) return -10;
        return 0;
}
//sings that the program transmission has completed
int drv_1_endPrg (void *pDrvData, uint32_t receiver, uint32_t mNum) {
        TAPClient * pCl = &((TAPTCPIPdrv *)(pDrvData))->cl;
        uint32_t * pH = ((TAPTCPIPdrv *)(pDrvData))->txHeader;

        return drv_1_sendHeader(pCl, pH, eAPMsgTypes_endPrg, receiver, mNum, 0);
}
//stops the AP
int drv_1_stop (void *pDrvData, uint32_t receiver, uint32_t mNum) {
        TAPClient * pCl = &((TAPTCPIPdrv *)(pDrvData))->cl;
        uint32_t * pH = ((TAPTCPIPdrv *)(pDrvData))->txHeader;

        return drv_1_sendHeader(pCl, pH, eAPMsgTypes_stop, receiver, mNum, 0);
}
//the AP executes one instruction
int drv_1_step (void *pDrvData, uint32_t receiver, uint32_t mNum) {
        TAPClient * pCl = &((TAPTCPIPdrv *)(pDrvData))->cl;
        uint32_t * pH = ((TAPTCPIPdrv *)(pDrvData))->txHeader;

        return drv_1_sendHeader(pCl, pH, eAPMsgTypes_step, receiver, mNum, 0);
}
//the AP runs the program
int drv_1_run (void *pDrvData, uint32_t receiver, uint32_t mNum) {
        TAPClient * pCl = &((TAPTCPIPdrv *)(pDrvData))->cl;
        uint32_t * pH = ((TAPTCPIPdrv *)(pDrvData))->txHeader;

        return drv_1_sendHeader(pCl, pH, eAPMsgTypes_run
```

```
, receiver, mNum, 0);
}
//a variable going to be updated
int drv_1_updateVariable (void *pDrvData, uint32_t receiver, uint32_t mNum, int32_t gIndex, int32_t dataElements) {
        TAPClient * pCl = &((TAPTCPIPdrv *)(pDrvData))->cl;
        uint32_t * pH = ((TAPTCPIPdrv *)(pDrvData))->txHeader;
        if (
                        drv_1_sendHeader(pCl, pH, eAPMsgTypes_updateVariable, receiver, mNum, dataElements + 1) ||
                        APclient_send(pCl,sizeof(int32_t),(uint8_t *)&gIndex)
                ) return -10;
        return 0;
}
//a AP is going to be logged in to the system
int drv_1_login (void *pDrvData, uint32_t receiver, uint32_t mNum) {
        TAPClient * pCl = &((TAPTCPIPdrv *)(pDrvData))->cl;
        uint32_t * pH = ((TAPTCPIPdrv *)(pDrvData))->txHeader;

        return drv_1_sendHeader(pCl, pH, eAPMsgTypes_login, receiver, mNum, 0);
}
//a AP is going to be logged out of the system
int drv_1_logout (void *pDrvData, uint32_t receiver, uint32_t mNum) {
        TAPClient * pCl = &((TAPTCPIPdrv *)(pDrvData))->cl;
        uint32_t * pH = ((TAPTCPIPdrv *)(pDrvData))->txHeader;

        return drv_1_sendHeader(pCl, pH, eAPMsgTypes_logout, receiver, mNum, 0);
}
```

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:

```c
// ================================
// AP uuid = 6
// ================================

#define WAVsampleCache (65536)

// the wav modul global var
TStjWAVmodule gWAVModule;


// inits the AP
int APinit (
                        TAP *       pAP,
                        TAPNodeID   nodeID,
                        const TAPMsgDrv *       pDrvList,
                        const int   driverNumber,
                        size_t      messagePoolSize,
                        int                 sysEndian
                )
{
        pAP->nodeID = nodeID;
        pAP->pNodeList = NULL;
        pAP->pDrvList = pDrvList;
        pAP->sysEndian = sysEndian;
        pAP->driverNumber = driverNumber;
        pAP->msgSysMMU = AP_MMU_create(messagePoolSize);
        pAP->IP = APInterpreterCreate(pAP);
        pAP->MS = APMScreate (pAP->msgSysMMU,sysEndian);
        pAP->msgNumber = 0;
        pAP->APstate = eAPstate_idle;

        if(
                (!pAP->msgSysMMU)||
                (!pAP->IP)||
                (!pAP->MS)
                ) return -1;
        // login the ap to the message system

        // init wav module
        TStjWAVOpenInfo wavIOs[] = {
                        {1,"input.wav",1,44100,0,WAVsampleCache},
                        {2,"bypass.wav",0,44100,0,
```

```
                                WAVsampleCache},
                             {3,"subbass.wav",0,44100,0,
                                WAVsampleCache},
                             {4,"lowerVoice.wav",0,44100,0,
                                WAVsampleCache},
                             {5,"upperVoice.wav",0,44100,0,
                                WAVsampleCache},
                             {6,"harmonics.wav",0,44100,0,
                                WAVsampleCache},
        };

        if (WAVmoduleInit (sizeof(wavIOs) / sizeof(
           TStjWAVOpenInfo),wavIOs,&gWAVModule)) {
                return -2;
        }

        return TX_login(pAP);
}

// deletes the AP
void APdelete (TAP * pAP)
{
        // close wav module
        WAVmoduleExit(&gWAVModule);

        APMSdelete (pAP->MS);
        APInterpreterDelete(pAP->IP);
        AP_MMU_delete(pAP->msgSysMMU);
}

// find a node at the list
TAPNode * APfindNode(TAP * pAP, TAPNodeID nodeID) {
        TAPNode * pN = pAP->pNodeList;
        while (pN) {
                if (pN->nodeID == nodeID) return pN;
                pN = pN->pNext;
        };
        return NULL;
}

// adds a new node to the node list
int APaddNode(TAP * pAP, TAPNodeID newNodeID, const
   TAPMsgDrv * pDrv) {
        if (APfindNode(pAP,newNodeID)) return 1;
        TAPNode * pN = (TAPNode *) malloc(sizeof(TAPNode
           ));
        if (!pN) return -1;
        pN->nodeID = newNodeID;
        pN->pDrv = pDrv;
        pN->pNext = pAP->pNodeList;
        pAP->pNodeList = pN;
        return 0;
}

// removes a node from the node list
```

```
void APremoveNode(TAP * pAP, TAPNodeID nodeID){
        TAPNode * pAntN = pAP->pNodeList; // antecessor
           node
        TAPNode * pActN = pAP->pNodeList; // actual node

        while (pActN) {
                // compare node id's
                if (pActN->nodeID == nodeID) {
                        // unchain

                        // check if we at the first
                           position at the list
                        if (pAP->pNodeList == pAntN) {
                                // reset the pointer
                                pAP->pNodeList = pActN->
                                   pNext;
                        } else {
                                // set the antecessor
                                pAntN->pNext = pActN->
                                   pNext;
                        }
                        // free node
                        free(pActN);
                        // and abort
                        return;
                }
                // the actual element becomes the
                   precessor element
                pAntN = pActN;
                pActN = pActN->pNext;
        }
}

// get a new message number
unsigned int APgetNewMessageNumber (TAP *pAP) {
        pAP->msgNumber++;
        return pAP->msgNumber;
}

// find the driver associated with der nodeID
const TAPMsgDrv * APfindDrvBySenderID (TAP * pAP,
   TAPNodeID node) {
        TAPNode * pN = pAP->pNodeList;
        while (pN) {
                if (pN->nodeID == node) {
                        return pN->pDrv;
                }
                pN = pN->pNext;
        }
        return NULL;
}

// runs the AP
int APrun(TAP *pAP) {
        pAP->APstate = eAPstate_run;
```

```
        return 0;
}


typedef struct SAPrealMMUMemory {
        int *   pData;                  // the data
        size_t  count;                  // amount of data elements
        struct SAPrealMMUMemory *       pNext;          // next element
        struct SAPrealMMUMemory *       pPrev;          // previous element
} TAPrealMMUMemory;

//the mmu type
typedef struct SAPrealMMU {
        int *                                   memory;                 // the memory bolck
        TAPrealMMUMemory *              pStart;                 // first element
        TAPrealMMUMemory *              pEnd;                   // second element
        TAPrealMMUMemory *              pUnusedList;            // list with the unused elements
        int *   pUnusedData;            // pointer to the unused memory
        size_t  elementsAvailable;      // amount of elements witch are available without using the garbage collector
        size_t  totalAvailable;         // total amount of free bytes

} TAPrealMMU;

// ========================================
// memory entry functions
// ========================================

// a little macro for unchaining an element
#define DMemoryEntryUnchain(pM) \
        if (pM->pNext) pM->pNext->pPrev = pM->pPrev; \
        if (pM->pPrev) pM->pPrev->pNext = pM->pNext


//creates an memory entry
TAPrealMMUMemory * MemoryEntry_create () {
        TAPrealMMUMemory * pM = NULL;

        pM = (TAPrealMMUMemory *) malloc(sizeof(TAPrealMMUMemory));
```

```
        if (!pM) return NULL;
        pM->pData = NULL;
        pM->count = 0;
        pM->pNext = NULL;
        pM->pPrev = NULL;

        return pM;
}

//deletes an memory Entry
void MemoryEntry_delete (
                TAPrealMMUMemory * pM    // the memory to
                    delete
                )
{
        // put the entry out of the chain
        DMemoryEntryUnchain(pM);
        // now we delete it
        free(pM);
}

// ========================================
// mmu helper
// ========================================

//alloc if needed a new memory entry
TAPrealMMUMemory * MMU_helper_createMemoryEntry (
                TAPrealMMU *     pMMU              // MMU
                   structure to init
                )
{
        // check if we have to alloc a new memory entry
        if (!pMMU->pUnusedList) return
           MemoryEntry_create();
        // no there is some left at the list
        TAPrealMMUMemory * pM;
        // take the first one
        pM = pMMU->pUnusedList;
        // reset the list
        pMMU->pUnusedList = pM->pNext;
        // now unchain the element (for sure)
        DMemoryEntryUnchain(pM);
        // set the element pointers
        pM->pNext = NULL;
        pM->pPrev = NULL;
        return pM;
}

//the garbage collector
void MMU_helper_garbageCollector (
                TAPrealMMU *     pMMU              // MMU
                   structure to init
                )
{
        TAPrealMMUMemory * pM = pMMU->pStart;
```

```
        int * pD = pMMU->memory;
        while (pM) {
                // check if we have to move the data
                if (pD != pM->pData) {
                        // move the data
                        memmove(pD,pM->pData,pM->count*sizeof(int));
                }
                // reset the destination pointer
                pD += pM->count;
                pM = pM->pNext;
        }
        // compressing memory finished
        // set the mmu vars new
        pMMU->elementsAvailable = pMMU->totalAvailable;
        pMMU->pUnusedData = pD;
}



// create a mmu
TAPMMU AP_MMU_create (size_t elementsNumber) {
        TAPrealMMU * pMMU;

        pMMU = (TAPrealMMU *) malloc (sizeof(TAPrealMMU));
        if (!pMMU) return NULL;



        // setup lists
        pMMU->pStart = NULL;
        pMMU->pEnd = NULL;
        pMMU->pUnusedList = NULL;

        pMMU->elementsAvailable =elementsNumber;
        pMMU->pUnusedData = (int *) malloc (pMMU->elementsAvailable);
        pMMU->totalAvailable = pMMU->elementsAvailable;
        return pMMU;
}

// destroying the mmu
void AP_MMU_delete (TAPMMU mmu) {
        TAPrealMMU * pMMU = (TAPrealMMU *) mmu;

        TAPrealMMUMemory * pM;
        TAPrealMMUMemory * pMnext;

// 1. delete al mmu entry's
        // 1.1 unused entry
        pM = pMMU->pUnusedList;
        while (pM) {
                pMnext = pM->pNext;
                MemoryEntry_delete(pM);
```

```
                pM = pMnext;
        }
        pMMU->pUnusedList = NULL;
        // 1.2 used blocks
        pM = pMMU->pStart;
        while (pM) {
                pMnext = pM->pNext;
                MemoryEntry_delete(pM);
                pM = pMnext;
        }
        pMMU->pStart = NULL;
        pMMU->pEnd = NULL;
// 2. delete mmu memory
        free (pMMU->memory);


}
// getting memmory from the mmu
TAPMMUmemmory AP_MMU_get (TAPMMU mmu, size_t elements) {
        TAPrealMMU * pMMU = (TAPrealMMU *) mmu;

        // check if there is enough space
        if (pMMU->totalAvailable < elements) goto error;
        // check if we have to use the garbage collector
        if (pMMU->elementsAvailable < elements) {
                // start garbage collector
                MMU_helper_garbageCollector(pMMU);
        }
        // we have enough memory so let's allocate some

        // get a new entry
        TAPrealMMUMemory * pM;
        pM = MMU_helper_createMemoryEntry(pMMU);
        if (!pM) return NULL;
        // get some memory
        pM->pData = pMMU->pUnusedData;
        pM->count = elements;
        // refresh data
        pMMU->pUnusedData += elements;
        pMMU->totalAvailable -= elements;
        pMMU->elementsAvailable -= elements;
        // insert memory element at the end of the list
           and update last element
        pM->pPrev = pMMU->pEnd;
        if (pMMU->pEnd) pMMU->pEnd->pNext = pM;
        if (!pMMU->pStart) pMMU->pStart = pM;
        pMMU->pEnd = pM;

        return pM;
error:

        return NULL;
}

// free memmory from the mmu
```

```
void AP_MMU_free (TAPMMU mmu, TAPMMUmemmory memory) {
        TAPrealMMU * pMMU = (TAPrealMMU *) mmu;
        TAPrealMMUMemory * pM = (TAPrealMMUMemory *) memory;

        if (!pM) return;
        // set mmu settings
        if (pMMU->pStart == pM) pMMU->pStart = pM->pNext;
        if (pMMU->pEnd == pM) pMMU->pEnd = pM->pPrev;
        // unchain element
        DMemoryEntryUnchain (pM);
        // and put it to the chain of unused
        pM->pNext = pMMU->pUnusedList;
        pM->pPrev = NULL;
        if (pMMU->pUnusedList) {
                pMMU->pUnusedList->pPrev = pM;
        }
        pMMU->pUnusedList = pM;
        // now set the mmu data new
        pMMU->totalAvailable += pM->count;

}

// getting access to the MMU data
void * AP_MMU_getData (TAPMMUmemmory memory) {
        TAPrealMMUMemory * pM = (TAPrealMMUMemory *) memory;
        return pM->pData;
}

// the real interpreter
typedef struct SAPrealInterpreter {
        int                     state;                  // state of the IP
        int                     nextState;              // the next state of the IP
        TAPInterpreterCPU                       cpu;    // the IP core
        TAPInterpreterFuncCall *        code;           // the code
        int32_t         instructionCount;       // number of instructions at the code
        TAPInterpreterVariable *        variables;      // the variables
        int32_t         variableCount;          // number of the variables
        int                     sysEndian;              // endian of the system
```

```
} TAPrealInterpreter;

// create a new interpreter
TAPInterpreter APInterpreterCreate (void * pAP) {
        TAPrealInterpreter * pIP = NULL;
        pIP = (TAPrealInterpreter *) malloc (sizeof(TAPrealInterpreter));
        if (!pIP) return NULL;

        pIP->state = eAPInterpreterState_idle;
        pIP->nextState = eAPInterpreterState_idle;
        pIP->sysEndian = ((TAP *)pAP)->sysEndian;
        pIP->cpu.CF = 0;
        pIP->cpu.EF = 0;
        pIP->cpu.pCodeStart = NULL;
        pIP->cpu.pCodeEnd = NULL;
        pIP->cpu.pIP = NULL;

        pIP->code = NULL;
        pIP->instructionCount = 0;

        pIP->variables = NULL;
        pIP->variableCount = 0;

        return pIP;
}

// cleans the interpreter
void APInterpreterClean (TAPInterpreter IP) {
        TAPrealInterpreter * pIP = (TAPrealInterpreter *) IP;

        // clean code
        if (pIP->code) {
                free (pIP->code);
                pIP->code = NULL;
        }
        pIP->instructionCount = 0;

        // clean variables
        TAPInterpreterVariable * pV = pIP->variables;
        int i;
        for (i = 0; i < pIP->variableCount; i++) {
                if (pV->pVI) pV->pVI->pFkt_delete(pV->pData);
                pV++;
        }
        if (pIP->variables) {
                free (pIP->variables);
                pIP->variables = NULL;
        }
        pIP->variableCount = 0;

}

```

```
// deletes the interpreter
void APInterpreterDelete (TAPInterpreter IP) {
        TAPrealInterpreter * pIP = (TAPrealInterpreter
           *) IP;
        APInterpreterClean(IP);
        free (pIP);
}

int APInterpreterStateRun(TAPInterpreter IP) {
        TAPrealInterpreter * pIP = (TAPrealInterpreter
           *) IP;
        TAPInterpreterFuncCall * pFC;

        // setup cpu
        pIP->cpu.CF = 0;
        pIP->cpu.EF = 0;
        pIP->cpu.pIP = pIP->code;
        pIP->cpu.pCodeStart = pIP->code;
        pIP->cpu.pCodeEnd = pIP->code + pIP->
           instructionCount;

        // run code
        while (eAPInterpreterState_run == pIP->state) {
                pFC = pIP->cpu.pIP;
                // check if we reached the end of the
                   code
                if (pFC > pIP->cpu.pCodeEnd) {
                        return 0;
                }
                // execute command
                pFC->pHALFkt (&(pIP->cpu), pFC->param);
                // check error flags
                if (pIP->cpu.EF) {
                        return -1;
                }
        }
        return 1;
}


// process the actual state
int APInterpreterProcessState(TAPInterpreter IP){
        TAPrealInterpreter * pIP = (TAPrealInterpreter
           *) IP;
        pIP->state = pIP->nextState;
        int rc = 0;

        switch (pIP->state) {
                case eAPInterpreterState_idle:
                        break;
                case eAPInterpreterState_loadProgramm:
                        break;
                case eAPInterpreterState_run:
                        rc = APInterpreterStateRun(IP);
                        if (rc >= 0) pIP->state =
```

```
                                eAPInterpreterState_idle;
                        break;
                case eAPInterpreterState_oneStep:
                        break;
                case eAPInterpreterState_halt:
                        break;
                default:
                        return -10;
        }
        return rc;
}


// set interpreter state
int APInterpreterSetState (TAPInterpreter IP, int
    msgEndian, int32_t state) {
        TAPrealInterpreter * pIP = (TAPrealInterpreter
            *) IP;
        if (msgEndian != pIP->sysEndian) {
                APendianConversation32Bit((uint32_t *)&
                    state);
        }
        pIP->nextState = (int) state;
        return 0;
}

// setup the interpreter for a new program
int APInterpreterInitNewProgramm (TAPInterpreter IP, int
     msgEndian, int32_t instructionsNumber, int32_t
    VariableNumber) {
        TAPrealInterpreter * pIP = (TAPrealInterpreter
            *) IP;
        int i;

        APInterpreterClean (IP);

        if (msgEndian != pIP->sysEndian) {
                APendianConversation32Bit((uint32_t *)&
                    instructionsNumber);
                APendianConversation32Bit((uint32_t *)&
                    VariableNumber);
        }

        pIP->code = (TAPInterpreterFuncCall *) malloc(
            sizeof(TAPInterpreterFuncCall)*
            instructionsNumber);
        pIP->instructionCount = instructionsNumber;

        pIP->variables = (TAPInterpreterVariable *)
            malloc(sizeof(TAPInterpreterVariable) * (
            VariableNumber));
        for (i = 0; i < VariableNumber;i++) {
                pIP->variables[i].pData = NULL;
                pIP->variables[i].pVI = NULL;
        }
```

```
        pIP->variableCount = VariableNumber;

        return 0;
}

// load a variable/~array to an index
int APInterpreterLoadVar (TAPInterpreter IP, int msgEndian, int32_t index, int32_t varTypeID, int32_t numberOfElements)
{
        TAPrealInterpreter * pIP = (TAPrealInterpreter *) IP;
        if (msgEndian != pIP->sysEndian) {
                APendianConversation32Bit((uint32_t *)&index);
                APendianConversation32Bit((uint32_t *)&varTypeID);
        }

        if ((index < 0) || (index > pIP->variableCount)) return -1;

        // set pointer to the runtime variable
        TAPInterpreterVariable * pRTV = pIP->variables + index;
        THAL_Variable const * pV = HALfindVar(varTypeID);
        if (!pV) return -2;

        pRTV->pData = pV->pFkt_create((unsigned int)numberOfElements);
        //if (!pRTV->pData) return -3;

        pRTV->pVI = pV;
        return 0;
}

// load a single Instruction to an index
int APInterpreterLoadInstr (TAPInterpreter IP,int msgEndian, int32_t index, int32_t * pRawInstr)
{
        TAPrealInterpreter * pIP = (TAPrealInterpreter *) IP;
        if (msgEndian != pIP->sysEndian) {
                APendianConversation32Bit((uint32_t *)&index);
        }
        if ((index < 0) || (index > pIP->instructionCount)) return -1;
        TAPInterpreterFuncCall * pIFC = pIP->code + index;
        memset (pIFC, 0, sizeof(TAPInterpreterFuncCall));

        // get function
```

```
        int32_t fid = *pRawInstr;
        if (msgEndian != pIP->sysEndian) {
                APendianConversation32Bit((uint32_t *)&fid);
        }
        THALFunction const * pF = HALfindFunction(fid);
        if (!pF) return -2;
        pIFC->pHALFkt = pF->pfktHAL;

        // convert parameters
        pRawInstr++; // set to the first parameter
        int i;
        THALFunctionParam const * pP = pF->paramList.pL;
        TuAPInterpreterFunctionParameter * pIFP = pIFC->param;
        for (i = 0; i < pF->paramList.number; i++) {
                if (APconvertRawParamData (msgEndian,pIP->sysEndian,pRawInstr,pP,pIFP,pIP->variables)) return -3;
                pP++;
                pRawInstr++;
                pIFP++;
        }
        return 0;
}

typedef struct SAPrealMsgSystem {
        TAPMsg *                     pOldRXMsg;           // pointer to the oldest received messages
        TAPMsg *                     pNewRXMsg;           // pointer to the newest received messages
        TAPMMU                       mmu;                 // the mmu
        int                          sysEndianness;       // the system endianness
        int                          messagecounter;      // a counter for checkin if a new message has been received


} TAPrealMsgSystem;


int SMinitial (
                void *               pVoidSM,             // pointer to the statemachine
                uint32_t *           pD,                  // pointer to the data
                int                  number               // the number of data elements
```

```
        );

int SMdata (
                void *                  pVoidSM,            // pointer to the statemachine
                uint32_t *              pD,                 // pointer to the data
                int                             number      // the number of data elements
        );

int SMmessageFinished (
                void *                  pVoidSM             // pointer to the statemachine
        );



// create AP message system
TAPMsgSystem APMScreate (
                TAPMMU                          mmu,        // the mmu
                int                     sysEndianness   // the system endianness
        ) {
        TAPrealMsgSystem * pMS = (TAPrealMsgSystem *) malloc (sizeof(TAPrealMsgSystem));
        if (!pMS) return NULL;
        pMS->mmu = mmu;
        pMS->sysEndianness = sysEndianness;
        pMS->pOldRXMsg = NULL;
        pMS->pNewRXMsg = NULL;
        pMS->messagecounter = 0;


        return pMS;
}

void APMSdelete (
        TAPMsgSystem ms
        ) {
        TAPrealMsgSystem * pMS = (TAPrealMsgSystem *) ms;
        free (pMS);
}


