void main(int argc, char \*argv[])

{

LARGE\_INTEGER t1, t2, tc;

FILE \*f\_speech; /\* File of speech data \*/

FILE \*f\_serial; /\* File of serial bits for transmission \*/

short speech16[L\_FRAME\_FSMAX \* 2];

short serial[NBITS\_MAX]; /\* serial parameters. \*/

unsigned char serialravs[NBITS\_MAX]; /\* serial parameters. \*/

float channel\_right[4 \* L\_FRAME\_FSMAX];

float channel\_left[2 \* L\_FRAME\_FSMAX];

float mem\_up\_right[2\*L\_FILT\_OVER\_FS], mem\_up\_left[2\*L\_FILT\_OVER\_FS];

int frac\_up\_right, frac\_up\_left;

int fac\_up, fac\_down, nb\_samp\_fs;

Coder\_State\_P \*st = NULL;

short numOfChannels, bitsPerSample;

void \*stravsEnc = NULL;

int nb\_bits = 0;

int i, lg, L\_frame;

long frame, samplingRate, dataSize;

int L\_next, L\_next\_st;

int nb\_samp, nb\_hold;

short /\*codec\_mode,\*/ mode, extension;

short serial\_size = 0;

char \*input\_filename;

char \*output\_filename;

int st\_mode, old\_st\_mode;

EncoderConfig conf;

short fst;

float rate;

char FileFormatType[25];

FILE \*f\_config = 0;

float rec\_time = 0.0;

float config\_file\_time = 0.0;

float t[4]; /\* time,conf.extension,conf.mode\_index,conf.fscale \*/

float old\_bitrate;

char \*config\_filename;

char \*status\_filename;

int Processed\_sample = 0;

int nb\_sample\_to\_process = 0;

/\*--------------------------------------------------------------\*

\* - aac variable \*

\*---------------------------------------------------------------\*/

AACENC\_CONFIG config;

AuChanInfo inputInfo;

hAudioChannel inputFile = NULL;

AuChanType auType = TYPE\_AUTODETECT