// frees a message from the message system
void APMSdeleteMsg (
        TAPMsgSystem    ms,
        TAPMsg *                pM
```

```
	) {
	TAPrealMsgSystem * pMS = (TAPrealMsgSystem *) ms;
	AP_MMU_free(pMS->mmu,pM->memory);
}

// get memory for a new message
TAPMsg * APMSgetNewMsg (
		TAPrealMsgSystem *	pMsgSys,
		int			dataElementsNumber,
		const TAPMsgDrv *	pDrv
		) {
	TAPMMUmemmory m = AP_MMU_get (pMsgSys->mmu, sizeof(TAPMsg)/sizeof(int) + dataElementsNumber*sizeof(uint32_t)/sizeof(int));
	if (!m) return NULL;

	int * pRD = (int *) AP_MMU_getData(m);
	TAPMsg * pM = (TAPMsg *) pRD;
	pM->memory = m;
	pM->extraData.pDrv = pDrv;
	pM->pH = (TAPMsgHeader *)((int *) pRD + sizeof(TAPMsg)/sizeof(int));
	pM->pData = (int *)pM->pH + sizeof(TAPMsgHeader)/sizeof(int);
	pM->pNext = NULL;
	return pM;
}

// insert a new message into the message queue
void APMSInsertMsg (
		TAPrealMsgSystem *	pMS,
		TAPMsg *			pM
	) {

	if (pMS->pNewRXMsg) {
		pMS->pNewRXMsg->pNext = pM;
	}
	pMS->pNewRXMsg = pM;
	if (!pMS->pOldRXMsg) {
		pMS->pOldRXMsg = pM;
	}
	pMS->messagecounter++;



}

// get oldest message
TAPMsg * APMSgetMsg (
		TAPMsgSystem		ms,	// the message system
		TAPMessageID		msgID,
```

```
                        // if 0 all messages are allowed
                TAPNodeID                       sender,         // if 0 all senders are allowed
                uint32_t                        mNumber         // if 0 all numbers are allowed
        ) {

        TAPrealMsgSystem * pMS = (TAPrealMsgSystem *) ms;

        TAPMsg * res = NULL;
        // search msg list

        TAPMsg * pM = pMS->pOldRXMsg;
        TAPMsg * pAntecessorM =  NULL;
        // flags
        int senderOK;
        int msgIDok;
        int numberOK;

        while (pM) {
                senderOK = 0;
                msgIDok = 0;
                numberOK = 0;

                if (!sender) {
                        senderOK = 1;
                } else {
                        if (*(pM->pH)[eAPMsgHeaderPosition_sender] == sender) senderOK = 1;
                }
                if (!msgID) {
                        msgIDok = 1;
                } else {
                        if (*(pM->pH)[eAPMsgHeaderPosition_msgTypeID] == msgID) msgIDok = 1;
                }
                if (!mNumber) {
                        numberOK = 1;
                } else {
                        if (*(pM->pH)[eAPMsgHeaderPosition_msgNumber] == mNumber) numberOK = 1;
                }

                if ((senderOK) && (msgIDok) && (numberOK)) {
                        res = pM;
                        goto exit;
                }
                // the id was wrong but the number was
```

```
                    ok >> so the action failed
                if (numberOK) goto fail;
                pAntecessorM = pM;
                pM = pM->pNext;
        }
fail:
        return NULL;
exit:
        if (pAntecessorM) {
                pAntecessorM->pNext = pM->pNext;
        } else {
                pMS->pOldRXMsg = pM->pNext;
        }
        if (pM == pMS->pNewRXMsg) {
                pMS->pNewRXMsg = NULL;
        }

        return res;
}

// wait till a new message has been received
void APMSwaitForNewMessage (TAPrealMsgSystem * pMS)
{
        volatile int mc;

        mc = pMS->messagecounter;

        while (mc == pMS->messagecounter) {

        }
}

// returns 0 if a new message is available
inline int APMSisMessageAvailble (TAPrealMsgSystem * pMS ) {
        return (!pMS->pOldRXMsg) ? 0 : -1;
}

// ========================================
// the receive state machine
// ========================================

// the receive state machine state function for receiving the msg header
int SMinitial (
                void *                  pVoidSM,        // pointer to the statemachine
                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 = 1; i < eAPMsgHeaderPosition_headerElementNumber;i++) {
                        APendianConversation32Bit(&pSM->header[i]);
                }
        }
        // 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;
        for (i = 0; i < eAPMsgHeaderPosition_headerElementNumber;i++) {
                *pSM->pD = pSM->header[i];
```

```
                pSM->pD++;
        }
        pSM->elementsLeft = pSM->header[eAPMsgHeaderPosition_length];
        // set up the data
        // 1. check if there is an data element
        if (!pSM->elementsLeft) {
                // no! now finish the message
                return SMmessageFinished(pVoidSM);
        }
        // 2. yes
        // 2.1 setup the sm for the data receiving
        pSM->state = SMdata;
        // 2.2 now check if we have to copy some data
        number -= copyAmount;
        if (number) {
                // set the data pointer
                pD += copyAmount;
                // and copy the data
                return SMdata (pVoidSM,pD,number);
        }
        return 0;
}


// the receive state machine state function for receiving the data
int SMdata (
                void *                  pVoidSM,                // pointer to the statemachine
                uint32_t *              pD,                     // pointer to the data
                int                             number          // the number of data elements
        ) {
        TAPReceiveStateMachine * pSM = (TAPReceiveStateMachine *) pVoidSM;
        int                     copyAmount = number;
        int                             i;
        // 1. transfer the data
        // do some clipping
        if (pSM->elementsLeft < copyAmount) copyAmount = pSM->elementsLeft;
        // copy
        for (i = 0; i < copyAmount;i++) {
                *pSM->pD = *pD;
                pSM->pD++;
                pD++;
        }
        // set statemachine work data
```

```
        pSM->elementsLeft -= copyAmount;
        // check if we have to change the statemachine
        if (pSM->elementsLeft) return 0;
        int res = SMmessageFinished (pVoidSM);
        if (res) return res;

        // check if there some bytes left to copy
        number -= copyAmount;
        if (number) {
                // set the data pointer
                pD += copyAmount;
                // and copy the data
                return pSM->state (pVoidSM,pD,number);
        }
        return 0;

}

// this function is called when all data have been received
int SMmessageFinished (
                void *                  pVoidSM         // pointer to the statemachine
        ) {
        TAPReceiveStateMachine *        pSM = (TAPReceiveStateMachine *) pVoidSM;
        TAPMsg *                        pM;
        // 1. reset SM
        // set the helper
        pSM->elementsLeft = eAPMsgHeaderPosition_headerElementNumber;
        pSM->pD = pSM->header;

        // data
        pM = pSM->pMsg; // save msg info for inserting
        pSM->pMsg = NULL;

        // right state function
        pSM->state = SMinitial;         // the state

        // 2. insert message at the message system
        APMSInsertMsg ((TAPrealMsgSystem *)pSM->pMS,pM);
        return 0;
}


// inits the state machine
void APInitReceiveStateMachine (
                TAPReceiveStateMachine *        pSM,    // pointer to the state machine
                TAPMsgSystem                    pMS,    // pointer to the message system
```

```
                const TAPMsgDrv  *                  pDrv    // the driver associated with the statemachine
        ) {
        pSM->state = SMinitial;
        pSM->pMS = pMS;
        pSM->pDrv = pDrv;
        // set the helper
        pSM->elementsLeft = eAPMsgHeaderPosition_headerElementNumber;
        pSM->pD = pSM->header;

        // data
        pSM->pMsg = NULL;
}

int APHandleMsg (
                TAP *               pAP,
                TAPMsg *            pM
        ) {

        TAPMessageID                                    msgID;
        const THALMsgProcessMessageAssociation * pMsgIDandFunctAsso;
        int                                                             i;

        // get message id
        msgID = (*(pM->pH))[eAPMsgHeaderPosition_msgTypeID];
        // search handler
        pMsgIDandFunctAsso = gHALMsgProcessRXHandlers.pL;
        for (i = 0; i < gHALMsgProcessRXHandlers.number; i++) {
                if (((TAPMessageID)pMsgIDandFunctAsso->msgID) == msgID) {
                        return pMsgIDandFunctAsso->pfktHandle(pAP,pM);
                }
                pMsgIDandFunctAsso++;
        }
        return -1;
}

void APMessageProcessingThread (TAP * pAP) {

        TAPrealMsgSystem *              pMS = (TAPrealMsgSystem *) pAP->MS;
        TAPMsg *                                pM;
        TAPNodeID                               recv;
        while (1) {
                // wait for a message
                APMSwaitForNewMessage(pMS);
```

```
                    // get the message
                    pM = APMSgetMsg (pMS,0,0,0);
                    // search the message handler
                    recv = (*(pM->pH))[eAPMsgHeaderPosition_receiver];
                    if ((recv == dAPNodeID_ALL) || (recv == pAP->nodeID)) {
                            if(APHandleMsg (pAP,pM)) goto exit;
                    }
                    // free memory
                    AP_MMU_free(pMS->mmu, pM->memory);
          }
exit:
          AP_MMU_free(pMS->mmu, pM->memory);
}
```

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:

```
// ===============================
// AP uuid = 4
// ===============================

// the global var for the Frame WAV modul
TStjFrameWAVmodule gFrameWAVModule;


// inits the AP
int APinit (
                        TAP *                    pAP,
                        TAPNodeID                nodeID,
                        const TAPMsgDrv *        pDrvList,
                        const int                driverNumber,
                        size_t                   messagePoolSize,
                        int                      sysEndian
```

```
                )
{
        pAP->nodeID = nodeID;
        pAP->pNodeList = NULL;
        pAP->pDrvList = pDrvList;
        pAP->sysEndian = sysEndian;
        pAP->driverNumber = driverNumber;
        pAP->msgSysMMU = AP_MMU_create(messagePoolSize);
        pAP->IP = APInterpreterCreate(pAP);
        pAP->MS = APMScreate (pAP->msgSysMMU,sysEndian);
        pAP->msgNumber = 0;
        pAP->APstate = eAPstate_idle;

        if(
                (!pAP->msgSysMMU)||
                (!pAP->IP)||
                (!pAP->MS)
                ) return -1;
        // setup the wav modul

        TStjFrameWAVOpenInfo WI[] = {
                        {1,"input.wav"
                            ,1,44100,1,1024,256},
                        {2,"output.wav"
                            ,0,44100,1,1024,256}
        };

        if (FrameWAVmoduleInit(2,WI,&gFrameWAVModule))
           return -2;


        // login the ap to the message system
        return TX_login(pAP);
}

// deletes the AP
void APdelete (TAP * pAP)
{
        APMSdelete (pAP->MS);
        APInterpreterDelete(pAP->IP);
        AP_MMU_delete(pAP->msgSysMMU);
        // closes the wav frame modul
        FrameWAVmoduleExit(&gFrameWAVModule);
}

// find a node at the list
TAPNode * APfindNode(TAP * pAP, TAPNodeID nodeID) {
        TAPNode * pN = pAP->pNodeList;
        while (pN) {
                if (pN->nodeID == nodeID) return pN;
                pN = pN->pNext;
        };
        return NULL;
}

```

```c
// adds a new node to the node list
int APaddNode(TAP * pAP, TAPNodeID newNodeID, const TAPMsgDrv * pDrv) {
	if (APfindNode(pAP,newNodeID)) return 1;
	TAPNode * pN = (TAPNode *) malloc(sizeof(TAPNode));
	if (!pN) return -1;
	pN->nodeID = newNodeID;
	pN->pDrv = pDrv;
	pN->pNext = pAP->pNodeList;
	pAP->pNodeList = pN;
	return 0;
}

// removes a node from the node list
void APremoveNode(TAP * pAP, TAPNodeID nodeID){
	TAPNode * pAntN = pAP->pNodeList; // antecessor node
	TAPNode * pActN = pAP->pNodeList; // actual node

	while (pActN) {
		// compare node id's
		if (pActN->nodeID == nodeID) {
			// unchain

			// check if we at the first position at the list
			if (pAP->pNodeList == pAntN) {
				// reset the pointer
				pAP->pNodeList = pActN->pNext;
			} else {
				// set the antecessor
				pAntN->pNext = pActN->pNext;
			}
			// free node
			free(pActN);
			// and abort
			return;
		}
		// the actual element becomes the precessor element
		pAntN = pActN;
		pActN = pActN->pNext;
	}
}

// get a new message number
unsigned int APgetNewMessageNumber (TAP *pAP) {
	pAP->msgNumber++;
	return pAP->msgNumber;
}

// find the driver associated with der nodeID
```

```
const TAPMsgDrv * APfindDrvBySenderID (TAP * pAP, TAPNodeID node) {
        TAPNode * pN = pAP->pNodeList;
        while (pN) {
                if (pN->nodeID == node) {
                        return pN->pDrv;
                }
                pN = pN->pNext;
        }
        return NULL;
}

// runs the AP
int APrun(TAP *pAP) {
        pAP->APstate = eAPstate_run;
}


typedef struct SAPrealMMUMemory {
        int *                           pData;                  // the data
        size_t                          count;                  // amount of data elements
        struct SAPrealMMUMemory *       pNext;                  // next element
        struct SAPrealMMUMemory *       pPrev;                  // previous element
} TAPrealMMUMemory;

//the mmu type
typedef struct SAPrealMMU {
        int *                                   memory;         // the memory bolck
        TAPrealMMUMemory *              pStart;                 // first element
        TAPrealMMUMemory *              pEnd;                   // second element
        TAPrealMMUMemory *              pUnusedList;            // list with the unused elements
        int *                           pUnusedData;            // pointer to the unused memory
        size_t                          elementsAvailable;      // amount of elements witch are available without using the garbage collector
        size_t                          totalAvailable;         // total amount of free bytes

} TAPrealMMU;

// ========================================
// memory entry functions
```

```
// ========================================

// a little macro for unchaining an element
#define DMemoryEntryUnchain(pM) \
        if (pM->pNext) pM->pNext->pPrev = pM->pPrev; \
        if (pM->pPrev) pM->pPrev->pNext = pM->pNext


//creates an memory entry
TAPrealMMUMemory * MemoryEntry_create () {
        TAPrealMMUMemory * pM = NULL;

        pM = (TAPrealMMUMemory *) malloc(sizeof(TAPrealMMUMemory));
        if (!pM) return NULL;
        pM->pData = NULL;
        pM->count = 0;
        pM->pNext = NULL;
        pM->pPrev = NULL;

        return pM;
}

//deletes an memory Entry
void MemoryEntry_delete (
                TAPrealMMUMemory * pM    // the memory to delete
                )
{
        // put the entry out of the chain
        DMemoryEntryUnchain(pM);
        // now we delete it
        free(pM);
}

// ========================================
// mmu helper
// ========================================

//alloc if needed a new memory entry
TAPrealMMUMemory * MMU_helper_createMemoryEntry (
                TAPrealMMU *    pMMU            // MMU structure to init
                )
{
        // check if we have to alloc a new memory entry
        if (!pMMU->pUnusedList) return MemoryEntry_create();
        // no there is some left at the list
        TAPrealMMUMemory * pM;
        // take the first one
        pM = pMMU->pUnusedList;
        // reset the list
        pMMU->pUnusedList = pM->pNext;
        // now unchain the element (for sure)
```

```
        DMemoryEntryUnchain(pM);
        // set the element pointers
        pM->pNext = NULL;
        pM->pPrev = NULL;
        return pM;
}

//the garbage collector
void MMU_helper_garbageCollector (
                TAPrealMMU *     pMMU            // MMU
                   structure to init
                )
{
        TAPrealMMUMemory * pM = pMMU->pStart;
        int * pD = pMMU->memory;
        while (pM) {
                // check if we have to move the data
                if (pD != pM->pData) {
                        // move the data
                        memmove(pD,pM->pData,pM->count*
                           sizeof(int));
                }
                // reset the destination pointer
                pD += pM->count;
                pM = pM->pNext;
        }
        // compressing memory finished
        // set the mmu vars new
        pMMU->elementsAvailable = pMMU->totalAvailable;
        pMMU->pUnusedData = pD;
}



// create a mmu
TAPMMU AP_MMU_create (size_t elementsNumber) {
        TAPrealMMU * pMMU;

        pMMU = (TAPrealMMU *) malloc (sizeof(TAPrealMMU)
           );
        if (!pMMU) return NULL;



        // setup lists
        pMMU->pStart = NULL;
        pMMU->pEnd = NULL;
        pMMU->pUnusedList = NULL;

        pMMU->elementsAvailable =elementsNumber;
        pMMU->pUnusedData = (int *) malloc (pMMU->
           elementsAvailable);
        pMMU->totalAvailable = pMMU->elementsAvailable;
        return pMMU;
}
```

```c

// destroying the mmu
void AP_MMU_delete (TAPMMU mmu) {
        TAPrealMMU * pMMU = (TAPrealMMU *) mmu;

        TAPrealMMUMemory * pM;
        TAPrealMMUMemory * pMnext;

// 1. delete al mmu entry's
        // 1.1 unused entry
        pM = pMMU->pUnusedList;
        while (pM) {
                pMnext = pM->pNext;
                MemoryEntry_delete(pM);
                pM = pMnext;
        }
        pMMU->pUnusedList = NULL;
        // 1.2 used blocks
        pM = pMMU->pStart;
        while (pM) {
                pMnext = pM->pNext;
                MemoryEntry_delete(pM);
                pM = pMnext;
        }
        pMMU->pStart = NULL;
        pMMU->pEnd = NULL;
// 2. delete mmu memory
        free (pMMU->memory);


}
// getting memmory from the mmu
TAPMMUmemmory AP_MMU_get (TAPMMU mmu, size_t elements) {
        TAPrealMMU * pMMU = (TAPrealMMU *) mmu;

        // check if there is enough space
        if (pMMU->totalAvailable < elements) goto error;
        // check if we have to use the garbage collector
        if (pMMU->elementsAvailable < elements) {
                // start garbage collector
                MMU_helper_garbageCollector(pMMU);
        }
        // we have enough memory so let's allocate some

        // get a new entry
        TAPrealMMUMemory * pM;
        pM = MMU_helper_createMemoryEntry(pMMU);
        if (!pM) return NULL;
        // get some memory
        pM->pData = pMMU->pUnusedData;
        pM->count = elements;
        // refresh data
        pMMU->pUnusedData += elements;
        pMMU->totalAvailable -= elements;
        pMMU->elementsAvailable -= elements;
```

```
        // insert memory element at the end of the list and update last element
        pM->pPrev = pMMU->pEnd;
        if (pMMU->pEnd) pMMU->pEnd->pNext = pM;
        if (!pMMU->pStart) pMMU->pStart = pM;
        pMMU->pEnd = pM;

        return pM;
error:

        return NULL;
}

// free memmory from the mmu
void AP_MMU_free (TAPMMU mmu, TAPMMUmemmory memory) {
        TAPrealMMU * pMMU = (TAPrealMMU *) mmu;
        TAPrealMMUMemory * pM = (TAPrealMMUMemory *) memory;

        if (!pM) return;
        // set mmu settings
        if (pMMU->pStart == pM) pMMU->pStart = pM->pNext;
        if (pMMU->pEnd == pM) pMMU->pEnd = pM->pPrev;
        // unchain element
        DMemoryEntryUnchain (pM);
        // and put it to the chain of unused
        pM->pNext = pMMU->pUnusedList;
        pM->pPrev = NULL;
        if (pMMU->pUnusedList) {
                pMMU->pUnusedList->pPrev = pM;
        }
        pMMU->pUnusedList = pM;
        // now set the mmu data new
        pMMU->totalAvailable += pM->count;

}

// getting access to the MMU data
void * AP_MMU_getData (TAPMMUmemmory memory) {
        TAPrealMMUMemory * pM = (TAPrealMMUMemory *) memory;
        return pM->pData;
}

// the real interpreter
typedef struct SAPrealInterpreter {
        int                     state;                  // state of the IP
        int                     nextState;              // the next state of the IP
        TAPInterpreterCPU                               cpu;    // the IP
```

```
core
        TAPInterpreterFuncCall *        code;                    // the code
        int32_t                 instructionCount;               // number of instructions at the code
        TAPInterpreterVariable *        variables;               // the variables
        int32_t                 variableCount;                  // number of the variables
        int                     sysEndian;                      // endian of the system
} TAPrealInterpreter;

// create a new interpreter
TAPInterpreter APInterpreterCreate (void * pAP) {
        TAPrealInterpreter * pIP = NULL;
        pIP = (TAPrealInterpreter *) malloc (sizeof(TAPrealInterpreter));
        if (!pIP) return NULL;

        pIP->state = eAPInterpreterState_idle;
        pIP->nextState = eAPInterpreterState_idle;
        pIP->sysEndian = ((TAP *)pAP)->sysEndian;
        pIP->cpu.CF = 0;
        pIP->cpu.EF = 0;
        pIP->cpu.pCodeStart = NULL;
        pIP->cpu.pCodeEnd = NULL;
        pIP->cpu.pIP = NULL;

        pIP->code = NULL;
        pIP->instructionCount = 0;

        pIP->variables = NULL;
        pIP->variableCount = 0;

        return pIP;
}

// cleans the interpreter
void APInterpreterClean (TAPInterpreter IP) {
        TAPrealInterpreter * pIP = (TAPrealInterpreter *) IP;

        // clean code
        if (pIP->code) {
                free (pIP->code);
                pIP->code = NULL;
        }
        pIP->instructionCount = 0;

        // clean variables
        TAPInterpreterVariable * pV = pIP->variables;
```

```
        int i;
        for (i = 0; i < pIP->variableCount; i++) {
                if (pV->pVI) pV->pVI->pFkt_delete(pV->
                    pData);
                pV++;
        }
        if (pIP->variables) {
                free (pIP->variables);
                pIP->variables = NULL;
        }
        pIP->variableCount = 0;

}

// deletes the interpreter
void APInterpreterDelete (TAPInterpreter IP) {
        TAPrealInterpreter * pIP = (TAPrealInterpreter
           *) IP;
        APInterpreterClean(IP);
        free (pIP);
}

int APInterpreterStateRun(TAPInterpreter IP) {
        TAPrealInterpreter * pIP = (TAPrealInterpreter
           *) IP;
        TAPInterpreterFuncCall * pFC;

        // setup cpu
        pIP->cpu.CF = 0;
        pIP->cpu.EF = 0;
        pIP->cpu.pIP = pIP->code;
        pIP->cpu.pCodeStart = pIP->code;
        pIP->cpu.pCodeEnd = pIP->code + pIP->
           instructionCount;

        // run code
        while (eAPInterpreterState_run == pIP->state) {
                pFC = pIP->cpu.pIP;
                // check if we reached the end of the
                   code
                if (pFC > pIP->cpu.pCodeEnd) {
                        return 0;
                }
                // execute command
                pFC->pHALFkt (&(pIP->cpu), pFC->param);
                // check error flags
                if (pIP->cpu.EF) {
                        return -1;
                }
        }
        return 1;
}


// process the actual state
```

```
int APInterpreterProcessState(TAPInterpreter IP){
	TAPrealInterpreter * pIP = (TAPrealInterpreter *) IP;
	pIP->state = pIP->nextState;
	int rc = 0;

	switch (pIP->state) {
		case eAPInterpreterState_idle:
			break;
		case eAPInterpreterState_loadProgramm:
			break;
		case eAPInterpreterState_run:
			rc = APInterpreterStateRun(IP);
			if (rc >= 0) pIP->state = eAPInterpreterState_idle;
			break;
		case eAPInterpreterState_oneStep:
			break;
		case eAPInterpreterState_halt:
			break;
		default:
			return -10;
	}
	return rc;
}


// set interpreter state
int APInterpreterSetState (TAPInterpreter IP, int msgEndian, int32_t state) {
	TAPrealInterpreter * pIP = (TAPrealInterpreter *) IP;
	if (msgEndian != pIP->sysEndian) {
		APendianConversation32Bit((uint32_t *)&state);
	}
	pIP->nextState = (int) state;
	return 0;
}

// setup the interpreter for a new program
int APInterpreterInitNewProgramm (TAPInterpreter IP, int msgEndian, int32_t instructionsNumber, int32_t VariableNumber) {
	TAPrealInterpreter * pIP = (TAPrealInterpreter *) IP;
	int i;

	APInterpreterClean (IP);

	if (msgEndian != pIP->sysEndian) {
		APendianConversation32Bit((uint32_t *)&instructionsNumber);
		APendianConversation32Bit((uint32_t *)&VariableNumber);
```

```c
        }

        pIP->code = (TAPInterpreterFuncCall *) malloc(sizeof(TAPInterpreterFuncCall)* instructionsNumber);
        pIP->instructionCount = instructionsNumber;

        pIP->variables = (TAPInterpreterVariable *) malloc(sizeof(TAPInterpreterVariable) * (VariableNumber));
        for (i = 0; i < VariableNumber;i++) {
                pIP->variables[i].pData = NULL;
                pIP->variables[i].pVI = NULL;
        }
        pIP->variableCount = VariableNumber;

        return 0;
}

// load a variable/~array to an index
int APInterpreterLoadVar (TAPInterpreter IP, int msgEndian, int32_t index, int32_t varTypeID, int32_t numberOfElements)
{
        TAPrealInterpreter * pIP = (TAPrealInterpreter *) IP;
        if (msgEndian != pIP->sysEndian) {
                APendianConversation32Bit((uint32_t *)&index);
                APendianConversation32Bit((uint32_t *)&varTypeID);
        }

        if ((index < 0) || (index > pIP->variableCount)) return -1;

        // set pointer to the runtime variable
        TAPInterpreterVariable * pRTV = pIP->variables + index;
        const THAL_Variable * pV = HALfindVar(varTypeID);
        if (!pV) return -2;

        pRTV->pData = pV->pFkt_create((unsigned int) numberOfElements);
        //if (!pRTV->pData) return -3;

        pRTV->pVI = pV;
        return 0;
}

// load a single Instruction to an index
int APInterpreterLoadInstr (TAPInterpreter IP,int msgEndian, int32_t index, int32_t * pRawInstr)
{
```

```
        TAPrealInterpreter * pIP = (TAPrealInterpreter
           *) IP;
        if (msgEndian != pIP->sysEndian) {
                APendianConversation32Bit((uint32_t *)&
                   index);
        }
        if ((index < 0) || (index > pIP->
           instructionCount)) return -1;
        TAPInterpreterFuncCall * pIFC = pIP->code +
           index;
        memset (pIFC, 0, sizeof(TAPInterpreterFuncCall))
           ;

        // get function
        int32_t fid = *pRawInstr;
        if (msgEndian != pIP->sysEndian) {
                APendianConversation32Bit((uint32_t *)&
                   fid);
        }
        const THALFunction * pF = HALfindFunction(fid);
        if (!pF) return -2;
        pIFC->pHALFkt = pF->pfktHAL;

        // convert parameters
        pRawInstr++; // set to the first parameter
        int i;
        const THALFunctionParam * pP = pF->paramList.pL;
        TuAPInterpreterFunctionParameter * pIFP = pIFC->
           param;
        for (i = 0; i < pF->paramList.number; i++) {
                if (APconvertRawParamData (msgEndian,pIP
                   ->sysEndian,pRawInstr,pP,pIFP,pIP->
                   variables)) return -3;
                pP++;
                pRawInstr++;
                pIFP++;
        }
        return 0;
}

typedef struct SAPrealMsgSystem {
        TAPMsg *                          pOldRXMsg;
                   // pointer to the oldest received
           messages
        TAPMsg *                          pNewRXMsg;
                   // pointer to the newest received
           messages
        TAPMMU                            mmu;
                   // the mmu
        int
           sysEndianness;  // the system endianness
        int
           messagecounter; // a counter for checkin if a
            new message has been received

```

```c

} TAPrealMsgSystem;


int SMinitial (
                void *                  pVoidSM,        // pointer to the statemachine
                uint32_t *              pD,             // pointer to the data
                int                     number          // the number of data elements
        );

int SMdata (
                void *                  pVoidSM,        // pointer to the statemachine
                uint32_t *              pD,             // pointer to the data
                int                     number          // the number of data elements
        );

int SMmessageFinished (
                void *                  pVoidSM         // pointer to the statemachine
        );



// create AP message system
TAPMsgSystem APMScreate (
                TAPMMU                  mmu,            // the mmu
                int                     sysEndianness   // the system endianness
        ) {
        TAPrealMsgSystem * pMS = (TAPrealMsgSystem *) malloc (sizeof(TAPrealMsgSystem));
        if (!pMS) return NULL;
        pMS->mmu = mmu;
        pMS->sysEndianness = sysEndianness;
        pMS->pOldRXMsg = NULL;
        pMS->pNewRXMsg = NULL;
        pMS->messagecounter = 0;


        return pMS;
}
```

```
void APMSdelete (
        TAPMsgSystem ms
        ) {
        TAPrealMsgSystem * pMS = (TAPrealMsgSystem *) ms
            ;
        free (pMS);
}


// frees a message from the message system
void APMSdeleteMsg (
        TAPMsgSystem    ms,
        TAPMsg *                pM
        ) {
        TAPrealMsgSystem * pMS = (TAPrealMsgSystem *) ms
            ;
        AP_MMU_free(pMS->mmu,pM->memory);
}

// get memory for a new message
TAPMsg * APMSgetNewMsg (
                TAPrealMsgSystem *      pMsgSys,
                int
                    dataElementsNumber,
                const TAPMsgDrv *       pDrv
                ) {
        TAPMMUmemmory m = AP_MMU_get (pMsgSys->mmu,
            sizeof(TAPMsg)/sizeof(int) +
            dataElementsNumber*sizeof(uint32_t)/sizeof(
            int));
        if (!m) return NULL;

        int * pRD = (int *) AP_MMU_getData(m);
        TAPMsg * pM = (TAPMsg *) pRD;
        pM->memory = m;
        pM->extraData.pDrv = pDrv;
        pM->pH = (TAPMsgHeader *)((int *) pRD + sizeof(
            TAPMsg)/sizeof(int));
        pM->pData = (int *)pM->pH + sizeof(TAPMsgHeader)
            /sizeof(int);
        pM->pNext = NULL;
        return pM;
}

// insert a new message into the message queue
void APMSInsertMsg (
                TAPrealMsgSystem *      pMS,
                TAPMsg *                        pM
        ) {

        if (pMS->pNewRXMsg) {
                pMS->pNewRXMsg->pNext = pM;
        }
        pMS->pNewRXMsg = pM;
```

```
	if (!pMS->pOldRXMsg) {
		pMS->pOldRXMsg = pM;
	}
	pMS->messagecounter++;



}

// get oldest message
TAPMsg * APMSgetMsg (
		TAPMsgSystem		ms,		// the message system
		TAPMessageID		msgID,	// if 0 all messages are allowed
		TAPNodeID			sender,		// if 0 all senders are allowed
		uint32_t			mNumber		// if 0 all numbers are allowed
	) {

	TAPrealMsgSystem * pMS = (TAPrealMsgSystem *) ms;

	TAPMsg * res = NULL;
	// search msg list

	TAPMsg * pM = pMS->pOldRXMsg;
	TAPMsg * pAntecessorM =  NULL;
	// flags
	int senderOK;
	int msgIDok;
	int numberOK;

	while (pM) {
		senderOK = 0;
		msgIDok = 0;
		numberOK = 0;

		if (!sender) {
			senderOK = 1;
		} else {
			if (*(pM->pH)[eAPMsgHeaderPosition_sender] == sender) senderOK = 1;
		}
		if (!msgID) {
			msgIDok = 1;
		} else {
			if (*(pM->pH)[eAPMsgHeaderPosition_msgTypeID] == msgID) msgIDok = 1;
		}
```

```
                if (!mNumber) {
                        numberOK = 1;
                } else {
                        if ((*(pM->pH)[eAPMsgHeaderPosition_msgNumber] == mNumber) numberOK = 1;
                }

                if ((senderOK) && (msgIDok) && (numberOK)) {
                        res = pM;
                        goto exit;
                }
                // the id was wrong but the number was ok >> so the action failed
                if (numberOK) goto fail;
                pAntecessorM = pM;
                pM = pM->pNext;
        }
fail:
        return NULL;
exit:
        if (pAntecessorM) {
                pAntecessorM->pNext = pM->pNext;
        } else {
                pMS->pOldRXMsg = pM->pNext;
        }
        if (pM == pMS->pNewRXMsg) {
                pMS->pNewRXMsg = NULL;
        }

        return res;
}

// wait till a new message has been received
void APMSwaitForNewMessage (TAPrealMsgSystem * pMS)
{
        volatile int mc;

        mc = pMS->messagecounter;

        while (mc == pMS->messagecounter) {

        }
}

// returns 0 if a new message is available
inline int APMSisMessageAvailible (TAPrealMsgSystem * pMS) {
        return (!pMS->pOldRXMsg) ? 0 : -1;
}

// ========================================
// the receive state machine
// ========================================
```

```
// the receive state machine state function for
   receiving the msg header
int SMinitial (
                void *                    pVoidSM,
                           // pointer to the
                   statemachine
                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 = 1; i <
                   eAPMsgHeaderPosition_headerElementNumber
                   ;i++) {
                        APendianConversation32Bit(&pSM->
                           header[i]);
                }
        }
        // 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;
        for (i = 0; i <
            eAPMsgHeaderPosition_headerElementNumber;i++)
             {
                *pSM->pD = pSM->header[i];
                pSM->pD++;
        }
        pSM->elementsLeft = pSM->header[
            eAPMsgHeaderPosition_length];
        // set up the data
        // 1. check if there is an data element
        if (!pSM->elementsLeft) {
                // no! now finish the message
                return SMmessageFinished(pVoidSM);
        }
        // 2. yes
        // 2.1 setup the sm for the data receiving
        pSM->state = SMdata;
        // 2.2 now check if we have to copy some data
        number -= copyAmount;
        if (number) {
                // set the data pointer
                pD += copyAmount;
                // and copy the data
                return SMdata (pVoidSM,pD,number);
        }
        return 0;
}


// the receive state machine state function for
    receiving the data
int SMdata (
                void *                  pVoidSM,
                            // pointer to the
                    statemachine
                uint32_t *              pD,
                                    // pointer to the
                    data
                int                             number
                                    // the number of data
                     elements
        ) {
        TAPReceiveStateMachine *
            pSM = (TAPReceiveStateMachine *) pVoidSM;
        int
```

```c
                                copyAmount = number;
        int
                                        i;
        // 1. transfer the data
        // do some clipping
        if (pSM->elementsLeft < copyAmount) copyAmount =
            pSM->elementsLeft;
        // copy
        for (i = 0; i < copyAmount;i++) {
                *pSM->pD = *pD;
                pSM->pD++;
                pD++;
        }
        // set statemachine work data
        pSM->elementsLeft -= copyAmount;
        // check if we have to change the statemachine
        if (pSM->elementsLeft) return 0;
        int res = SMmessageFinished (pVoidSM);
        if (res) return res;

        // check if there some bytes left to copy
        number -= copyAmount;
        if (number) {
                // set the data pointer
                pD += copyAmount;
                // and copy the data
                return pSM->state (pVoidSM,pD,number);
        }
        return 0;

}

// this function is called when all data have been
   received
int SMmessageFinished (
                void *                  pVoidSM
                   // pointer to the statemachine
        ) {
        TAPReceiveStateMachine *
           pSM = (TAPReceiveStateMachine *) pVoidSM;
        TAPMsg *
                                pM;
        // 1. reset SM
        // set the helper
        pSM->elementsLeft =
           eAPMsgHeaderPosition_headerElementNumber;
        pSM->pD = pSM->header;

        // data
        pM = pSM->pMsg; // save msg info for inserting
        pSM->pMsg = NULL;

        // right state function
        pSM->state = SMinitial;         // the state

```

```
        // 2. insert message at the message system
        APMSInsertMsg ((TAPrealMsgSystem *)pSM->pMS,pM);
        return 0;
}


// inits the state machine
void APInitReceiveStateMachine (
                TAPReceiveStateMachine *           pSM,     // pointer to the state machine
                TAPMsgSystem                       pMS,     // pointer to the message system
                const TAPMsgDrv  *                 pDrv     // the driver associated with the statemachine
        ) {
        pSM->state = SMinitial;
        pSM->pMS = pMS;
        pSM->pDrv = pDrv;
        // set the helper
        pSM->elementsLeft = eAPMsgHeaderPosition_headerElementNumber;
        pSM->pD = pSM->header;

        // data
        pSM->pMsg = NULL;
}

int APHandleMsg (
                TAP *           pAP,
                TAPMsg *        pM
        ) {

        TAPMessageID                                    msgID;
        const THALMsgProcessMessageAssociation * pMsgIDandFunctAsso;
        int                                                             i;

        // get message id
        msgID = (*(pM->pH))[eAPMsgHeaderPosition_msgTypeID];
        // search handler
        pMsgIDandFunctAsso = gHALMsgProcessRXHandlers.pL;
        for (i = 0; i < gHALMsgProcessRXHandlers.number; i++) {
                if (((TAPMessageID)pMsgIDandFunctAsso->msgID) == msgID) {
                        return pMsgIDandFunctAsso->pfktHandle(pAP,pM);
                }
```

```
                pMsgIDandFunctAsso++;
        }
        return -1;
}

void APMessageProcessingThread (TAP * pAP) {

        TAPrealMsgSystem *                pMS = (
           TAPrealMsgSystem *) pAP->MS;
        TAPMsg *                                  pM;
        TAPNodeID                                 recv;
        while (1) {
                // wait for a message
                APMSwaitForNewMessage(pMS);
                // get the message
                pM = APMSgetMsg (pMS,0,0,0);
                // search the message handler
                recv = (*(pM->pH))[
                   eAPMsgHeaderPosition_receiver];
                if ((recv == dAPNodeID_ALL) || (recv ==
                   pAP->nodeID)) {
                        if(APHandleMsg (pAP,pM)) goto
                           exit;
                }
                // free memory
                AP_MMU_free(pMS->mmu, pM->memory);
        }
exit:
        AP_MMU_free(pMS->mmu, pM->memory);
}
```

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:

```
// ===============================
// AP uuid = 7
// ===============================
// inits the AP
int APinit (
                        TAP *
                           pAP,
```

```
                                TAPNodeID
                                    nodeID,
                                const TAPMsgDrv *         pDrvList
                                    ,
                                const int
                                    driverNumber,
                                size_t
                                    messagePoolSize,
                                int
                                               sysEndian
                     )
{
          gAPendianFlag = sysEndian;

          pAP->nodeID = nodeID;
          pAP->pNodeList = NULL;
          pAP->pDrvList = pDrvList;
          pAP->driverNumber = driverNumber;
          pAP->msgSysMMU = AP_MMU_create(messagePoolSize);
          pAP->IP = APInterpreterCreate(pAP);
          pAP->MS = APMScreate (pAP->msgSysMMU,sysEndian);
          pAP->msgNumber = 0;
          pAP->APstate = eAPstate_idle;

          if(
                    (!pAP->msgSysMMU)||
                    (!pAP->IP)||
                    (!pAP->MS)
                    ) return -1;

          initHW(drv_2_cbAPClient);

          startHW();

          // init drv
          TAPMsgDrv * pDrv = (TAPMsgDrv *)pDrvList;
          int i;
          for (i = 0; i < driverNumber; i++) {
                    pDrv->pfkt_open(pAP, pDrv);
                    pDrv++;
          }

          // login the ap to the message system
          return TX_login(pAP);
}

// deletes the AP
void APdelete (TAP * pAP)
{
          TX_logout(pAP);
          // close & destroy drv
          TAPMsgDrv * pDrv = (TAPMsgDrv *)pAP->pDrvList;
          int i;
          for (i = 0; i < pAP->driverNumber; i++) {
                    pDrv->pfkt_close(pDrv);
```

```
                pDrv->pfkt_destroy(pDrv);
                pDrv++;
        }

        APMSdelete (pAP->MS);
        APInterpreterDelete(pAP->IP);
        AP_MMU_delete(pAP->msgSysMMU);
}

// find a node at the list
TAPNode * APfindNode(TAP * pAP, TAPNodeID nodeID) {
        TAPNode * pN = pAP->pNodeList;
        while (pN) {
                if (pN->nodeID == nodeID) return pN;
                pN = pN->pNext;
        };
        return NULL;
}

// adds a new node to the node list
int APaddNode(TAP * pAP, TAPNodeID newNodeID, const TAPMsgDrv * pDrv) {
        if (APfindNode(pAP,newNodeID)) return 1;
        TAPNode * pN = (TAPNode *) malloc(sizeof(TAPNode));
        if (!pN) return -1;
        pN->nodeID = newNodeID;
        pN->pDrv = pDrv;
        pN->pNext = pAP->pNodeList;
        pAP->pNodeList = pN;
        return 0;
}

// removes a node from the node list
void APremoveNode(TAP * pAP, TAPNodeID nodeID){
        TAPNode * pAntN = pAP->pNodeList; // antecessor node
        TAPNode * pActN = pAP->pNodeList; // actual node

        while (pActN) {
                // compare node id's
                if (pActN->nodeID == nodeID) {
                        // unchain

                        // check if we at the first position at the list
                        if (pAP->pNodeList == pAntN) {
                                // reset the pointer
                                pAP->pNodeList = pActN->pNext;
                        } else {
                                // set the antecessor
                                pAntN->pNext = pActN->pNext;
                        }
```

```
                        // free node
                        free(pActN);
                        // and abort
                        return;
                }
                // the actual element becomes the precessor element
                pAntN = pActN;
                pActN = pActN->pNext;
        }
}

// get a new message number
unsigned int APgetNewMessageNumber (TAP *pAP) {
        pAP->msgNumber++;
        return pAP->msgNumber;
}

// find the driver associated with der nodeID
const TAPMsgDrv * APfindDrvBySenderID (TAP * pAP, TAPNodeID node) {
        TAPNode * pN = pAP->pNodeList;
        while (pN) {
                if (pN->nodeID == node) {
                        return pN->pDrv;
                }
                pN = pN->pNext;
        }
        return NULL;
}

// runs the AP
int APrun(TAP *pAP) {
        pAP->APstate = eAPstate_run;
        return 0;
}


typedef struct SAPrealMMUMemory {
        int *                   pData;          // the data
        size_t                  count;          // amount of data elements
        struct SAPrealMMUMemory *       pNext;          // next element
        struct SAPrealMMUMemory *       pPrev;          // previous element
} TAPrealMMUMemory;

//the mmu type
typedef struct SAPrealMMU {
        int *                                   memory;         // the memory bolck
        TAPrealMMUMemory *              pStart;
```

```c
                                // first element
        TAPrealMMUMemory *                     pEnd;                           // second element
        TAPrealMMUMemory *                     pUnusedList;                    // list with the unused elements
        int *                                  pUnusedData;                    // pointer to the unused memory
        size_t                                 elementsAvailable;              // amount of elements witch are available without using the garbage collector
        size_t                                 totalAvailable;                 // total amount of free bytes

} TAPrealMMU;

// ========================================
// memory entry functions
// ========================================

// a little macro for unchaining an element
#define DMemoryEntryUnchain(pM) \
        if (pM->pNext) pM->pNext->pPrev = pM->pPrev; \
        if (pM->pPrev) pM->pPrev->pNext = pM->pNext


//creates an memory entry
TAPrealMMUMemory * MemoryEntry_create () {
        TAPrealMMUMemory * pM = NULL;

        pM = (TAPrealMMUMemory *) malloc(sizeof(TAPrealMMUMemory));
        if (!pM) return NULL;
        pM->pData = NULL;
        pM->count = 0;
        pM->pNext = NULL;
        pM->pPrev = NULL;

        return pM;
}

//deletes an memory Entry
void MemoryEntry_delete (
                TAPrealMMUMemory * pM   // the memory to delete
                )
{
        // put the entry out of the chain
        DMemoryEntryUnchain(pM);
        // now we delete it
        free(pM);
}

```

```
// =========================================
// mmu helper
// =========================================

//alloc if needed a new memory entry
TAPrealMMUMemory * MMU_helper_createMemoryEntry (
                TAPrealMMU *     pMMU            // MMU structure to init
                )
{
        // check if we have to alloc a new memory entry
        if (!pMMU->pUnusedList) return MemoryEntry_create();
        // no there is some left at the list
        TAPrealMMUMemory * pM;
        // take the first one
        pM = pMMU->pUnusedList;
        // reset the list
        pMMU->pUnusedList = pM->pNext;
        // now unchain the element (for sure)
        DMemoryEntryUnchain(pM);
        // set the element pointers
        pM->pNext = NULL;
        pM->pPrev = NULL;
        return pM;
}

//the garbage collector
void MMU_helper_garbageCollector (
                TAPrealMMU *     pMMU            // MMU structure to init
                )
{
        TAPrealMMUMemory * pM = pMMU->pStart;
        int * pD = pMMU->memory;
        while (pM) {
                // check if we have to move the data
                if (pD != pM->pData) {
                        // move the data
                        memmove(pD,pM->pData,pM->count*sizeof(int));
                }
                // reset the destination pointer
                pD += pM->count;
                pM = pM->pNext;
        }
        // compressing memory finished
        // set the mmu vars new
        pMMU->elementsAvailable = pMMU->totalAvailable;
        pMMU->pUnusedData = pD;
}



// create a mmu
```

```
TAPMMU AP_MMU_create (size_t elementsNumber) {
        TAPrealMMU * pMMU;

        pMMU = (TAPrealMMU *) malloc (sizeof(TAPrealMMU)
            );
        if (!pMMU) return NULL;



        // setup lists
        pMMU->pStart = NULL;
        pMMU->pEnd = NULL;
        pMMU->pUnusedList = NULL;

        pMMU->elementsAvailable =elementsNumber;
        pMMU->pUnusedData = (int *) malloc (pMMU->
            elementsAvailable);
        pMMU->totalAvailable = pMMU->elementsAvailable;
        return pMMU;
}

// destroying the mmu
void AP_MMU_delete (TAPMMU mmu) {
        TAPrealMMU * pMMU = (TAPrealMMU *) mmu;

        TAPrealMMUMemory * pM;
        TAPrealMMUMemory * pMnext;

// 1. delete al mmu entry's
        // 1.1 unused entry
        pM = pMMU->pUnusedList;
        while (pM) {
                pMnext = pM->pNext;
                MemoryEntry_delete(pM);
                pM = pMnext;
        }
        pMMU->pUnusedList = NULL;
        // 1.2 used blocks
        pM = pMMU->pStart;
        while (pM) {
                pMnext = pM->pNext;
                MemoryEntry_delete(pM);
                pM = pMnext;
        }
        pMMU->pStart = NULL;
        pMMU->pEnd = NULL;
// 2. delete mmu memory
        free (pMMU->memory);


}
// getting memmory from the mmu
TAPMMUmemmory AP_MMU_get (TAPMMU mmu, size_t elements) {
        TAPrealMMU * pMMU = (TAPrealMMU *) mmu;

```

```
	// check if there is enough space
	if (pMMU->totalAvailable < elements) goto error;
	// check if we have to use the garbage collector
	if (pMMU->elementsAvailable < elements) {
		// start garbage collector
		MMU_helper_garbageCollector(pMMU);
	}
	// we have enough memory so let's allocate some

	// get a new entry
	TAPrealMMUMemory * pM;
	pM = MMU_helper_createMemoryEntry(pMMU);
	if (!pM) return NULL;
	// get some memory
	pM->pData = pMMU->pUnusedData;
	pM->count = elements;
	// refresh data
	pMMU->pUnusedData += elements;
	pMMU->totalAvailable -= elements;
	pMMU->elementsAvailable -= elements;
	// insert memory element at the end of the list and update last element
	pM->pPrev = pMMU->pEnd;
	if (pMMU->pEnd) pMMU->pEnd->pNext = pM;
	if (!pMMU->pStart) pMMU->pStart = pM;
	pMMU->pEnd = pM;

	return pM;
error:

	return NULL;
}

// free memmory from the mmu
void AP_MMU_free (TAPMMU mmu, TAPMMUmemmory memory) {
	TAPrealMMU * pMMU = (TAPrealMMU *) mmu;
	TAPrealMMUMemory * pM = (TAPrealMMUMemory *) memory;

	if (!pM) return;
	// set mmu settings
	if (pMMU->pStart == pM) pMMU->pStart = pM->pNext;
	if (pMMU->pEnd == pM) pMMU->pEnd = pM->pPrev;
	// unchain element
	DMemoryEntryUnchain (pM);
	// and put it to the chain of unused
	pM->pNext = pMMU->pUnusedList;
	pM->pPrev = NULL;
	if (pMMU->pUnusedList) {
		pMMU->pUnusedList->pPrev = pM;
	}
	pMMU->pUnusedList = pM;
	// now set the mmu data new
	pMMU->totalAvailable += pM->count;
```

```
}

// getting access to the MMU data
void * AP_MMU_getData (TAPMMUmemmory memory) {
        TAPrealMMUMemory * pM = (TAPrealMMUMemory *)
           memory;
        return pM->pData;
}

// the real interpreter
typedef struct SAPrealInterpreter {
        TAP *
           pAP;                                         //
           pointer to the audio processor
        int
                   state;
                   // state of the IP
        int
                   nextState;
                   // the next state of the IP
        TAPInterpreterCPU                        cpu;
                                           // the IP
           core
        TAPInterpreterFuncCall *        code;
                                   // the code
        int32_t
           instructionCount;                  // number of
           instructions at the code
        TAPInterpreterVariable *        variables;
                                   // the variables
        int32_t
           variableCount;                     // number of
           the variables
} TAPrealInterpreter;

// create a new interpreter
TAPInterpreter APInterpreterCreate (void * pAP) {
        TAPrealInterpreter * pIP = NULL;
        pIP = (TAPrealInterpreter *) malloc (sizeof(
           TAPrealInterpreter));
        if (!pIP) return NULL;

        pIP->pAP = pAP;
        pIP->state = eAPInterpreterState_idle;
        pIP->nextState = eAPInterpreterState_idle;
        pIP->cpu.IP = pIP;
        pIP->cpu.CF = 0;
        pIP->cpu.EF = 0;
        pIP->cpu.pCodeStart = NULL;
        pIP->cpu.pCodeEnd = NULL;
        pIP->cpu.pIP = NULL;

        pIP->code = NULL;
        pIP->instructionCount = 0;
```

```c

        pIP->variables = NULL;
        pIP->variableCount = 0;

        return pIP;
}

// cleans the interpreter
void APInterpreterClean (TAPInterpreter IP) {
        TAPrealInterpreter * pIP = (TAPrealInterpreter
            *) IP;

        // clean code
        if (pIP->code) {
                free (pIP->code);
                pIP->code = NULL;
        }
        pIP->instructionCount = 0;

        // clean variables
        TAPInterpreterVariable * pV = pIP->variables;
        int i;
        for (i = 0; i < pIP->variableCount; i++) {
                if (pV->pVI) pV->pVI->pFkt_delete(pV->
                    pData);
                pV++;
        }
        if (pIP->variables) {
                free (pIP->variables);
                pIP->variables = NULL;
        }
        pIP->variableCount = 0;

}

// deletes the interpreter
void APInterpreterDelete (TAPInterpreter IP) {
        TAPrealInterpreter * pIP = (TAPrealInterpreter
            *) IP;
        APInterpreterClean(IP);
        free (pIP);
}

int APInterpreterStateRun(TAPInterpreter IP) {
        TAPrealInterpreter * pIP = (TAPrealInterpreter
            *) IP;
        TAPInterpreterFuncCall * pFC;
        int cc; // cycle counter

        // setup cpu
        pIP->cpu.CF = 0;
        pIP->cpu.EF = 0;
        pIP->cpu.pIP = pIP->code;
        pIP->cpu.pCodeStart = pIP->code;
        pIP->cpu.pCodeEnd = pIP->code + pIP->
```

```
            instructionCount;

        // run code
        cc = 10;
        while (eAPInterpreterState_run == pIP->state) {
                pFC = pIP->cpu.pIP;
                // check if we reached the end of the
                   code
                if (pFC > pIP->cpu.pCodeEnd) {
                        return 0;
                }
                // execute command
                pFC->pHALFkt (&(pIP->cpu), pFC->param);
                // check error flags
                if (pIP->cpu.EF) {
                        return -1;
                }
                // check message system after x cycles
                if (!cc) {
                        APMessageProcess(pIP->pAP);
                        cc = 100;
                } else {
                        cc--;
                }

        }
        return 1;
}


// process the actual state
int APInterpreterProcessState(TAPInterpreter IP){
        TAPrealInterpreter * pIP = (TAPrealInterpreter
           *) IP;
        pIP->state = pIP->nextState;
        int rc = 0;

        switch (pIP->state) {
                case eAPInterpreterState_idle:
                        break;
                case eAPInterpreterState_loadProgramm:
                        break;
                case eAPInterpreterState_run:
                        rc = APInterpreterStateRun(IP);
                        if (rc >= 0) pIP->state =
                           eAPInterpreterState_idle;
                        break;
                case eAPInterpreterState_oneStep:
                        break;
                case eAPInterpreterState_halt:
                        break;
                default:
                        return -10;
        }
        return rc;
```

```
}


// set interpreter state
int APInterpreterSetState (TAPInterpreter IP, int
   msgEndian, int32_t state) {
        TAPrealInterpreter * pIP = (TAPrealInterpreter
           *) IP;
        if (msgEndian != gAPendianFlag) {
                APendianConversation32Bit((uint32_t *)&
                   state, msgEndian);
        }
        pIP->nextState = (int) state;
        return 0;
}

// setup the interpreter for a new program
int APInterpreterInitNewProgramm (TAPInterpreter IP, int
    msgEndian, int32_t instructionsNumber, int32_t
   VariableNumber) {
        TAPrealInterpreter * pIP = (TAPrealInterpreter
           *) IP;
        int i;

        APInterpreterClean (IP);

        if (msgEndian != gAPendianFlag) {
                APendianConversation32Bit((uint32_t *)&
                   instructionsNumber, msgEndian);
                APendianConversation32Bit((uint32_t *)&
                   VariableNumber, msgEndian);
        }

        pIP->code = (TAPInterpreterFuncCall *) malloc(
           sizeof(TAPInterpreterFuncCall)*
           instructionsNumber);
        pIP->instructionCount = instructionsNumber;

        pIP->variables = (TAPInterpreterVariable *)
           malloc(sizeof(TAPInterpreterVariable) * (
           VariableNumber));
        for (i = 0; i < VariableNumber;i++) {
                pIP->variables[i].pData = NULL;
                pIP->variables[i].pVI = NULL;
        }
        pIP->variableCount = VariableNumber;

        return 0;
}

// load a variable/~array to an index
int APInterpreterLoadVar (TAPInterpreter IP, int
   msgEndian, int32_t index, int32_t varTypeID, int32_t
   numberOfElements)
{
```

```
        TAPrealInterpreter * pIP = (TAPrealInterpreter *) IP;
        if (msgEndian != gAPendianFlag) {
                APendianConversation32Bit((uint32_t *)&index, msgEndian);
                APendianConversation32Bit((uint32_t *)&varTypeID, msgEndian);
                APendianConversation32Bit((uint32_t *)&numberOfElements, msgEndian);
        }

        if ((index < 0) || (index > pIP->variableCount)) return -1;

        // set pointer to the runtime variable
        TAPInterpreterVariable * pRTV = pIP->variables + index;
        THAL_Variable const * pV = HALfindVar(varTypeID);
        if (!pV) return -2;

        pRTV->pData = pV->pFkt_create((unsigned int)numberOfElements);
        //if (!pRTV->pData) return -3;

        pRTV->pVI = pV;
        return 0;
}

// load a single Instruction to an index
int APInterpreterLoadInstr (TAPInterpreter IP,int msgEndian, int32_t index, int32_t * pRawInstr)
{
        TAPrealInterpreter * pIP = (TAPrealInterpreter *) IP;
        if (msgEndian != gAPendianFlag) {
                APendianConversation32Bit((uint32_t *)&index, msgEndian);
        }
        if ((index < 0) || (index > pIP->instructionCount)) return -1;
        TAPInterpreterFuncCall * pIFC = pIP->code + index;
        memset (pIFC, 0, sizeof(TAPInterpreterFuncCall));

        // get function
        int32_t fid = *pRawInstr;
        if (msgEndian != gAPendianFlag) {
                APendianConversation32Bit((uint32_t *)&fid, msgEndian);
        }
        THALFunction const * pF = HALfindFunction(fid);
        if (!pF) return -2;
        pIFC->pHALFkt = pF->pfktHAL;
```

```
	// convert parameters
	pRawInstr++; // set to the first parameter
	int i;
	THALFunctionParam const * pP = pF->paramList.pL;
	TuAPInterpreterFunctionParameter * pIFP = pIFC->param;
	for (i = 0; i < pF->paramList.number; i++) {
		if (APconvertRawParamData (msgEndian, pRawInstr,pP,pIFP,pIP->variables)) return -3;
		pP++;
		pRawInstr++;
		pIFP++;
	}
	return 0;
}

// gets the varaible by it's index
TAPInterpreterVariable * APInterpreterGetVariableByIndex (TAPInterpreter IP, int index) {
	return &(((TAPrealInterpreter *) IP)->variables[index]);
}

// gets the AP from the IP
void * APInterpreterGetAPfromIP (TAPInterpreter IP) {
	return ((TAPrealInterpreter *) IP)->pAP;
}


typedef struct SAPrealMsgSystem {
	TAPMsg *				pOldRXMsg;		// pointer to the oldest received messages
	TAPMsg *				pNewRXMsg;		// pointer to the newest received messages
	TAPMMU					mmu;			// the mmu
	int	sysEndianness;	// the system endianness
	int	messagecounter; // a counter for checkin if a new message has been received


} TAPrealMsgSystem;


int SMinitial (
		void *				pVoidSM,		// pointer to the statemachine
		uint32_t *			pD,
```

```
                                        // pointer to the
                    data
                int                                 number
                                        // the number of data
                     elements
        );

int SMdata (
                void *                      pVoidSM,
                                // pointer to the
                   statemachine
                uint32_t *                  pD,
                                        // pointer to the
                   data
                int                                 number
                                        // the number of data
                     elements
        );

int SMmessageFinished (
                void *                      pVoidSM
                   // pointer to the statemachine
        );



// create AP message system
TAPMsgSystem APMScreate (
                TAPMMU                              mmu,
                                        // the mmu
                int
                   sysEndianness        // the system
                   endianness
        ) {
        TAPrealMsgSystem * pMS = (TAPrealMsgSystem *)
           malloc (sizeof(TAPrealMsgSystem));
        if (!pMS) return NULL;
        pMS->mmu = mmu;
        pMS->sysEndianness = sysEndianness;
        pMS->pOldRXMsg = NULL;
        pMS->pNewRXMsg = NULL;
        pMS->messagecounter = 0;


        return pMS;
}

void APMSdelete (
        TAPMsgSystem ms
        ) {
        TAPrealMsgSystem * pMS = (TAPrealMsgSystem *) ms
           ;
        free (pMS);
}

```

```c

// frees a message from the message system
void APMSdeleteMsg (
        TAPMsgSystem    ms,
        TAPMsg *                pM
        ) {
        TAPrealMsgSystem * pMS = (TAPrealMsgSystem *) ms;
        AP_MMU_free(pMS->mmu,pM->memory);
}

// get memory for a new message
TAPMsg * APMSgetNewMsg (
                TAPrealMsgSystem *      pMsgSys,
                int     dataElementsNumber,
                const TAPMsgDrv *       pDrv
                ) {
        TAPMMUmemmory m = AP_MMU_get (pMsgSys->mmu, sizeof(TAPMsg)/sizeof(int) + dataElementsNumber*sizeof(uint32_t)/sizeof(int));
        if (!m) return NULL;

        int * pRD = (int *) AP_MMU_getData(m);
        TAPMsg * pM = (TAPMsg *) pRD;
        pM->memory = m;
        pM->extraData.pDrv = pDrv;
        pM->pH = (TAPMsgHeader *)((int *) pRD + sizeof(TAPMsg)/sizeof(int));
        pM->pData = (int *)pM->pH + sizeof(TAPMsgHeader)/sizeof(int);
        pM->pNext = NULL;
        return pM;
}

// insert a new message into the message queue
void APMSInsertMsg (
                TAPrealMsgSystem *      pMS,
                TAPMsg *                        pM
        ) {

        if (pMS->pNewRXMsg) {
                pMS->pNewRXMsg->pNext = pM;
        }
        pMS->pNewRXMsg = pM;
        if (!pMS->pOldRXMsg) {
                pMS->pOldRXMsg = pM;
        }
        pMS->messagecounter++;

}

// unchains a received message
void APMSunchainMessage (
```

```
                TAPrealMsgSystem *      pMS,
                TAPMsg *                        pM,
                TAPMsg *                        pAntecessorM
) {
        if (pAntecessorM) {
                pAntecessorM->pNext = pM->pNext;
        } else {
                pMS->pOldRXMsg = pM->pNext;
        }
        if (pM == pMS->pNewRXMsg) {
                pMS->pNewRXMsg = NULL;
        }

        // now there is one message less left
        pMS->messagecounter--;
}

// get oldest message
TAPMsg * APMSgetMsg (
                TAPMsgSystem            ms,                     // the message system
                TAPMessageID            msgID,          // if 0 all messages are allowed
                TAPNodeID                               sender,                 // if 0 all senders are allowed
                uint32_t                                mNumber,                // if 0 all numbers are allowed
                int                                     ackMsgAllowed
        ) {

        TAPrealMsgSystem * pMS = (TAPrealMsgSystem *) ms;
        // flags
        int senderOK;
        int msgIDok;
        int numberOK;
        // temp vars
        TAPMsg * pM;
        TAPMsg * pAntecessorM;
        uint32_t * pH;

        // result var
        TAPMsg * res = NULL;

checkMessages:

        // search msg list
        pM = pMS->pOldRXMsg;
        pAntecessorM =  NULL;

```

```
        if (!pM) goto waitForMessage;


        senderOK = 0;
        msgIDok = 0;
        numberOK = 0;

        pH = *(pM->pH);

        if (!sender) {
                senderOK = 1;
        } else {
                if (pH[eAPMsgHeaderPosition_sender] ==
                   sender) senderOK = 1;
        }
        if (!msgID) {
                // filter ack/nack msg
                if (ackMsgAllowed) {
                        msgIDok = 1;
                } else {
                        if (
                                                (pH[
                                                   eAPMsgHeaderPosition_msgTypeI
                                                   ] !=
                                                   eAPMsgTypes_ACK
                                                   )  &&
                                                (pH[
                                                   eAPMsgHeaderPosition_msgTypeI
                                                   ] !=
                                                   eAPMsgTypes_NACK
                                                   )
                                ) {
                                msgIDok = 1;
                        }
                }
        } else {
                if (pH[eAPMsgHeaderPosition_msgTypeID]
                   == msgID) msgIDok = 1;
        }
        if (!mNumber) {
                numberOK = 1;
        } else {
                if (pH[eAPMsgHeaderPosition_msgNumber]
                   == mNumber) numberOK = 1;
        }
        if ((senderOK) && (msgIDok) && (numberOK)) {
                res = pM;
                goto exit;
        }
        pAntecessorM = pM;
        pM = pM->pNext;
        if (pM) goto checkMessages;
waitForMessage:
        goto checkMessages;
```

```
exit:
        // unchain message
        APMSunchainMessage(pMS,pM,pAntecessorM);
        // now one thread is less waiting for a message
        return res;
error:
        return NULL;
}


// wait till a new message has been received
void APMSwaitForNewMessage (TAPrealMsgSystem * pMS)
{
        volatile int mc;

        mc = pMS->messagecounter;

        while (mc == pMS->messagecounter) {

        }
}

// returns 0 if a new message is available
inline int APMSisMessageAvailble (TAPrealMsgSystem * pMS) {
        return (!pMS->pOldRXMsg) ? 0 : -1;
}

// ========================================
// the receive state machine
// ========================================

// the receive state machine state function for receiving the msg header
int SMinitial (
                void *                  pVoidSM,                // pointer to the statemachine
                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], 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;
		for (i = 0; i < eAPMsgHeaderPosition_headerElementNumber;i++) {
			*pSM->pD = pSM->header[i];
			pSM->pD++;
		}
		pSM->elementsLeft = pSM->header[eAPMsgHeaderPosition_length];
		// set up the data
		// 1. check if there is an data element
		if (!pSM->elementsLeft) {
			// no! now finish the message
			return SMmessageFinished(pVoidSM);
		}
```

```
		// 2. yes
		// 2.1 setup the sm for the data receiving
		pSM->state = SMdata;
		// 2.2 now check if we have to copy some data
		number -= copyAmount;
		if (number) {
			// set the data pointer
			pD += copyAmount;
			// and copy the data
			return SMdata (pVoidSM,pD,number);
		}
		return 0;
}


// the receive state machine state function for receiving the data
int SMdata (
			void *			pVoidSM,		// pointer to the statemachine
			uint32_t *		pD,			// pointer to the data
			int				number			// the number of data elements
		) {
		TAPReceiveStateMachine * pSM = (TAPReceiveStateMachine *) pVoidSM;
		int				copyAmount = number;
		int					i;
		// 1. transfer the data
		// do some clipping
		if (pSM->elementsLeft < copyAmount) copyAmount = pSM->elementsLeft;
		// copy
		for (i = 0; i < copyAmount;i++) {
			*pSM->pD = *pD;
			pSM->pD++;
			pD++;
		}
		// set statemachine work data
		pSM->elementsLeft -= copyAmount;
		// check if we have to change the statemachine
		if (pSM->elementsLeft) return 0;
		int res = SMmessageFinished (pVoidSM);
		if (res) return res;

		// check if there some bytes left to copy
		number -= copyAmount;
		if (number) {
			// set the data pointer
```

```
                pD += copyAmount;
                // and copy the data
                return pSM->state (pVoidSM,pD,number);
        }
        return 0;

}

// this function is called when all data have been
   received
int SMmessageFinished (
                void *                  pVoidSM
                   // pointer to the statemachine
        ) {
        TAPReceiveStateMachine *
           pSM = (TAPReceiveStateMachine *) pVoidSM;
        TAPMsg *
                           pM;
        // 1. reset SM
        // set the helper
        pSM->elementsLeft =
           eAPMsgHeaderPosition_headerElementNumber;
        pSM->pD = pSM->header;

        // data
        pM = pSM->pMsg; // save msg info for inserting
        pSM->pMsg = NULL;

        // right state function
        pSM->state = SMinitial;         // the state

        // 2. insert message at the message system
        APMSInsertMsg ((TAPrealMsgSystem *)pSM->pMS,pM);
        return 0;
}


// inits the state machine
void APInitReceiveStateMachine (
                TAPReceiveStateMachine *
                           pSM,    // pointer to the
                   state machine
                TAPMsgSystem
                                   pMS,    // pointer to
                    the message system
                const TAPMsgDrv   *
                                   pDrv    // the driver
                    associated with the statemachine
        ) {
        pSM->state = SMinitial;
        pSM->pMS = pMS;
        pSM->pDrv = pDrv;
        // set the helper
        pSM->elementsLeft =
           eAPMsgHeaderPosition_headerElementNumber;
```

```
        pSM->pD = pSM->header;

        // data
        pSM->pMsg = NULL;
}

int APHandleMsg (
                TAP *           pAP,
                TAPMsg *        pM
        ) {

        TAPMessageID
                                        msgID;
        const THALMsgProcessMessageAssociation *
           pMsgIDandFunctAsso;
        int
                                                        i;

        // get message id
        msgID = (*(pM->pH))[
           eAPMsgHeaderPosition_msgTypeID];
        // search handler
        pMsgIDandFunctAsso = gHALMsgProcessRXHandlers.pL
           ;
        for (i = 0; i < gHALMsgProcessRXHandlers.number;
           i++) {
                if (((TAPMessageID)pMsgIDandFunctAsso->
                   msgID) == msgID) {
                        return pMsgIDandFunctAsso->
                           pfktHandle(pAP,pM);
                }
                pMsgIDandFunctAsso++;
        }
        return -1;
}

void APMessageProcessingThread (TAP * pAP) {
        TAPrealMsgSystem *              pMS = (
           TAPrealMsgSystem *) pAP->MS;
        TAPMsg *                                pM;
        TAPNodeID                               recv;
        while (1) {
                // get the message
                pM = APMSgetMsg (pMS,0,0,0,0);
                if (!pM) goto error;
                // search the message handler
                recv = (*(pM->pH))[
                   eAPMsgHeaderPosition_receiver];
                if ((recv == dAPNodeID_ALL) || (recv ==
                   pAP->nodeID)) {
                        if(APHandleMsg (pAP,pM)) goto
                           exit;
                }
                // free memory
                APMSdeleteMsg (pAP->MS,pM);
```

```
        }
exit:
        // free memory
        APMSdeleteMsg (pAP->MS,pM);
error:
        return;
}


// if a message is in the queue available it will be
   processed
void APMessageProcess (TAP * pAP) {
        TAPrealMsgSystem *              pMS = (
           TAPrealMsgSystem *) pAP->MS;
        TAPMsg *                                pM;
        TAPNodeID                               recvID;

        // search msg list
        pM = pMS->pOldRXMsg;

        // if there is no message we will return
        if (!pM) return;

        // unchain first message
        APMSunchainMessage(pMS, pM, NULL);

        // check header
        recvID = (*(pM->pH))[
           eAPMsgHeaderPosition_receiver];
        if ((recvID == dAPNodeID_ALL) || (recvID == pAP
           ->nodeID)) {
                APHandleMsg (pAP,pM);
        }
        // free memory
        APMSdeleteMsg (pAP->MS,pM);
}
```

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:

```
// ==============================
// AP uuid = 8
// ==============================
```

```
// the global var for the Frame WAV modul
TStjFrameWAVmodule gFrameWAVModule;

// inits the AP
int APinit (
                        TAP *           pAP,
                        TAPNodeID       nodeID,
                        const TAPMsgDrv *       pDrvList,
                        const int       driverNumber,
                        size_t      messagePoolSize,
                        int                     sysEndian
                )
{
        pAP->nodeID = nodeID;
        pAP->pNodeList = NULL;
        pAP->pDrvList = pDrvList;
        pAP->sysEndian = sysEndian;
        pAP->driverNumber = driverNumber;
        pAP->msgSysMMU = AP_MMU_create(messagePoolSize);
        pAP->IP = APInterpreterCreate(pAP);
        pAP->MS = APMScreate (pAP->msgSysMMU,sysEndian);
        pAP->msgNumber = 0;
        pAP->APstate = eAPstate_idle;

        if(
                (!pAP->msgSysMMU)||
                (!pAP->IP)||
                (!pAP->MS)
                ) return -1;

        TStjFrameWAVOpenInfo WI[] = {
                        {1,"x.wav",1,48000,1},
                        {2,"y.wav",0,48000,1}
        };

        if (FrameWAVmoduleInit(2,WI,&gFrameWAVModule)) return -2;


        // login the ap to the message system
        return TX_login(pAP);
}

// deletes the AP
void APdelete (TAP * pAP)
{
        APMSdelete (pAP->MS);
        APInterpreterDelete(pAP->IP);
```

```c
        AP_MMU_delete(pAP->msgSysMMU);
        // closes the wav frame modul
        FrameWAVmoduleExit(&gFrameWAVModule);
}

// find a node at the list
TAPNode * APfindNode(TAP * pAP, TAPNodeID nodeID) {
        TAPNode * pN = pAP->pNodeList;
        while (pN) {
                if (pN->nodeID == nodeID) return pN;
                pN = pN->pNext;
        };
        return NULL;
}

// adds a new node to the node list
int APaddNode(TAP * pAP, TAPNodeID newNodeID, const TAPMsgDrv * pDrv) {
        if (APfindNode(pAP,newNodeID)) return 1;
        TAPNode * pN = (TAPNode *) malloc(sizeof(TAPNode));
        if (!pN) return -1;
        pN->nodeID = newNodeID;
        pN->pDrv = pDrv;
        pN->pNext = pAP->pNodeList;
        pAP->pNodeList = pN;
        return 0;
}

// removes a node from the node list
void APremoveNode(TAP * pAP, TAPNodeID nodeID){
        TAPNode * pAntN = pAP->pNodeList; // antecessor node
        TAPNode * pActN = pAP->pNodeList; // actual node

        while (pActN) {
                // compare node id's
                if (pActN->nodeID == nodeID) {
                        // unchain

                        // check if we at the first position at the list
                        if (pAP->pNodeList == pAntN) {
                                // reset the pointer
                                pAP->pNodeList = pActN->pNext;
                        } else {
                                // set the antecessor
                                pAntN->pNext = pActN->pNext;
                        }
                        // free node
                        free(pActN);
                        // and abort
                        return;
```

```
                }
                // the actual element becomes the precessor element
                pAntN = pActN;
                pActN = pActN->pNext;
        }
}

// get a new message number
unsigned int APgetNewMessageNumber (TAP *pAP) {
        pAP->msgNumber++;
        return pAP->msgNumber;
}

// find the driver associated with der nodeID
const TAPMsgDrv * APfindDrvBySenderID (TAP * pAP, TAPNodeID node) {
        TAPNode * pN = pAP->pNodeList;
        while (pN) {
                if (pN->nodeID == node) {
                        return pN->pDrv;
                }
                pN = pN->pNext;
        }
        return NULL;
}

// runs the AP
int APrun(TAP *pAP) {
        pAP->APstate = eAPstate_run;
        return 0;
}


typedef struct SAPrealMMUMemory {
        int * pData;                    // the data
        size_t count;                   // amount of data elements
        struct SAPrealMMUMemory *       pNext;          // next element
        struct SAPrealMMUMemory *       pPrev;          // previous element
} TAPrealMMUMemory;

//the mmu type
typedef struct SAPrealMMU {
        int *                                   memory;         // the memory bolck
        TAPrealMMUMemory *              pStart;         // first element
        TAPrealMMUMemory *              pEnd;           // second element
        TAPrealMMUMemory *              pUnusedList;
```

```
                    // list with the unused elements
        int *
           pUnusedData;            // pointer to the
           unused memory
        size_t
           elementsAvailable;      // amount of elements
            witch are available without using the
           garbage collector
        size_t
           totalAvailable;         // total amount of
           free bytes

} TAPrealMMU;

// ========================================
// memory entry functions
// ========================================

// a little macro for unchaining an element
#define DMemoryEntryUnchain(pM) \
        if (pM->pNext) pM->pNext->pPrev = pM->pPrev; \
        if (pM->pPrev) pM->pPrev->pNext = pM->pNext


//creates an memory entry
TAPrealMMUMemory * MemoryEntry_create () {
        TAPrealMMUMemory * pM = NULL;

        pM = (TAPrealMMUMemory *) malloc(sizeof(
           TAPrealMMUMemory));
        if (!pM) return NULL;
        pM->pData = NULL;
        pM->count = 0;
        pM->pNext = NULL;
        pM->pPrev = NULL;

        return pM;
}

//deletes an memory Entry
void MemoryEntry_delete (
                TAPrealMMUMemory * pM   // the memory to
                    delete
                )
{
        // put the entry out of the chain
        DMemoryEntryUnchain(pM);
        // now we delete it
        free(pM);
}

// ========================================
// mmu helper
// ========================================

```

```
//alloc if needed a new memory entry
TAPrealMMUMemory * MMU_helper_createMemoryEntry (
                TAPrealMMU *    pMMU            // MMU structure to init
                )
{
        // check if we have to alloc a new memory entry
        if (!pMMU->pUnusedList) return MemoryEntry_create();
        // no there is some left at the list
        TAPrealMMUMemory * pM;
        // take the first one
        pM = pMMU->pUnusedList;
        // reset the list
        pMMU->pUnusedList = pM->pNext;
        // now unchain the element (for sure)
        DMemoryEntryUnchain(pM);
        // set the element pointers
        pM->pNext = NULL;
        pM->pPrev = NULL;
        return pM;
}

//the garbage collector
void MMU_helper_garbageCollector (
                TAPrealMMU *    pMMU            // MMU structure to init
                )
{
        TAPrealMMUMemory * pM = pMMU->pStart;
        int * pD = pMMU->memory;
        while (pM) {
                // check if we have to move the data
                if (pD != pM->pData) {
                        // move the data
                        memmove(pD,pM->pData,pM->count*sizeof(int));
                }
                // reset the destination pointer
                pD += pM->count;
                pM = pM->pNext;
        }
        // compressing memory finished
        // set the mmu vars new
        pMMU->elementsAvailable = pMMU->totalAvailable;
        pMMU->pUnusedData = pD;
}



// create a mmu
TAPMMU AP_MMU_create (size_t elementsNumber) {
        TAPrealMMU * pMMU;

        pMMU = (TAPrealMMU *) malloc (sizeof(TAPrealMMU)
```

```
        );
        if (!pMMU) return NULL;



        // setup lists
        pMMU->pStart = NULL;
        pMMU->pEnd = NULL;
        pMMU->pUnusedList = NULL;

        pMMU->elementsAvailable =elementsNumber;
        pMMU->pUnusedData = (int *) malloc (pMMU->
            elementsAvailable);
        pMMU->totalAvailable = pMMU->elementsAvailable;
        return pMMU;
}

// destroying the mmu
void AP_MMU_delete (TAPMMU mmu) {
        TAPrealMMU * pMMU = (TAPrealMMU *) mmu;

        TAPrealMMUMemory * pM;
        TAPrealMMUMemory * pMnext;

// 1. delete al mmu entry's
        // 1.1 unused entry
        pM = pMMU->pUnusedList;
        while (pM) {
                pMnext = pM->pNext;
                MemoryEntry_delete(pM);
                pM = pMnext;
        }
        pMMU->pUnusedList = NULL;
        // 1.2 used blocks
        pM = pMMU->pStart;
        while (pM) {
                pMnext = pM->pNext;
                MemoryEntry_delete(pM);
                pM = pMnext;
        }
        pMMU->pStart = NULL;
        pMMU->pEnd = NULL;
// 2. delete mmu memory
        free (pMMU->memory);


}
// getting memmory from the mmu
TAPMMUmemmory AP_MMU_get (TAPMMU mmu, size_t elements) {
        TAPrealMMU * pMMU = (TAPrealMMU *) mmu;

        // check if there is enough space
        if (pMMU->totalAvailable < elements) goto error;
        // check if we have to use the garbage collector
        if (pMMU->elementsAvailable < elements) {
```

```
                    // start garbage collector
                    MMU_helper_garbageCollector(pMMU);
        }
        // we have enough memory so let's allocate some

        // get a new entry
        TAPrealMMUMemory * pM;
        pM = MMU_helper_createMemoryEntry(pMMU);
        if (!pM) return NULL;
        // get some memory
        pM->pData = pMMU->pUnusedData;
        pM->count = elements;
        // refresh data
        pMMU->pUnusedData += elements;
        pMMU->totalAvailable -= elements;
        pMMU->elementsAvailable -= elements;
        // insert memory element at the end of the list and update last element
        pM->pPrev = pMMU->pEnd;
        if (pMMU->pEnd) pMMU->pEnd->pNext = pM;
        if (!pMMU->pStart) pMMU->pStart = pM;
        pMMU->pEnd = pM;

        return pM;
error:

        return NULL;
}

// free memmory from the mmu
void AP_MMU_free (TAPMMU mmu, TAPMMUmemmory memory) {
        TAPrealMMU * pMMU = (TAPrealMMU *) mmu;
        TAPrealMMUMemory * pM = (TAPrealMMUMemory *) memory;

        if (!pM) return;
        // set mmu settings
        if (pMMU->pStart == pM) pMMU->pStart = pM->pNext;
        if (pMMU->pEnd == pM) pMMU->pEnd = pM->pPrev;
        // unchain element
        DMemoryEntryUnchain (pM);
        // and put it to the chain of unused
        pM->pNext = pMMU->pUnusedList;
        pM->pPrev = NULL;
        if (pMMU->pUnusedList) {
                pMMU->pUnusedList->pPrev = pM;
        }
        pMMU->pUnusedList = pM;
        // now set the mmu data new
        pMMU->totalAvailable += pM->count;

}

// getting access to the MMU data
```

```
void * AP_MMU_getData (TAPMMUmemmory memory) {
        TAPrealMMUMemory * pM = (TAPrealMMUMemory *) memory;
        return pM->pData;
}

// the real interpreter
typedef struct SAPrealInterpreter {
        int                     state;                  // state of the IP
        int                     nextState;              // the next state of the IP
        TAPInterpreterCPU                       cpu;    // the IP core
        TAPInterpreterFuncCall *        code;           // the code
        int32_t                 instructionCount;       // number of instructions at the code
        TAPInterpreterVariable *        variables;      // the variables
        int32_t                 variableCount;          // number of the variables
        int                     sysEndian;              // endian of the system
} TAPrealInterpreter;

// create a new interpreter
TAPInterpreter APInterpreterCreate (void * pAP) {
        TAPrealInterpreter * pIP = NULL;
        pIP = (TAPrealInterpreter *) malloc (sizeof(TAPrealInterpreter));
        if (!pIP) return NULL;

        pIP->state = eAPInterpreterState_idle;
        pIP->nextState = eAPInterpreterState_idle;
        pIP->sysEndian = ((TAP *)pAP)->sysEndian;
        pIP->cpu.CF = 0;
        pIP->cpu.EF = 0;
        pIP->cpu.pCodeStart = NULL;
        pIP->cpu.pCodeEnd = NULL;
        pIP->cpu.pIP = NULL;

        pIP->code = NULL;
        pIP->instructionCount = 0;

        pIP->variables = NULL;
        pIP->variableCount = 0;

        return pIP;
```

```
}

// cleans the interpreter
void APInterpreterClean (TAPInterpreter IP) {
        TAPrealInterpreter * pIP = (TAPrealInterpreter
           *) IP;

        // clean code
        if (pIP->code) {
                free (pIP->code);
                pIP->code = NULL;
        }
        pIP->instructionCount = 0;

        // clean variables
        TAPInterpreterVariable * pV = pIP->variables;
        int i;
        for (i = 0; i < pIP->variableCount; i++) {
                if (pV->pVI) pV->pVI->pFkt_delete(pV->
                   pData);
                pV++;
        }
        if (pIP->variables) {
                free (pIP->variables);
                pIP->variables = NULL;
        }
        pIP->variableCount = 0;

}

// deletes the interpreter
void APInterpreterDelete (TAPInterpreter IP) {
        TAPrealInterpreter * pIP = (TAPrealInterpreter
           *) IP;
        APInterpreterClean(IP);
        free (pIP);
}

int APInterpreterStateRun(TAPInterpreter IP) {
        TAPrealInterpreter * pIP = (TAPrealInterpreter
           *) IP;
        TAPInterpreterFuncCall * pFC;

        // setup cpu
        pIP->cpu.CF = 0;
        pIP->cpu.EF = 0;
        pIP->cpu.pIP = pIP->code;
        pIP->cpu.pCodeStart = pIP->code;
        pIP->cpu.pCodeEnd = pIP->code + pIP->
           instructionCount;

        // run code
        while (eAPInterpreterState_run == pIP->state) {
                pFC = pIP->cpu.pIP;
                // check if we reached the end of the
```

```
                        code
                if (pFC > pIP->cpu.pCodeEnd) {
                        return 0;
                }
                // execute command
                pFC->pHALFkt (&(pIP->cpu), pFC->param);
                // check error flags
                if (pIP->cpu.EF) {
                        return -1;
                }
        }
        return 1;
}


// process the actual state
int APInterpreterProcessState(TAPInterpreter IP){
        TAPrealInterpreter * pIP = (TAPrealInterpreter *) IP;
        pIP->state = pIP->nextState;
        int rc = 0;

        switch (pIP->state) {
                case eAPInterpreterState_idle:
                        break;
                case eAPInterpreterState_loadProgramm:
                        break;
                case eAPInterpreterState_run:
                        rc = APInterpreterStateRun(IP);
                        if (rc >= 0) pIP->state = eAPInterpreterState_idle;
                        break;
                case eAPInterpreterState_oneStep:
                        break;
                case eAPInterpreterState_halt:
                        break;
                default:
                        return -10;
        }
        return rc;
}


// set interpreter state
int APInterpreterSetState (TAPInterpreter IP, int msgEndian, int32_t state) {
        TAPrealInterpreter * pIP = (TAPrealInterpreter *) IP;
        if (msgEndian != pIP->sysEndian) {
                APendianConversation32Bit((uint32_t *)&state);
        }
        pIP->nextState = (int) state;
        return 0;
}
```

```
// setup the interpreter for a new program
int APInterpreterInitNewProgramm (TAPInterpreter IP, int msgEndian, int32_t instructionsNumber, int32_t VariableNumber) {
        TAPrealInterpreter * pIP = (TAPrealInterpreter *) IP;
        int i;

        APInterpreterClean (IP);

        if (msgEndian != pIP->sysEndian) {
                APendianConversation32Bit((uint32_t *)&instructionsNumber);
                APendianConversation32Bit((uint32_t *)&VariableNumber);
        }

        pIP->code = (TAPInterpreterFuncCall *) malloc(sizeof(TAPInterpreterFuncCall)*instructionsNumber);
        pIP->instructionCount = instructionsNumber;

        pIP->variables = (TAPInterpreterVariable *) malloc(sizeof(TAPInterpreterVariable) * (VariableNumber));
        for (i = 0; i < VariableNumber;i++) {
                pIP->variables[i].pData = NULL;
                pIP->variables[i].pVI = NULL;
        }
        pIP->variableCount = VariableNumber;

        return 0;
}

// load a variable/~array to an index
int APInterpreterLoadVar (TAPInterpreter IP, int msgEndian, int32_t index, int32_t varTypeID, int32_t numberOfElements)
{
        TAPrealInterpreter * pIP = (TAPrealInterpreter *) IP;
        if (msgEndian != pIP->sysEndian) {
                APendianConversation32Bit((uint32_t *)&index);
                APendianConversation32Bit((uint32_t *)&varTypeID);
        }

        if ((index < 0) || (index > pIP->variableCount)) return -1;

        // set pointer to the runtime variable
        TAPInterpreterVariable * pRTV = pIP->variables + index;
```

```
        THAL_Variable const * pV = HALfindVar(varTypeID);
        if (!pV) return -2;

        pRTV->pData = pV->pFkt_create((unsigned int)numberOfElements);
        //if (!pRTV->pData) return -3;

        pRTV->pVI = pV;
        return 0;
}

// load a single Instruction to an index
int APInterpreterLoadInstr (TAPInterpreter IP,int msgEndian, int32_t index, int32_t * pRawInstr)
{
        TAPrealInterpreter * pIP = (TAPrealInterpreter *) IP;
        if (msgEndian != pIP->sysEndian) {
                APendianConversation32Bit((uint32_t *)&index);
        }
        if ((index < 0) || (index > pIP->instructionCount)) return -1;
        TAPInterpreterFuncCall * pIFC = pIP->code + index;
        memset (pIFC, 0, sizeof(TAPInterpreterFuncCall));

        // get function
        int32_t fid = *pRawInstr;
        if (msgEndian != pIP->sysEndian) {
                APendianConversation32Bit((uint32_t *)&fid);
        }
        THALFunction const * pF = HALfindFunction(fid);
        if (!pF) return -2;
        pIFC->pHALFkt = pF->pfktHAL;

        // convert parameters
        pRawInstr++; // set to the first parameter
        int i;
        THALFunctionParam const * pP = pF->paramList.pL;
        TuAPInterpreterFunctionParameter * pIFP = pIFC->param;
        for (i = 0; i < pF->paramList.number; i++) {
                if (APconvertRawParamData (msgEndian,pIP->sysEndian,pRawInstr,pP,pIFP,pIP->variables)) return -3;
                pP++;
                pRawInstr++;
                pIFP++;
        }
        return 0;
}
```

```
typedef struct SAPrealMsgSystem {
	TAPMsg *				pOldRXMsg;		// pointer to the oldest received messages
	TAPMsg *				pNewRXMsg;		// pointer to the newest received messages
	TAPMMU					mmu;			// the mmu
	int					sysEndianness;	// the system endianness
	int					messagecounter; // a counter for checkin if a new message has been received


} TAPrealMsgSystem;


int SMinitial (
			void *			pVoidSM,		// pointer to the statemachine
			uint32_t *		pD,			// pointer to the data
			int				number		// the number of data elements
	);

int SMdata (
			void *			pVoidSM,		// pointer to the statemachine
			uint32_t *		pD,			// pointer to the data
			int				number		// the number of data elements
	);

int SMmessageFinished (
			void *			pVoidSM		// pointer to the statemachine
	);



// create AP message system
TAPMsgSystem APMScreate (
			TAPMMU				mmu,			// the mmu
```

```
                    int             sysEndianness    // the system endianness
        ) {
        TAPrealMsgSystem * pMS = (TAPrealMsgSystem *) malloc (sizeof(TAPrealMsgSystem));
        if (!pMS) return NULL;
        pMS->mmu = mmu;
        pMS->sysEndianness = sysEndianness;
        pMS->pOldRXMsg = NULL;
        pMS->pNewRXMsg = NULL;
        pMS->messagecounter = 0;


        return pMS;
}

void APMSdelete (
        TAPMsgSystem ms
        ) {
        TAPrealMsgSystem * pMS = (TAPrealMsgSystem *) ms;
        free (pMS);
}


// frees a message from the message system
void APMSdeleteMsg (
        TAPMsgSystem    ms,
        TAPMsg *                pM
        ) {
        TAPrealMsgSystem * pMS = (TAPrealMsgSystem *) ms;
        AP_MMU_free(pMS->mmu,pM->memory);
}

// get memory for a new message
TAPMsg * APMSgetNewMsg (
                TAPrealMsgSystem *      pMsgSys,
                int             dataElementsNumber,
                const TAPMsgDrv *       pDrv
                ) {
        TAPMMUmemmory m = AP_MMU_get (pMsgSys->mmu, sizeof(TAPMsg)/sizeof(int) + dataElementsNumber*sizeof(uint32_t)/sizeof(int));
        if (!m) return NULL;

        int * pRD = (int *) AP_MMU_getData(m);
        TAPMsg * pM = (TAPMsg *) pRD;
        pM->memory = m;
        pM->extraData.pDrv = pDrv;
        pM->pH = (TAPMsgHeader *)((int *) pRD + sizeof(TAPMsg)/sizeof(int));
```

```
        pM->pData = (int *)pM->pH + sizeof(TAPMsgHeader)/sizeof(int);
        pM->pNext = NULL;
        return pM;
}

// insert a new message into the message queue
void APMSInsertMsg (
                TAPrealMsgSystem *      pMS,
                TAPMsg *                        pM
        ) {

        if (pMS->pNewRXMsg) {
                pMS->pNewRXMsg->pNext = pM;
        }
        pMS->pNewRXMsg = pM;
        if (!pMS->pOldRXMsg) {
                pMS->pOldRXMsg = pM;
        }
        pMS->messagecounter++;



}

// get oldest message
TAPMsg * APMSgetMsg (
                TAPMsgSystem            ms,             // the message system
                TAPMessageID            msgID,  // if 0 all messages are allowed
                TAPNodeID                       sender,         // if 0 all senders are allowed
                uint32_t                        mNumber         // if 0 all numbers are allowed
        ) {

        TAPrealMsgSystem * pMS = (TAPrealMsgSystem *) ms;

        TAPMsg * res = NULL;
        // search msg list

        TAPMsg * pM = pMS->pOldRXMsg;
        TAPMsg * pAntecessorM =  NULL;
        // flags
        int senderOK;
        int msgIDok;
        int numberOK;

        while (pM) {
                senderOK = 0;
                msgIDok = 0;
```

```
        numberOK = 0;

        if (!sender) {
            senderOK = 1;
        } else {
            if (*(pM->pH)[eAPMsgHeaderPosition_sender] == sender) senderOK = 1;
        }
        if (!msgID) {
            msgIDok = 1;
        } else {
            if (*(pM->pH)[eAPMsgHeaderPosition_msgTypeID] == msgID) msgIDok = 1;
        }
        if (!mNumber) {
            numberOK = 1;
        } else {
            if (*(pM->pH)[eAPMsgHeaderPosition_msgNumber] == mNumber) numberOK = 1;
        }

        if ((senderOK) && (msgIDok) && (numberOK)) {
            res = pM;
            goto exit;
        }
        // the id was wrong but the number was ok >> so the action failed
        if (numberOK) goto fail;
        pAntecessorM = pM;
        pM = pM->pNext;
    }
fail:
    return NULL;
exit:
    if (pAntecessorM) {
        pAntecessorM->pNext = pM->pNext;
    } else {
        pMS->pOldRXMsg = pM->pNext;
    }
    if (pM == pMS->pNewRXMsg) {
        pMS->pNewRXMsg = NULL;
    }

    return res;
}

// wait till a new message has been received
void APMSwaitForNewMessage (TAPrealMsgSystem * pMS)
{
    volatile int mc;

```

```
        mc = pMS->messagecounter;

        while (mc == pMS->messagecounter) {

        }
}

// returns 0 if a new message is available
inline int APMSisMessageAvailble (TAPrealMsgSystem * pMS ) {
        return (!pMS->pOldRXMsg) ? 0 : -1;
}

// ========================================
// the receive state machine
// ========================================

// the receive state machine state function for receiving the msg header
int SMinitial (
                void *                  pVoidSM,                // pointer to the statemachine
                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 = 1; i < eAPMsgHeaderPosition_headerElementNumber;i++) {
			APendianConversation32Bit(&pSM->header[i]);
		}
	}
	// 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;
	for (i = 0; i < eAPMsgHeaderPosition_headerElementNumber;i++) {
		*pSM->pD = pSM->header[i];
		pSM->pD++;
	}
	pSM->elementsLeft = pSM->header[eAPMsgHeaderPosition_length];
	// set up the data
	// 1. check if there is an data element
	if (!pSM->elementsLeft) {
		// no! now finish the message
		return SMmessageFinished(pVoidSM);
	}
	// 2. yes
	// 2.1 setup the sm for the data receiving
	pSM->state = SMdata;
	// 2.2 now check if we have to copy some data
	number -= copyAmount;
	if (number) {
		// set the data pointer
		pD += copyAmount;
		// and copy the data
		return SMdata (pVoidSM,pD,number);
	}
	return 0;
}
```

```
// the receive state machine state function for receiving the data
int SMdata (
                void *                  pVoidSM,                // pointer to the statemachine
                uint32_t *              pD,                     // pointer to the data
                int                     number                  // the number of data elements
        ) {
        TAPReceiveStateMachine * pSM = (TAPReceiveStateMachine *) pVoidSM;
        int                     copyAmount = number;
        int                     i;
        // 1. transfer the data
        // do some clipping
        if (pSM->elementsLeft < copyAmount) copyAmount = pSM->elementsLeft;
        // copy
        for (i = 0; i < copyAmount;i++) {
                *pSM->pD = *pD;
                pSM->pD++;
                pD++;
        }
        // set statemachine work data
        pSM->elementsLeft -= copyAmount;
        // check if we have to change the statemachine
        if (pSM->elementsLeft) return 0;
        int res = SMmessageFinished (pVoidSM);
        if (res) return res;

        // check if there some bytes left to copy
        number -= copyAmount;
        if (number) {
                // set the data pointer
                pD += copyAmount;
                // and copy the data
                return pSM->state (pVoidSM,pD,number);
        }
        return 0;

}

// this function is called when all data have been received
int SMmessageFinished (
                void *                  pVoidSM                 // pointer to the statemachine
        ) {
        TAPReceiveStateMachine *
```

```
           pSM = (TAPReceiveStateMachine *) pVoidSM;
        TAPMsg *
                          pM;
        // 1. reset SM
        // set the helper
        pSM->elementsLeft =
           eAPMsgHeaderPosition_headerElementNumber;
        pSM->pD = pSM->header;

        // data
        pM = pSM->pMsg; // save msg info for inserting
        pSM->pMsg = NULL;

        // right state function
        pSM->state = SMinitial;          // the state

        // 2. insert message at the message system
        APMSInsertMsg ((TAPrealMsgSystem *)pSM->pMS,pM);
        return 0;
}


// inits the state machine
void APInitReceiveStateMachine (
                TAPReceiveStateMachine *
                          pSM,    // pointer to the
                   state machine
                TAPMsgSystem
                                  pMS,    // pointer to
                    the message system
                const TAPMsgDrv   *
                                  pDrv    // the driver
                    associated with the statemachine
        ) {
        pSM->state = SMinitial;
        pSM->pMS = pMS;
        pSM->pDrv = pDrv;
        // set the helper
        pSM->elementsLeft =
           eAPMsgHeaderPosition_headerElementNumber;
        pSM->pD = pSM->header;

        // data
        pSM->pMsg = NULL;
}

int APHandleMsg (
                TAP *           pAP,
                TAPMsg *        pM
        ) {

        TAPMessageID
                                  msgID;
        const THALMsgProcessMessageAssociation *
           pMsgIDandFunctAsso;
```

```c
        int                                             i;

        // get message id
        msgID = (*(pM->pH))[eAPMsgHeaderPosition_msgTypeID];
        // search handler
        pMsgIDandFunctAsso = gHALMsgProcessRXHandlers.pL;
        for (i = 0; i < gHALMsgProcessRXHandlers.number; i++) {
                if (((TAPMessageID)pMsgIDandFunctAsso->msgID) == msgID) {
                        return pMsgIDandFunctAsso->pfktHandle(pAP,pM);
                }
                pMsgIDandFunctAsso++;
        }
        return -1;
}

void APMessageProcessingThread (TAP * pAP) {

        TAPrealMsgSystem *               pMS = (TAPrealMsgSystem *) pAP->MS;
        TAPMsg *                                pM;
        TAPNodeID                               recv;
        while (1) {
                // wait for a message
                APMSwaitForNewMessage(pMS);
                // get the message
                pM = APMSgetMsg (pMS,0,0,0);
                // search the message handler
                recv = (*(pM->pH))[eAPMsgHeaderPosition_receiver];
                if ((recv == dAPNodeID_ALL) || (recv == pAP->nodeID)) {
                        if(APHandleMsg (pAP,pM)) goto exit;
                }
                // free memory
                AP_MMU_free(pMS->mmu, pM->memory);
        }
exit:
        AP_MMU_free(pMS->mmu, pM->memory);
}
```

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:

```
// ================================
// AP uuid = 11
// ================================

// inits the AP
int APinit (
                        TAP *
                           pAP,
                        TAPNodeID
                           nodeID,
                        const TAPMsgDrv *        pDrvList
                           ,
                        const int
                           driverNumber,
                        size_t
                           messagePoolSize,
                        int
                                   sysEndian
                )
{
        gAPendianFlag = sysEndian;

        pAP->nodeID = nodeID;
        pAP->pNodeList = NULL;
        pAP->pDrvList = pDrvList;
        pAP->driverNumber = driverNumber;
        pAP->msgSysMMU = AP_MMU_create(messagePoolSize);
        pAP->IP = APInterpreterCreate(pAP);
        pAP->MS = APMScreate (pAP->msgSysMMU,sysEndian);
        pAP->msgNumber = 0;
        pAP->APstate = eAPstate_idle;

        if(
                (!pAP->msgSysMMU)||
                (!pAP->IP)||
                (!pAP->MS)
                ) return -1;

        msp430_initHW(10);

        // init drv
        TAPMsgDrv * pDrv = (TAPMsgDrv *)pDrvList;
        int i;
        for (i = 0; i < driverNumber; i++) {
                pDrv->pfkt_open(pAP, pDrv);
                pDrv++;
        }

        // login the ap to the message system
```

```
        return TX_login(pAP);
}

// deletes the AP
void APdelete (TAP * pAP)
{
        TX_logout(pAP);
        // close & destroy drv
        TAPMsgDrv * pDrv = (TAPMsgDrv *)pAP->pDrvList;
        int i;
        for (i = 0; i < pAP->driverNumber; i++) {
                pDrv->pfkt_close(pDrv);
                pDrv->pfkt_destroy(pDrv);
                pDrv++;
        }
        APMSdelete (pAP->MS);
        APInterpreterDelete(pAP->IP);
        AP_MMU_delete(pAP->msgSysMMU);
}

// find a node at the list
TAPNode * APfindNode(TAP * pAP, TAPNodeID nodeID) {
        TAPNode * pN = pAP->pNodeList;
        while (pN) {
                if (pN->nodeID == nodeID) return pN;
                pN = pN->pNext;
        };
        return NULL;
}

// adds a new node to the node list
int APaddNode(TAP * pAP, TAPNodeID newNodeID, const TAPMsgDrv * pDrv) {
        if (APfindNode(pAP,newNodeID)) return 1;
        TAPNode * pN = (TAPNode *) malloc(sizeof(TAPNode));
        if (!pN) return -1;
        pN->nodeID = newNodeID;
        pN->pDrv = pDrv;
        pN->pNext = pAP->pNodeList;
        pAP->pNodeList = pN;
        return 0;
}

// removes a node from the node list
void APremoveNode(TAP * pAP, TAPNodeID nodeID){
        TAPNode * pAntN = pAP->pNodeList; // antecessor node
        TAPNode * pActN = pAP->pNodeList; // actual node

        while (pActN) {
                // compare node id's
                if (pActN->nodeID == nodeID) {
                        // unchain

```

```
			// check if we at the first position at the list
			if (pAP->pNodeList == pAntN) {
				// reset the pointer
				pAP->pNodeList = pActN->pNext;
			} else {
				// set the antecessor
				pAntN->pNext = pActN->pNext;
			}
			// free node
			free(pActN);
			// and abort
			return;
		}
		// the actual element becomes the precessor element
		pAntN = pActN;
		pActN = pActN->pNext;
	}
}

// get a new message number
unsigned int APgetNewMessageNumber (TAP *pAP) {
	pAP->msgNumber++;
	return pAP->msgNumber;
}

// find the driver associated with der nodeID
const TAPMsgDrv * APfindDrvBySenderID (TAP * pAP, TAPNodeID node) {
	TAPNode * pN = pAP->pNodeList;
	while (pN) {
		if (pN->nodeID == node) {
			return pN->pDrv;
		}
		pN = pN->pNext;
	}
	return NULL;
}

// runs the AP
int APrun(TAP *pAP) {
	pAP->APstate = eAPstate_run;
	return 0;
}

// ================================
// MMU functions
// ================================

typedef struct SAPrealMMUMemory {
	uint32_t * pData;                     // the data
```

```
	int                      count;                 // amount of data elements
	struct SAPrealMMUMemory *       pNext;          // next element
	struct SAPrealMMUMemory *       pPrev;          // previous element
} TAPrealMMUMemory;

//the mmu type
typedef struct SAPrealMMU {
	uint32_t *                                memory;          // the memory block
	TAPrealMMUMemory *              pStart;         // first element
	TAPrealMMUMemory *              pEnd;           // second element
	TAPrealMMUMemory *              pUnusedList;            // list with the unused elements
	uint32_t  *                     pUnusedData;            // pointer to the unused memory
	int                      elementsAvailable;      // amount of elements witch are available without using the garbage collector
	int                      totalAvailable;         // total amount of free bytes

} TAPrealMMU;

// ========================================
// memory entry functions
// ========================================

// a little macro for unchaining an element
#define DMemoryEntryUnchain(pM) \
	if (pM->pNext) pM->pNext->pPrev = pM->pPrev; \
	if (pM->pPrev) pM->pPrev->pNext = pM->pNext


//creates an memory entry
TAPrealMMUMemory * MemoryEntry_create () {
	TAPrealMMUMemory * pM = NULL;

	pM = (TAPrealMMUMemory *) malloc(sizeof(TAPrealMMUMemory));
	if (!pM) return NULL;
	pM->pData = NULL;
	pM->count = 0;
	pM->pNext = NULL;
	pM->pPrev = NULL;

	return pM;
}
```

```

//deletes an memory Entry
void MemoryEntry_delete (
                TAPrealMMUMemory * pM    // the memory to delete
                )
{
        // put the entry out of the chain
        DMemoryEntryUnchain(pM);
        // now we delete it
        free(pM);
}

// ========================================
// mmu helper
// ========================================

//alloc if needed a new memory entry
TAPrealMMUMemory * MMU_helper_createMemoryEntry (
                TAPrealMMU *     pMMU             // MMU structure to init
                )
{
        // check if we have to alloc a new memory entry
        if (!pMMU->pUnusedList) return MemoryEntry_create();
        // no there is some left at the list
        TAPrealMMUMemory * pM;
        // take the first one
        pM = pMMU->pUnusedList;
        // reset the list
        pMMU->pUnusedList = pM->pNext;
        // now unchain the element (for sure)
        DMemoryEntryUnchain(pM);
        // set the element pointers
        pM->pNext = NULL;
        pM->pPrev = NULL;
        return pM;
}

//the garbage collector
void MMU_helper_garbageCollector (
                TAPrealMMU *     pMMU             // MMU structure to init
                )
{
        TAPrealMMUMemory * pM = pMMU->pStart;
        uint32_t * pD = pMMU->memory;
        while (pM) {
                // check if we have to move the data
                if (pD != pM->pData) {
                        // move the data
                        memmove(pD,pM->pData,pM->count*sizeof(uint32_t));
                }
```

```
                // reset the destination pointer
                pD += pM->count;
                pM = pM->pNext;
        }
        // compressing memory finished
        // set the mmu vars new
        pMMU->elementsAvailable = pMMU->totalAvailable;
        pMMU->pUnusedData = pD;
}

// create a mmu
TAPMMU AP_MMU_create (size_t elementsNumber) {
        TAPrealMMU * pMMU;

        pMMU = (TAPrealMMU *) malloc (sizeof(TAPrealMMU)
           );
        if (!pMMU) return NULL;


        pMMU->memory = (uint32_t *) malloc (
           elementsNumber*sizeof(uint32_t));
        pMMU->pUnusedData = pMMU->memory;

        // setup lists
        pMMU->pStart = NULL;
        pMMU->pEnd = NULL;
        pMMU->pUnusedList = NULL;

        pMMU->elementsAvailable =elementsNumber;

        pMMU->totalAvailable = elementsNumber;
        return pMMU;
}

// destroying the mmu
void AP_MMU_delete (TAPMMU mmu) {
        TAPrealMMU * pMMU = (TAPrealMMU *) mmu;

        TAPrealMMUMemory * pM;
        TAPrealMMUMemory * pMnext;

// 1. delete al mmu entry's
        // 1.1 unused entry
        pM = pMMU->pUnusedList;
        while (pM) {
                pMnext = pM->pNext;
                MemoryEntry_delete(pM);
                pM = pMnext;
        }
        pMMU->pUnusedList = NULL;
        // 1.2 used blocks
        pM = pMMU->pStart;
        while (pM) {
                pMnext = pM->pNext;
                MemoryEntry_delete(pM);
```

```
                pM = pMnext;
        }
        pMMU->pStart = NULL;
        pMMU->pEnd = NULL;
// 2. delete mmu memory
        free (pMMU->memory);
}

// getting memmory from the mmu
TAPMMUmemmory AP_MMU_get (TAPMMU mmu, size_t elements) {
        TAPrealMMU * pMMU = (TAPrealMMU *) mmu;

        // check if there is enough space
        if (pMMU->totalAvailable < elements) goto error;
        // check if we have to use the garbage collector
        if (pMMU->elementsAvailable < elements) {
                // start garbage collector
                MMU_helper_garbageCollector(pMMU);
        }
        // we have enough memory so let's allocate some

        // get a new entry
        TAPrealMMUMemory * pM;
        pM = MMU_helper_createMemoryEntry(pMMU);
        if (!pM) return NULL;
        // get some memory
        pM->pData = pMMU->pUnusedData;
        pM->count = elements;
        // refresh data
        pMMU->pUnusedData += elements;
        pMMU->totalAvailable -= elements;
        pMMU->elementsAvailable -= elements;
        // insert memory element at the end of the list and update last element
        pM->pPrev = pMMU->pEnd;
        if (pMMU->pEnd) pMMU->pEnd->pNext = pM;
        if (!pMMU->pStart) pMMU->pStart = pM;
        pMMU->pEnd = pM;
        return pM;
error:
        return NULL;
}

// free memmory from the mmu
void AP_MMU_free (TAPMMU mmu, TAPMMUmemmory memory) {
        TAPrealMMU * pMMU = (TAPrealMMU *) mmu;
        TAPrealMMUMemory * pM = (TAPrealMMUMemory *) memory;

        if (!pM) return;
        // set mmu settings
        if (pMMU->pStart == pM) pMMU->pStart = pM->pNext;
        if (pMMU->pEnd == pM) pMMU->pEnd = pM->pPrev;
        // unchain element
```

```
        DMemoryEntryUnchain (pM);
        // and put it to the chain of unused
        pM->pNext = pMMU->pUnusedList;
        pM->pPrev = NULL;
        if (pMMU->pUnusedList) {
                pMMU->pUnusedList->pPrev = pM;
        }
        pMMU->pUnusedList = pM;
        // now set the mmu data new
        pMMU->totalAvailable += pM->count;
}

// getting access to the MMU data
void * AP_MMU_getData (TAPMMUmemmory memory) {
        TAPrealMMUMemory * pM = (TAPrealMMUMemory *) memory;
        return pM->pData;
}

// the real interpreter
typedef struct SAPrealInterpreter {
        TAP * pAP;                                      // pointer to the audio processor
        int             state;                          // state of the IP
        int             nextState;                      // the next state of the IP
        TAPInterpreterCPU                       cpu;    // the IP core
        TAPInterpreterFuncCall *        code;           // the code
        int32_t instructionCount;                       // number of instructions at the code
        TAPInterpreterVariable *        variables;      // the variables
        int32_t variableCount;                          // number of the variables
} TAPrealInterpreter;

// create a new interpreter
TAPInterpreter APInterpreterCreate (void * pAP) {
        TAPrealInterpreter * pIP = NULL;
        pIP = (TAPrealInterpreter *) malloc (sizeof(TAPrealInterpreter));
        if (!pIP) return NULL;

        pIP->pAP = pAP;
        pIP->state = eAPInterpreterState_idle;
        pIP->nextState = eAPInterpreterState_idle;
```

```
        pIP->cpu.IP = pIP;
        pIP->cpu.CF = 0;
        pIP->cpu.EF = 0;
        pIP->cpu.pCodeStart = NULL;
        pIP->cpu.pCodeEnd = NULL;
        pIP->cpu.pIP = NULL;

        pIP->code = NULL;
        pIP->instructionCount = 0;

        pIP->variables = NULL;
        pIP->variableCount = 0;

        return pIP;
}

// cleans the interpreter
void APInterpreterClean (TAPInterpreter IP) {
        TAPrealInterpreter * pIP = (TAPrealInterpreter
           *) IP;

        // clean code
        if (pIP->code) {
                free (pIP->code);
                pIP->code = NULL;
        }
        pIP->instructionCount = 0;

        // clean variables
        TAPInterpreterVariable * pV = pIP->variables;
        int i;
        for (i = 0; i < pIP->variableCount; i++) {
                if (pV->pVI) pV->pVI->pFkt_delete(pV->
                   pData);
                pV++;
        }
        if (pIP->variables) {
                free (pIP->variables);
                pIP->variables = NULL;
        }
        pIP->variableCount = 0;

}

// deletes the interpreter
void APInterpreterDelete (TAPInterpreter IP) {
        TAPrealInterpreter * pIP = (TAPrealInterpreter
           *) IP;
        APInterpreterClean(IP);
        free (pIP);
}

extern void drv_3_feedRecvStateM (void);

int APInterpreterStateRun(TAPInterpreter IP) {
```

```
        TAPrealInterpreter * pIP = (TAPrealInterpreter *) IP;
        TAPInterpreterFuncCall * pFC;
        int cc; // cycle counter

        // setup cpu
        pIP->cpu.CF = 0;
        pIP->cpu.EF = 0;
        pIP->cpu.pIP = pIP->code;
        pIP->cpu.pCodeStart = pIP->code;
        pIP->cpu.pCodeEnd = pIP->code + pIP->instructionCount;

        // run code
        cc = 10;
        while (eAPInterpreterState_run == pIP->state) {
                pFC = pIP->cpu.pIP;
                // check if we reached the end of the code
                if (pFC > pIP->cpu.pCodeEnd) {
                        return 0;
                }
                // execute command
                pFC->pHALFkt (&(pIP->cpu), pFC->param);
                // check error flags
                if (pIP->cpu.EF) {
                        return -1;
                }
                // process msg receiving
                drv_3_feedRecvStateM ();

                // check message system after x cycles
                if (!cc) {
                        APMessageProcess(pIP->pAP);
                        cc = 100;
                } else {
                        cc--;
                }

        }
        return 1;
}

// process the actual state
int APInterpreterProcessState(TAPInterpreter IP){
        TAPrealInterpreter * pIP = (TAPrealInterpreter *) IP;
        pIP->state = pIP->nextState;
        int rc = 0;

        switch (pIP->state) {
                case eAPInterpreterState_idle:
                        break;
                case eAPInterpreterState_loadProgramm:
                        break;
```

```
			case eAPInterpreterState_run:
				rc = APInterpreterStateRun(IP);
				if (rc >= 0) pIP->state = eAPInterpreterState_idle;
				break;
			case eAPInterpreterState_oneStep:
				break;
			case eAPInterpreterState_halt:
				break;
			default:
				return -10;
		}
		return rc;
}


// set interpreter state
int APInterpreterSetState (TAPInterpreter IP, int msgEndian, int32_t state) {
		TAPrealInterpreter * pIP = (TAPrealInterpreter *) IP;
		if (msgEndian != gAPendianFlag) {
			APendianConversation32Bit((uint32_t *)&state, msgEndian);
		}
		pIP->nextState = (int) state;
		return 0;
}

// setup the interpreter for a new program
int APInterpreterInitNewProgramm (TAPInterpreter IP, int msgEndian, int32_t instructionsNumber, int32_t VariableNumber) {
		TAPrealInterpreter * pIP = (TAPrealInterpreter *) IP;
		int i;

		APInterpreterClean (IP);

		if (msgEndian != gAPendianFlag) {
			APendianConversation32Bit((uint32_t *)&instructionsNumber, msgEndian);
			APendianConversation32Bit((uint32_t *)&VariableNumber, msgEndian);
		}

		pIP->code = (TAPInterpreterFuncCall *) malloc(sizeof(TAPInterpreterFuncCall)*instructionsNumber);
		pIP->instructionCount = instructionsNumber;

		pIP->variables = (TAPInterpreterVariable *) malloc(sizeof(TAPInterpreterVariable) * (VariableNumber));
		for (i = 0; i < VariableNumber;i++) {
```

```
                pIP->variables[i].pData = NULL;
                pIP->variables[i].pVI = NULL;
        }
        pIP->variableCount = VariableNumber;

        return 0;
}

// load a variable/~array to an index
int APInterpreterLoadVar (TAPInterpreter IP, int msgEndian, int32_t index, int32_t varTypeID, int32_t numberOfElements)
{
        TAPrealInterpreter * pIP = (TAPrealInterpreter *) IP;
        if (msgEndian != gAPendianFlag) {
                APendianConversation32Bit((uint32_t *)&index, msgEndian);
                APendianConversation32Bit((uint32_t *)&varTypeID, msgEndian);
                APendianConversation32Bit((uint32_t *)&numberOfElements, msgEndian);
        }

        if ((index < 0) || (index > pIP->variableCount)) return -1;

        // set pointer to the runtime variable
        TAPInterpreterVariable * pRTV = pIP->variables + index;
        THAL_Variable const * pV = HALfindVar(varTypeID);
        if (!pV) return -2;

        pRTV->pData = pV->pFkt_create((unsigned int)numberOfElements);
        //if (!pRTV->pData) return -3;

        pRTV->pVI = pV;
        return 0;
}

// load a single Instruction to an index
int APInterpreterLoadInstr (TAPInterpreter IP,int msgEndian, int32_t index, int32_t * pRawInstr)
{
        TAPrealInterpreter * pIP = (TAPrealInterpreter *) IP;
        if (msgEndian != gAPendianFlag) {
                APendianConversation32Bit((uint32_t *)&index, msgEndian);
        }
        if ((index < 0) || (index > pIP->instructionCount)) return -1;
        TAPInterpreterFuncCall * pIFC = pIP->code +
```

```
        index;
        memset (pIFC, 0, sizeof(TAPInterpreterFuncCall));

        // get function
        int32_t fid = *pRawInstr;
        if (msgEndian != gAPendianFlag) {
                APendianConversation32Bit((uint32_t *)&fid, msgEndian);
        }
        THALFunction const * pF = HALfindFunction(fid);
        if (!pF) return -2;
        pIFC->pHALFkt = pF->pfktHAL;

        // convert parameters
        pRawInstr++; // set to the first parameter
        int i;
        THALFunctionParam const * pP = pF->paramList.pL;
        TuAPInterpreterFunctionParameter * pIFP = pIFC->param;
        for (i = 0; i < pF->paramList.number; i++) {
                if (APconvertRawParamData (msgEndian, pRawInstr,pP,pIFP,pIP->variables)) return -3;
                pP++;
                pRawInstr++;
                pIFP++;
        }
        return 0;
}

// gets the varaible by it's index
TAPInterpreterVariable * APInterpreterGetVariableByIndex (TAPInterpreter IP, int index) {
        return &(((TAPrealInterpreter *) IP)->variables[index]);
}

// gets the AP from the IP
void * APInterpreterGetAPfromIP (TAPInterpreter IP) {
        return ((TAPrealInterpreter *) IP)->pAP;
}


typedef struct SAPrealMsgSystem {
        TAPMsg *                        pOldRXMsg;              // pointer to the oldest received messages
        TAPMsg *                        pNewRXMsg;              // pointer to the newest received messages
        TAPMMU                          mmu;                    // the mmu
        int                             sysEndianness;  // the system endianness
```

```
        int messagecounter; // a counter for checkin if a new message has been received


} TAPrealMsgSystem;


int SMinitial (
                void *              pVoidSM,        // pointer to the statemachine
                uint32_t *          pD,             // pointer to the data
                int                 number          // the number of data elements
        );

int SMdata (
                void *              pVoidSM,        // pointer to the statemachine
                uint32_t *          pD,             // pointer to the data
                int                 number          // the number of data elements
        );

int SMmessageFinished (
                void *              pVoidSM         // pointer to the statemachine
        );



// create AP message system
TAPMsgSystem APMScreate (
                TAPMMU              mmu,            // the mmu
                int                 sysEndianness   // the system endianness
        ) {
        TAPrealMsgSystem * pMS = (TAPrealMsgSystem *) malloc (sizeof(TAPrealMsgSystem));
        if (!pMS) return NULL;
        pMS->mmu = mmu;
        pMS->sysEndianness = sysEndianness;
        pMS->pOldRXMsg = NULL;
        pMS->pNewRXMsg = NULL;
        pMS->messagecounter = 0;
```

```


        return pMS;
}

void APMSdelete (
        TAPMsgSystem ms
        ) {
        TAPrealMsgSystem * pMS = (TAPrealMsgSystem *) ms;
        free (pMS);
}


// frees a message from the message system
void APMSdeleteMsg (
        TAPMsgSystem    ms,
        TAPMsg *                pM
        ) {
        TAPrealMsgSystem * pMS = (TAPrealMsgSystem *) ms;
        AP_MMU_free(pMS->mmu,pM->memory);
}

// get memory for a new message
TAPMsg * APMSgetNewMsg (
                TAPrealMsgSystem *      pMsgSys,
                int dataElementsNumber,
                const TAPMsgDrv *       pDrv
                ) {
        // we go for sure that we get enough memory
        // if mod(sizeof(TAPMsg)/sizeof(uint32_t)) != 0 we need one uint32_t more -> +1
        TAPMMUmemmory m = AP_MMU_get (pMsgSys->mmu, sizeof(TAPMsg)/sizeof(uint32_t) + 1+ eAPMsgHeaderPosition_headerElementNumber + dataElementsNumber);
        if (!m) return NULL;

        // set the pointers
        uint32_t * pRD = (uint32_t *) AP_MMU_getData(m);
        TAPMsg * pM = (TAPMsg *) pRD;
        pM->memory = m;
        pM->extraData.pDrv = pDrv;
        pM->pH = (TAPMsgHeader *)((uint32_t *) pRD + sizeof(TAPMsg)/sizeof(uint32_t)+1);
        pM->pData = (uint32_t *)pM->pH + sizeof(TAPMsgHeader)/sizeof(uint32_t);
        pM->pNext = NULL;
        return pM;
}

// insert a new message into the message queue
void APMSInsertMsg (
```

```
                TAPrealMsgSystem *      pMS,
                TAPMsg *                        pM
        ) {

        if (pMS->pNewRXMsg) {
                pMS->pNewRXMsg->pNext = pM;
        }
        pMS->pNewRXMsg = pM;
        if (!pMS->pOldRXMsg) {
                pMS->pOldRXMsg = pM;
        }
        pMS->messagecounter++;

}

// unchains a received message
void APMSunchainMessage (
                TAPrealMsgSystem *      pMS,
                TAPMsg *                        pM,
                TAPMsg *                        pAntecessorM
) {
        if (pAntecessorM) {
                pAntecessorM->pNext = pM->pNext;
        } else {
                pMS->pOldRXMsg = pM->pNext;
        }
        if (pM == pMS->pNewRXMsg) {
                pMS->pNewRXMsg = NULL;
        }

        // now there is one message less left
        pMS->messagecounter--;
}


// get oldest message
TAPMsg * APMSgetMsg (
                TAPMsgSystem            ms,                     // the message system
                TAPMessageID            msgID,          // if 0 all messages are allowed
                TAPNodeID                       sender,                 // if 0 all senders are allowed
                uint32_t                        mNumber,                // if 0 all numbers are allowed
                int                                     ackMsgAllowed
        ) {

        TAPrealMsgSystem * pMS = (TAPrealMsgSystem *) ms;
```

```
        // flags
        int senderOK;
        int msgIDok;
        int numberOK;
        // temp vars
        TAPMsg * pM;
        TAPMsg * pAntecessorM;
        uint32_t * pH;

        // result var
        TAPMsg * res = NULL;

checkMessages:
        // search msg list
        pM = pMS->pOldRXMsg;
        pAntecessorM =  NULL;

        if (!pM) goto waitForMessage;


        senderOK = 0;
        msgIDok = 0;
        numberOK = 0;

        pH = *(pM->pH);

        if (!sender) {
                senderOK = 1;
        } else {
                if (pH[eAPMsgHeaderPosition_sender] ==
                   sender) senderOK = 1;
        }
        if (!msgID) {
                // filter ack/nack msg
                if (ackMsgAllowed) {
                        msgIDok = 1;
                } else {
                        if (
                                        (pH[
                                           eAPMsgHeaderPosition_msgTypeI
                                           ] !=
                                           eAPMsgTypes_ACK
                                           )  &&
                                        (pH[
                                           eAPMsgHeaderPosition_msgTypeI
                                           ] !=
                                           eAPMsgTypes_NACK
                                           )
                                ) {
                                msgIDok = 1;
                        }
                }
        } else {
                if (pH[eAPMsgHeaderPosition_msgTypeID]
                   == msgID) msgIDok = 1;
```

```
        }
        if (!mNumber) {
                numberOK = 1;
        } else {
                if (pH[eAPMsgHeaderPosition_msgNumber]
                   == mNumber) numberOK = 1;
        }
        if ((senderOK) && (msgIDok) && (numberOK)) {
                res = pM;
                goto exit;
        }
        pAntecessorM = pM;
        pM = pM->pNext;
        if (pM) goto checkMessages;
waitForMessage:
        drv_3_feedRecvStateM();
        goto checkMessages;
exit:
        // unchain message
        APMSunchainMessage(pMS,pM,pAntecessorM);
        // now one thread is less waiting for a message
        return res;
error:
        return NULL;
}

// wait till a new message has been received
void APMSwaitForNewMessage (TAPrealMsgSystem * pMS)
{
        volatile int mc;

        mc = pMS->messagecounter;

        while (mc == pMS->messagecounter) {
                drv_3_feedRecvStateM();
        }
}

// returns 0 if a new message is available
inline int APMSisMessageAvailble (TAPrealMsgSystem * pMS
   ) {
        return (!pMS->pOldRXMsg) ? 0 : -1;
}

// ========================================
// the receive state machine
// ========================================

// the receive state machine state function for
   receiving the msg header
int SMinitial (
                void *                  pVoidSM,
                           // pointer to the
                   statemachine
                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;
```

```
        // copy message header
        pSM->pD = (uint32_t *) pSM->pMsg->pH;
        for (i = 0; i < eAPMsgHeaderPosition_headerElementNumber;i++) {
                *pSM->pD = pSM->header[i];
                pSM->pD++;
        }
        pSM->elementsLeft = pSM->header[eAPMsgHeaderPosition_length];
        // set up the data
        // 1. check if there is an data element
        if (!pSM->elementsLeft) {
                // no! now finish the message
                return SMmessageFinished(pVoidSM);
        }
        // 2. yes
        // 2.1 setup the sm for the data receiving
        pSM->state = SMdata;
        // 2.2 now check if we have to copy some data
        number -= copyAmount;
        if (number) {
                // set the data pointer
                pD += copyAmount;
                // and copy the data
                return SMdata (pVoidSM,pD,number);
        }
        return 0;
}


// the receive state machine state function for receiving the data
int SMdata (
                void *                  pVoidSM,        // pointer to the statemachine
                uint32_t *              pD,             // pointer to the data
                int                             number          // the number of data elements
        ) {
        TAPReceiveStateMachine * pSM = (TAPReceiveStateMachine *) pVoidSM;
        int                     copyAmount = number;
        int                             i;
        // 1. transfer the data
        // do some clipping
        if (pSM->elementsLeft < copyAmount) copyAmount = pSM->elementsLeft;
        // copy
```

```
		for (i = 0; i < copyAmount;i++) {
				*pSM->pD = *pD;
				pSM->pD++;
				pD++;
		}
		// set statemachine work data
		pSM->elementsLeft -= copyAmount;
		// check if we have to change the statemachine
		if (pSM->elementsLeft) return 0;
		int res = SMmessageFinished (pVoidSM);
		if (res) return res;

		// check if there some bytes left to copy
		number -= copyAmount;
		if (number) {
				// set the data pointer
				pD += copyAmount;
				// and copy the data
				return pSM->state (pVoidSM,pD,number);
		}
		return 0;

}

// this function is called when all data have been received
int SMmessageFinished (
				void *				pVoidSM		// pointer to the statemachine
		) {
		TAPReceiveStateMachine * pSM = (TAPReceiveStateMachine *) pVoidSM;
		TAPMsg *				pM;
		// 1. reset SM
		// set the helper
		pSM->elementsLeft = eAPMsgHeaderPosition_headerElementNumber;
		pSM->pD = pSM->header;

		// data
		pM = pSM->pMsg; // save msg info for inserting
		pSM->pMsg = NULL;

		// right state function
		pSM->state = SMinitial;			// the state

		// 2. insert message at the message system
		APMSInsertMsg ((TAPrealMsgSystem *)pSM->pMS,pM);
		return 0;
}


// inits the state machine
void APInitReceiveStateMachine (
```

```
                TAPReceiveStateMachine *          pSM,    // pointer to the state machine
                TAPMsgSystem                      pMS,    // pointer to the message system
                const TAPMsgDrv  *                pDrv    // the driver associated with the statemachine
        ) {
        pSM->state = SMinitial;
        pSM->pMS = pMS;
        pSM->pDrv = pDrv;
        // set the helper
        pSM->elementsLeft = eAPMsgHeaderPosition_headerElementNumber;
        pSM->pD = pSM->header;

        // data
        pSM->pMsg = NULL;
}

int APHandleMsg (
                TAP *             pAP,
                TAPMsg *          pM
        ) {

        TAPMessageID                          msgID;
        const THALMsgProcessMessageAssociation * pMsgIDandFunctAsso;
        int                                                       i;

        // get message id
        msgID = (*(pM->pH))[eAPMsgHeaderPosition_msgTypeID];
        // search handler
        pMsgIDandFunctAsso = gHALMsgProcessRXHandlers.pL;
        for (i = 0; i < gHALMsgProcessRXHandlers.number; i++) {
                if (((TAPMessageID)pMsgIDandFunctAsso->msgID) == msgID) {
                        return pMsgIDandFunctAsso->pfktHandle(pAP,pM);
                }
                pMsgIDandFunctAsso++;
        }
        return -1;
}

void APMessageProcessingThread (TAP * pAP) {
        TAPrealMsgSystem *                pMS = (TAPrealMsgSystem *) pAP->MS;
```

```
        TAPMsg *                                pM;
        TAPNodeID                               recv;
        while (1) {
                // get the message
                pM = APMSgetMsg (pMS,0,0,0,0);
                if (!pM) goto error;
                // search the message handler
                recv = (*(pM->pH))[
                   eAPMsgHeaderPosition_receiver];
                if ((recv == dAPNodeID_ALL) || (recv ==
                   pAP->nodeID)) {
                        if(APHandleMsg (pAP,pM)) goto
                           exit;
                }
                // free memory
                APMSdeleteMsg (pAP->MS,pM);
        }
exit:
        // free memory
        APMSdeleteMsg (pAP->MS,pM);
error:
        return;
}

// if a message is in the queue available it will be
   processed
void APMessageProcess (TAP * pAP) {
        TAPrealMsgSystem *              pMS = (
           TAPrealMsgSystem *) pAP->MS;
        TAPMsg *                                pM;
        TAPNodeID                               recvID;

        // search msg list
        pM = pMS->pOldRXMsg;

        // if there is no message we will return
        if (!pM) return;

        // unchain first message
        APMSunchainMessage(pMS, pM, NULL);

        // check header
        recvID = (*(pM->pH))[
           eAPMsgHeaderPosition_receiver];
        if ((recvID == dAPNodeID_ALL) || (recvID == pAP
           ->nodeID)) {
                APHandleMsg (pAP,pM);
        }
        // free memory
        APMSdeleteMsg (pAP->MS,pM);
}
```

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:

```
// ================================
// AP uuid = 10
// ================================


// inits the AP
int APinit (
                        TAP *
                           pAP,
                        TAPNodeID
                           nodeID,
                        const TAPMsgDrv *         pDrvList
                           ,
                        const int
                           driverNumber,
                        size_t
                           messagePoolSize,
                        int
                                   sysEndian
                )
{
        gAPendianFlag = sysEndian;

        pAP->nodeID = nodeID;
        pAP->pNodeList = NULL;
        pAP->pDrvList = pDrvList;
        pAP->driverNumber = driverNumber;
        pAP->msgSysMMU = AP_MMU_create(messagePoolSize);
        pAP->IP = APInterpreterCreate(pAP);
        pAP->MS = APMScreate (pAP->msgSysMMU,sysEndian);
        pAP->msgNumber = 0;
        pAP->APstate = eAPstate_idle;

        if(
                (!pAP->msgSysMMU)||
                (!pAP->IP)||
                (!pAP->MS)
                ) return -1;
        // init drv
        TAPMsgDrv * pDrv = (TAPMsgDrv *)pDrvList;
        int i;
        for (i = 0; i < driverNumber; i++) {
                pDrv->pfkt_open(pAP, pDrv);
```

```
        pDrv++;
    }

    // login the ap to the message system
    return TX_login(pAP);
}

// deletes the AP
void APdelete (TAP * pAP)
{
    // logout form all other devices
    TX_logout(pAP);
    // close & destroy drv
    TAPMsgDrv * pDrv = (TAPMsgDrv *)pAP->pDrvList;
    int i;
    for (i = 0; i < pAP->driverNumber; i++) {
        pDrv->pfkt_close(pDrv);
        pDrv->pfkt_destroy(pDrv);
        pDrv++;
    }

    APMSdelete (pAP->MS);
    APInterpreterDelete(pAP->IP);
    AP_MMU_delete(pAP->msgSysMMU);
}

// find a node at the list
TAPNode * APfindNode(TAP * pAP, TAPNodeID nodeID) {
    TAPNode * pN = pAP->pNodeList;
    while (pN) {
        if (pN->nodeID == nodeID) return pN;
        pN = pN->pNext;
    };
    return NULL;
}

// adds a new node to the node list
int APaddNode(TAP * pAP, TAPNodeID newNodeID, const TAPMsgDrv * pDrv) {
    if (APfindNode(pAP,newNodeID)) return 1;
    TAPNode * pN = (TAPNode *) malloc(sizeof(TAPNode));
    if (!pN) return -1;
    pN->nodeID = newNodeID;
    pN->pDrv = pDrv;
    pN->pNext = pAP->pNodeList;
    pAP->pNodeList = pN;
    return 0;
}

// removes a node from the node list
void APremoveNode(TAP * pAP, TAPNodeID nodeID){
    TAPNode * pAntN = pAP->pNodeList; // antecessor node
    TAPNode * pActN = pAP->pNodeList; // actual node
```

```c

        while (pActN) {
                // compare node id's
                if (pActN->nodeID == nodeID) {
                        // unchain

                        // check if we at the first position at the list
                        if (pAP->pNodeList == pAntN) {
                                // reset the pointer
                                pAP->pNodeList = pActN->pNext;
                        } else {
                                // set the antecessor
                                pAntN->pNext = pActN->pNext;
                        }
                        // free node
                        free(pActN);
                        // and abort
                        return;
                }
                // the actual element becomes the precessor element
                pAntN = pActN;
                pActN = pActN->pNext;
        }
}

// get a new message number
unsigned int APgetNewMessageNumber (TAP *pAP) {
        pAP->msgNumber++;
        return pAP->msgNumber;
}

// find the driver associated with der nodeID
const TAPMsgDrv * APfindDrvBySenderID (TAP * pAP, TAPNodeID node) {
        TAPNode * pN = pAP->pNodeList;
        while (pN) {
                if (pN->nodeID == node) {
                        return pN->pDrv;
                }
                pN = pN->pNext;
        }
        return NULL;
}

// runs the AP
int APrun(TAP *pAP) {
        pAP->APstate = eAPstate_run;
        return 0;
}

typedef struct SAPrealMMUMemory {
```

```
        uint32_t * pData;                   // the data
        int count;                          // amount of data elements
        struct SAPrealMMUMemory *       pNext;      // next element
        struct SAPrealMMUMemory *       pPrev;      // previous element
} TAPrealMMUMemory;

//the mmu type
typedef struct SAPrealMMU {
        uint32_t *                          memory;     // the memory block
        TAPrealMMUMemory *              pStart;     // first element
        TAPrealMMUMemory *              pEnd;       // second element
        TAPrealMMUMemory *              pUnusedList;        // list with the unused elements
        uint32_t  *                     pUnusedData;        // pointer to the unused memory
        int elementsAvailable;      // amount of elements witch are available without using the garbage collector
        int totalAvailable;         // total amount of free bytes

} TAPrealMMU;

// ========================================
// memory entry functions
// ========================================

// a little macro for unchaining an element
#define DMemoryEntryUnchain(pM) \
        if (pM->pNext) pM->pNext->pPrev = pM->pPrev; \
        if (pM->pPrev) pM->pPrev->pNext = pM->pNext


//creates an memory entry
TAPrealMMUMemory * MemoryEntry_create () {
        TAPrealMMUMemory * pM = NULL;

        pM = (TAPrealMMUMemory *) malloc(sizeof(TAPrealMMUMemory));
        if (!pM) return NULL;
        pM->pData = NULL;
        pM->count = 0;
        pM->pNext = NULL;
        pM->pPrev = NULL;

```

```
        return pM;
}

//deletes an memory Entry
void MemoryEntry_delete (
                TAPrealMMUMemory * pM   // the memory to delete
                )
{
        // put the entry out of the chain
        DMemoryEntryUnchain(pM);
        // now we delete it
        free(pM);
}

// ========================================
// mmu helper
// ========================================

//alloc if needed a new memory entry
TAPrealMMUMemory * MMU_helper_createMemoryEntry (
                TAPrealMMU *    pMMU            // MMU structure to init
                )
{
        // check if we have to alloc a new memory entry
        if (!pMMU->pUnusedList) return MemoryEntry_create();
        // no there is some left at the list
        TAPrealMMUMemory * pM;
        // take the first one
        pM = pMMU->pUnusedList;
        // reset the list
        pMMU->pUnusedList = pM->pNext;
        // now unchain the element (for sure)
        DMemoryEntryUnchain(pM);
        // set the element pointers
        pM->pNext = NULL;
        pM->pPrev = NULL;
        return pM;
}

//the garbage collector
void MMU_helper_garbageCollector (
                TAPrealMMU *    pMMU            // MMU structure to init
                )
{
        TAPrealMMUMemory * pM = pMMU->pStart;
        uint32_t * pD = pMMU->memory;
        while (pM) {
                // check if we have to move the data
                if (pD != pM->pData) {
                        // move the data
                        memmove(pD,pM->pData,pM->count*
```

```
                            sizeof(uint32_t));
                }
                // reset the destination pointer
                pD += pM->count;
                pM = pM->pNext;
        }
        // compressing memory finished
        // set the mmu vars new
        pMMU->elementsAvailable = pMMU->totalAvailable;
        pMMU->pUnusedData = pD;
}



// create a mmu
TAPMMU AP_MMU_create (size_t elementsNumber) {
        TAPrealMMU * pMMU;

        pMMU = (TAPrealMMU *) malloc (sizeof(TAPrealMMU)
           );
        if (!pMMU) return NULL;


        pMMU->memory = (uint32_t *) malloc (
           elementsNumber*sizeof(uint32_t));
        pMMU->pUnusedData = pMMU->memory;

        // setup lists
        pMMU->pStart = NULL;
        pMMU->pEnd = NULL;
        pMMU->pUnusedList = NULL;

        pMMU->elementsAvailable =elementsNumber;

        pMMU->totalAvailable = elementsNumber;
        return pMMU;
}

// destroying the mmu
void AP_MMU_delete (TAPMMU mmu) {
        TAPrealMMU * pMMU = (TAPrealMMU *) mmu;

        TAPrealMMUMemory * pM;
        TAPrealMMUMemory * pMnext;

// 1. delete al mmu entry's
        // 1.1 unused entry
        pM = pMMU->pUnusedList;
        while (pM) {
                pMnext = pM->pNext;
                MemoryEntry_delete(pM);
                pM = pMnext;
        }
        pMMU->pUnusedList = NULL;
        // 1.2 used blocks
```

```c
        pM = pMMU->pStart;
        while (pM) {
                pMnext = pM->pNext;
                MemoryEntry_delete(pM);
                pM = pMnext;
        }
        pMMU->pStart = NULL;
        pMMU->pEnd = NULL;
// 2. delete mmu memory
        free (pMMU->memory);
}

// getting memmory from the mmu
TAPMMUmemmory AP_MMU_get (TAPMMU mmu, size_t elements) {
        TAPrealMMU * pMMU = (TAPrealMMU *) mmu;

        // check if there is enough space
        if (pMMU->totalAvailable < elements) goto error;
        // check if we have to use the garbage collector
        if (pMMU->elementsAvailable < elements) {
                // start garbage collector
                MMU_helper_garbageCollector(pMMU);
        }
        // we have enough memory so let's allocate some

        // get a new entry
        TAPrealMMUMemory * pM;
        pM = MMU_helper_createMemoryEntry(pMMU);
        if (!pM) return NULL;
        // get some memory
        pM->pData = pMMU->pUnusedData;
        pM->count = elements;
        // refresh data
        pMMU->pUnusedData += elements;
        pMMU->totalAvailable -= elements;
        pMMU->elementsAvailable -= elements;
        // insert memory element at the end of the list and update last element
        pM->pPrev = pMMU->pEnd;
        if (pMMU->pEnd) pMMU->pEnd->pNext = pM;
        if (!pMMU->pStart) pMMU->pStart = pM;
        pMMU->pEnd = pM;
        return pM;
error:
        return NULL;
}

// free memmory from the mmu
void AP_MMU_free (TAPMMU mmu, TAPMMUmemmory memory) {
        TAPrealMMU * pMMU = (TAPrealMMU *) mmu;
        TAPrealMMUMemory * pM = (TAPrealMMUMemory *) memory;

        if (!pM) return;
        // set mmu settings
```

```
	if (pMMU->pStart == pM) pMMU->pStart = pM->pNext;
	if (pMMU->pEnd == pM) pMMU->pEnd = pM->pPrev;
	// unchain element
	DMemoryEntryUnchain (pM);
	// and put it to the chain of unused
	pM->pNext = pMMU->pUnusedList;
	pM->pPrev = NULL;
	if (pMMU->pUnusedList) {
		pMMU->pUnusedList->pPrev = pM;
	}
	pMMU->pUnusedList = pM;
	// now set the mmu data new
	pMMU->totalAvailable += pM->count;
}

// getting access to the MMU data
void * AP_MMU_getData (TAPMMUmemmory memory) {
	TAPrealMMUMemory * pM = (TAPrealMMUMemory *) memory;
	return pM->pData;
}

// ========================================
// the AP interpreter (thread save)
// ========================================

// the real interpreter
typedef struct SAPrealInterpreter {
	TAP *					pAP;			// pointer to the audio processor
	int					state;			// state of the IP
	int					nextState;		// the next state of the IP
	TAPInterpreterCPU			cpu;			// the IP core
	TAPInterpreterFuncCall *		code;			// the code
	int32_t					instructionCount;	// number of instructions at the code
	TAPInterpreterVariable *		variables;		// the variables
	int32_t					variableCount;		// number of the variables
	pthread_mutex_t				gM;			// a guarding mutex
} TAPrealInterpreter;
```

```
// create a new interpreter
TAPInterpreter APInterpreterCreate (void * pAP) {
        TAPrealInterpreter * pIP = NULL;
        pIP = (TAPrealInterpreter *) malloc (sizeof(
            TAPrealInterpreter));
        if (!pIP) return NULL;

        pIP->pAP = pAP;
        pIP->state = eAPInterpreterState_idle;
        pIP->nextState = eAPInterpreterState_idle;
        pIP->cpu.IP = pIP;
        pIP->cpu.CF = 0;
        pIP->cpu.EF = 0;
        pIP->cpu.pCodeStart = NULL;
        pIP->cpu.pCodeEnd = NULL;
        pIP->cpu.pIP = NULL;

        pIP->code = NULL;
        pIP->instructionCount = 0;

        pIP->variables = NULL;
        pIP->variableCount = 0;

        pIP->gM = PTHREAD_MUTEX_INITIALIZER;

        return pIP;
}

// cleans the interpreter
void APInterpreterClean (TAPInterpreter IP) {
        TAPrealInterpreter * pIP = (TAPrealInterpreter
            *) IP;

        // clean code
        if (pIP->code) {
                free (pIP->code);
                pIP->code = NULL;
        }
        pIP->instructionCount = 0;

        // clean variables
        TAPInterpreterVariable * pV = pIP->variables;
        int i;
        for (i = 0; i < pIP->variableCount; i++) {
                if (pV->pVI) pV->pVI->pFkt_delete(pV->
                    pData);
                pV++;
        }
        if (pIP->variables) {
                free (pIP->variables);
                pIP->variables = NULL;
        }
        pIP->variableCount = 0;

```

```
}

// deletes the interpreter
void APInterpreterDelete (TAPInterpreter IP) {
        TAPrealInterpreter * pIP = (TAPrealInterpreter *) IP;
        APInterpreterClean(IP);
        free (pIP);
}

int APInterpreterStateRun(TAPInterpreter IP) {
        TAPrealInterpreter * pIP = (TAPrealInterpreter *) IP;
        TAPInterpreterFuncCall * pFC;

        // setup cpu
        pIP->cpu.CF = 0;
        pIP->cpu.EF = 0;
        pIP->cpu.pIP = pIP->code;
        pIP->cpu.pCodeStart = pIP->code;
        pIP->cpu.pCodeEnd = pIP->code + pIP->instructionCount;

        // run code
        while (eAPInterpreterState_run == pIP->state) {

                pthread_mutex_lock(&pIP->gM);

                pFC = pIP->cpu.pIP;
                // check if we reached the end of the code
                if (pFC > pIP->cpu.pCodeEnd) {
                        pthread_mutex_unlock(&pIP->gM);
                        return 0;
                }
                // execute command
                pFC->pHALFkt (&(pIP->cpu), pFC->param);
                // check error flags
                if (pIP->cpu.EF) {
                        pthread_mutex_unlock(&pIP->gM);
                        return -1;
                }
                pthread_mutex_unlock(&pIP->gM);
        }
        return 1;
}


// process the actual state
int APInterpreterProcessState(TAPInterpreter IP){
        TAPrealInterpreter * pIP = (TAPrealInterpreter *) IP;
        pthread_mutex_lock(&pIP->gM);
        pIP->state = pIP->nextState;
        int rc = 0;
```

```
	switch (pIP->state) {
		case eAPInterpreterState_idle:
			break;
		case eAPInterpreterState_loadProgramm:
			break;
		case eAPInterpreterState_run:
			pthread_mutex_unlock(&pIP->gM);
			rc = APInterpreterStateRun(IP);
			pthread_mutex_lock(&pIP->gM);
			if (rc >= 0) pIP->state = eAPInterpreterState_idle;
			break;
		case eAPInterpreterState_oneStep:
			break;
		case eAPInterpreterState_halt:
			break;
		default:
			pthread_mutex_unlock(&pIP->gM);
			return -10;
	}
	pthread_mutex_unlock(&pIP->gM);
	return rc;
}


// set interpreter state
int APInterpreterSetState (TAPInterpreter IP, int msgEndian, int32_t state) {
	TAPrealInterpreter * pIP = (TAPrealInterpreter *) IP;
	pthread_mutex_lock(&pIP->gM);
	APendianConversation32Bit((uint32_t *)&state, msgEndian);
	pIP->nextState = (int) state;
	pthread_mutex_unlock(&pIP->gM);
	return 0;
}

// get interpreter state
int32_t APInterpreterGetState (TAPInterpreter IP) {
	TAPrealInterpreter * pIP = (TAPrealInterpreter *) IP;
	return (int32_t) pIP->state;
}

// setup the interpreter for a new program
int APInterpreterInitNewProgramm (TAPInterpreter IP, int msgEndian, int32_t instructionsNumber, int32_t VariableNumber) {
	TAPrealInterpreter * pIP = (TAPrealInterpreter *) IP;
	int i;

	pthread_mutex_lock(&pIP->gM);
```

```
        APInterpreterClean (IP);

        APendianConversation32Bit((uint32_t *)&
           instructionsNumber, msgEndian);
        APendianConversation32Bit((uint32_t *)&
           VariableNumber, msgEndian);

        pIP->code = (TAPInterpreterFuncCall *) malloc(
           sizeof(TAPInterpreterFuncCall)*
           instructionsNumber);
        pIP->instructionCount = instructionsNumber;

        pIP->variables = (TAPInterpreterVariable *)
           malloc(sizeof(TAPInterpreterVariable) * (
           VariableNumber));
        for (i = 0; i < VariableNumber;i++) {
                pIP->variables[i].pData = NULL;
                pIP->variables[i].pVI = NULL;
        }
        pIP->variableCount = VariableNumber;

        pthread_mutex_unlock(&pIP->gM);

        return 0;
}

// load a variable/~array to an index
int APInterpreterLoadVar (TAPInterpreter IP, int
   msgEndian, int32_t index, int32_t varTypeID, int32_t
   numberOfElements)
{
        TAPrealInterpreter * pIP = (TAPrealInterpreter
           *) IP;

        pthread_mutex_lock(&pIP->gM);

        APendianConversation32Bit((uint32_t *)&index,
           msgEndian);
        APendianConversation32Bit((uint32_t *)&varTypeID
           ,msgEndian);
        APendianConversation32Bit((uint32_t *)&
           numberOfElements, msgEndian);


        if ((index < 0) || (index > pIP->variableCount))
            return -1;

        // set pointer to the runtime variable
        TAPInterpreterVariable * pRTV = pIP->variables +
            index;
        THAL_Variable const * pV = HALfindVar(varTypeID)
           ;
        if (!pV) {
                pthread_mutex_unlock(&pIP->gM);
```

```
		return -2;
	}

	pRTV->pData = pV->pFkt_create((unsigned int)numberOfElements);
	//if (!pRTV->pData) return -3;

	pRTV->pVI = pV;
	pthread_mutex_unlock(&pIP->gM);
	return 0;
}

// load a single Instruction to an index
int APInterpreterLoadInstr (TAPInterpreter IP,int msgEndian, int32_t index, int32_t * pRawInstr)
{
	TAPrealInterpreter * pIP = (TAPrealInterpreter *) IP;

	pthread_mutex_lock(&pIP->gM);

	APendianConversation32Bit((uint32_t *)&index,msgEndian);

	if ((index < 0) || (index > pIP->instructionCount)){
		pthread_mutex_unlock(&pIP->gM);
		return -1;
	}
	TAPInterpreterFuncCall * pIFC = pIP->code + index;
	memset (pIFC, 0, sizeof(TAPInterpreterFuncCall));

	// get function
	int32_t fid = *pRawInstr;
	APendianConversation32Bit((uint32_t *)&fid,msgEndian);

	THALFunction const * pF = HALfindFunction(fid);
	if (!pF) {
		pthread_mutex_unlock(&pIP->gM);
		return -2;
	}
	pIFC->pHALFkt = pF->pfktHAL;

	// convert parameters
	pRawInstr++; // set to the first parameter
	int i;
	THALFunctionParam const * pP = pF->paramList.pL;
	TuAPInterpreterFunctionParameter * pIFP = pIFC->param;
	for (i = 0; i < pF->paramList.number; i++) {
		if (APconvertRawParamData (msgEndian,pRawInstr,pP,pIFP,pIP->variables)) {
```

```
                                pthread_mutex_unlock(&pIP->gM);
                                return -3;
                        }
                        pP++;
                        pRawInstr++;
                        pIFP++;
                }
                pthread_mutex_unlock(&pIP->gM);
                return 0;
}

// gets the varaible by it's index
TAPInterpreterVariable * APInterpreterGetVariableByIndex (TAPInterpreter IP, int index) {
        return &(((TAPrealInterpreter *) IP)->variables[index]);
}

// gets the AP from the IP
void * APInterpreterGetAPfromIP (TAPInterpreter IP) {
        return ((TAPrealInterpreter *) IP)->pAP;
}

// gets the CPU from the IP
TAPInterpreterCPU * APInterpreterGetCPUref (TAPInterpreter IP) {
        return &(((TAPrealInterpreter *) IP)->cpu);
}

// ========================================
// the AP message system (thread save)
// ========================================

// ========================================
// the AP message system (thread save)
// ========================================

typedef struct SAPrealMsgSystem {
        TAPMsg *                                pOldRXMsg;              // pointer to the oldest received messages
        TAPMsg *                                pNewRXMsg;              // pointer to the newest received messages
        TAPMMU                                  mmu;                    // the mmu
        int                                     sysEndianness;  // the system endianness
        int                                     messagecounter; // a counter for checkin if a new message has been received

        sem_t                                   waitSem;                // a semaphore to wait for a message
        int
```

```
                getMsgCounter;  // a counter incremented how many threads calling getMsg and are waiting
        pthread_mutex_t          gM;               // a guarding mutex
} TAPrealMsgSystem;


int SMinitial (
                void *                    pVoidSM,               // pointer to the statemachine
                uint32_t *                pD,                    // pointer to the data
                int                       number                 // the number of data elements
        );

int SMdata (
                void *                    pVoidSM,               // pointer to the statemachine
                uint32_t *                pD,                    // pointer to the data
                int                       number                 // the number of data elements
        );

int SMmessageFinished (
                void *                    pVoidSM                // pointer to the statemachine
        );



// create AP message system
TAPMsgSystem APMScreate (
                TAPMMU                    mmu,                   // the mmu
                int                       sysEndianness   // the system endianness
        ) {
        TAPrealMsgSystem * pMS = (TAPrealMsgSystem *) malloc (sizeof(TAPrealMsgSystem));
        if (!pMS) return NULL;
        pMS->mmu = mmu;
        pMS->sysEndianness = sysEndianness;
        pMS->pOldRXMsg = NULL;
        pMS->pNewRXMsg = NULL;
        pMS->messagecounter = 0;

```

```c
        pMS->gM = PTHREAD_MUTEX_INITIALIZER;
        sem_init (&pMS->waitSem,0,0);
        pMS->getMsgCounter = 0;

        return pMS;
}

void APMSdelete (
        TAPMsgSystem ms
        ) {
        TAPrealMsgSystem * pMS = (TAPrealMsgSystem *) ms
           ;
        sem_destroy(&pMS->waitSem);
        free (pMS);
}


// frees a message from the message system
void APMSdeleteMsg (
        TAPMsgSystem    ms,
        TAPMsg *                pM
        ) {
        TAPrealMsgSystem * pMS = (TAPrealMsgSystem *) ms
           ;
        pthread_mutex_lock(&pMS->gM);

        AP_MMU_free(pMS->mmu,pM->memory);

        pthread_mutex_unlock(&pMS->gM);
}

// get memory for a new message
TAPMsg * APMSgetNewMsg (
                TAPrealMsgSystem *      pMsgSys,
                int
                   dataElementsNumber,
                const TAPMsgDrv *       pDrv
                ) {
        pthread_mutex_lock(&pMsgSys->gM);

        // we go for shure that we get enough memory
        // if mod(sizeof(TAPMsg)/sizeof(uint32_t)) != 0
           we need one uint32_t more -> +1
        TAPMMUmemmory m = AP_MMU_get (pMsgSys->mmu,
           sizeof(TAPMsg)/sizeof(uint32_t) + 1+
           eAPMsgHeaderPosition_headerElementNumber +
           dataElementsNumber);
        if (!m) return NULL;

        // set the pointers
        uint32_t * pRD = (uint32_t *) AP_MMU_getData(m);
        TAPMsg * pM = (TAPMsg *) pRD;
        pM->memory = m;
        pM->extraData.pDrv = pDrv;
        pM->pH = (TAPMsgHeader *)((uint32_t *) pRD +
```

```
            sizeof(TAPMsg)/sizeof(uint32_t)+1);
        pM->pData = (uint32_t *)pM->pH + sizeof(TAPMsgHeader)/sizeof(uint32_t);
        pM->pNext = NULL;

        pthread_mutex_unlock(&pMsgSys->gM);
        return pM;
}

// insert a new message into the message queue
void APMSInsertMsg (
                TAPrealMsgSystem *       pMS,
                TAPMsg *                         pM
        ) {
        pthread_mutex_lock(&pMS->gM);

        if (pMS->pNewRXMsg) {
                pMS->pNewRXMsg->pNext = pM;
        }
        pMS->pNewRXMsg = pM;
        if (!pMS->pOldRXMsg) {
                pMS->pOldRXMsg = pM;
        }
        pMS->messagecounter++;

        pthread_mutex_unlock(&pMS->gM);
        sem_post(&pMS->waitSem);
}

// get oldest message
TAPMsg * APMSgetMsg (
                TAPMsgSystem             ms,             // the message system
                TAPMessageID             msgID,          // if 0 all messages are allowed
                TAPNodeID                        sender,         // if 0 all senders are allowed
                uint32_t                         mNumber,        // if 0 all numbers are allowed
                int                              ackMsgAllowed
        ) {

        TAPrealMsgSystem * pMS = (TAPrealMsgSystem *) ms;
        // flags
        int senderOK;
        int msgIDok;
        int numberOK;
        // result var
        TAPMsg * res = NULL;

```

```
        pthread_mutex_lock(&pMS->gM);
        pMS->getMsgCounter++;
        pthread_mutex_unlock(&pMS->gM);

checkMessages:
        pthread_mutex_lock(&pMS->gM);

        // search msg list
        TAPMsg * pM = pMS->pOldRXMsg;
        TAPMsg * pAntecessorM =  NULL;
        uint32_t * pH;

        if (!pM) goto waitForMessage;


        senderOK = 0;
        msgIDok = 0;
        numberOK = 0;

        pH = *(pM->pH);

        if (!sender) {
                senderOK = 1;
        } else {
                if (pH[eAPMsgHeaderPosition_sender] ==
                   sender) senderOK = 1;
        }
        if (!msgID) {
                // filter ack/nack msg
                if (ackMsgAllowed) {
                        msgIDok = 1;
                } else {
                        if (
                                                (pH[
                                                   eAPMsgHeaderPosition_msgTypeI
                                                   ] !=
                                                   eAPMsgTypes_ACK
                                                   )  &&
                                                (pH[
                                                   eAPMsgHeaderPosition_msgTypeI
                                                   ] !=
                                                   eAPMsgTypes_NACK
                                                   )
                                ) {
                                msgIDok = 1;
                        }
                }
        } else {
                if (pH[eAPMsgHeaderPosition_msgTypeID]
                   == msgID) msgIDok = 1;
        }
        if (!mNumber) {
                numberOK = 1;
        } else {
                if (pH[eAPMsgHeaderPosition_msgNumber]
```

```
                        == mNumber) numberOK = 1;
            }
            if ((senderOK) && (msgIDok) && (numberOK)) {
                    res = pM;
                    goto exit;
            }
            pAntecessorM = pM;
            pM = pM->pNext;
            if (pM) goto checkMessages;
waitForMessage:
            pthread_mutex_unlock(&pMS->gM);
            if (sem_wait(&(pMS->waitSem)) == -1) goto error;

            // if other threads are waiting for a message
               give the sign to them
            pthread_mutex_lock(&pMS->gM);
            if (pMS->getMsgCounter > 1) {
                    if (sem_post(&pMS->waitSem) == -1) goto
                       error;
            }
            pthread_mutex_unlock(&pMS->gM);
            goto checkMessages;

exit:
            if (pAntecessorM) {
                    pAntecessorM->pNext = pM->pNext;
            } else {
                    pMS->pOldRXMsg = pM->pNext;
            }
            if (pM == pMS->pNewRXMsg) {
                    pMS->pNewRXMsg = NULL;
            }

            // now there is one message less left
            pMS->messagecounter--;
            // now one thread is less waiting for a message
            pMS->getMsgCounter--;
            pthread_mutex_unlock(&pMS->gM);
            return res;
error:
            pthread_mutex_unlock(&pMS->gM);
            return NULL;
}

// ==========================================
// the receive state machine
// ==========================================

// the receive state machine state function for
   receiving the msg header
int SMinitial (
                    void *                  pVoidSM,
                                // pointer to the
                       statemachine
                    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;
```

```
        for (i = 0; i < eAPMsgHeaderPosition_headerElementNumber;i++) {
                *pSM->pD = pSM->header[i];
                pSM->pD++;
        }
        pSM->elementsLeft = pSM->header[eAPMsgHeaderPosition_length];
        // set up the data
        // 1. check if there is an data element
        if (!pSM->elementsLeft) {
                // no! now finish the message
                return SMmessageFinished(pVoidSM);
        }
        // 2. yes
        // 2.1 setup the sm for the data receiving
        pSM->state = SMdata;
        // 2.2 now check if we have to copy some data
        number -= copyAmount;
        if (number) {
                // set the data pointer
                pD += copyAmount;
                // and copy the data
                return SMdata (pVoidSM,pD,number);
        }
        return 0;
}


// the receive state machine state function for receiving the data
int SMdata (
                void *                  pVoidSM,        // pointer to the statemachine
                uint32_t *              pD,             // pointer to the data
                int                     number          // the number of data elements
        ) {
        TAPReceiveStateMachine * pSM = (TAPReceiveStateMachine *) pVoidSM;
        int                     copyAmount = number;
        int                     i;
        // 1. transfer the data
        // do some clipping
        if (pSM->elementsLeft < copyAmount) copyAmount = pSM->elementsLeft;
        // copy
        for (i = 0; i < copyAmount;i++) {
                *pSM->pD = *pD;
```

```
			pSM->pD++;
			pD++;
		}
		// set statemachine work data
		pSM->elementsLeft -= copyAmount;
		// check if we have to change the statemachine
		if (pSM->elementsLeft) return 0;
		int res = SMmessageFinished (pVoidSM);
		if (res) return res;

		// check if there some bytes left to copy
		number -= copyAmount;
		if (number) {
			// set the data pointer
			pD += copyAmount;
			// and copy the data
			return pSM->state (pVoidSM,pD,number);
		}
		return 0;

}

// this function is called when all data have been received
int SMmessageFinished (
			void *				pVoidSM		// pointer to the statemachine
		) {
		TAPReceiveStateMachine *	pSM = (TAPReceiveStateMachine *) pVoidSM;
		TAPMsg *				pM;
		// 1. reset SM
		// set the helper
		pSM->elementsLeft = eAPMsgHeaderPosition_headerElementNumber;
		pSM->pD = pSM->header;

		// data
		pM = pSM->pMsg; // save msg info for inserting
		pSM->pMsg = NULL;

		// right state function
		pSM->state = SMinitial;		// the state

		// 2. insert message at the message system
		APMSInsertMsg ((TAPrealMsgSystem *)pSM->pMS,pM);
		return 0;
}


// inits the state machine
void APInitReceiveStateMachine (
			TAPReceiveStateMachine *
				pSM,	// pointer to the
```

```
                    state machine
                TAPMsgSystem
                                    pMS,    // pointer to
                    the message system
                const TAPMsgDrv   *
                                    pDrv    // the driver
                    associated with the statemachine
        ) {
        pSM->state = SMinitial;
        pSM->pMS = pMS;
        pSM->pDrv = pDrv;
        // set the helper
        pSM->elementsLeft =
           eAPMsgHeaderPosition_headerElementNumber;
        pSM->pD = pSM->header;

        // data
        pSM->pMsg = NULL;
}

int APHandleMsg (
                TAP *           pAP,
                TAPMsg *        pM
        ) {

        TAPMessageID
                                    msgID;
        const THALMsgProcessMessageAssociation *
           pMsgIDandFunctAsso;
        int
                                                    i;

        // get message id
        msgID = (*(pM->pH))[
           eAPMsgHeaderPosition_msgTypeID];
        // search handler
        pMsgIDandFunctAsso = gHALMsgProcessRXHandlers.pL
           ;
        for (i = 0; i < gHALMsgProcessRXHandlers.number;
           i++) {
                if (((TAPMessageID)pMsgIDandFunctAsso->
                   msgID) == msgID) {
                        return pMsgIDandFunctAsso->
                           pfktHandle(pAP,pM);
                }
                pMsgIDandFunctAsso++;
        }
        return -1;
}

void APMessageProcessingThread (TAP * pAP) {

        TAPrealMsgSystem *              pMS = (
           TAPrealMsgSystem *) pAP->MS;
        TAPMsg *                            pM;
```

```
        TAPNodeID                               recv;
        while (1) {
                // get the message
                pM = APMSgetMsg (pMS,0,0,0,0);
                if (!pM) goto error;
                // search the message handler
                recv = (*(pM->pH))[
                   eAPMsgHeaderPosition_receiver];
                if ((recv == dAPNodeID_ALL) || (recv ==
                   pAP->nodeID)) {
                        if(APHandleMsg (pAP,pM)) goto
                           exit;
                }
                // free memory
                APMSdeleteMsg (pAP->MS,pM);
        }
exit:
        // free memory
        APMSdeleteMsg (pAP->MS,pM);
error:
        return;
}
```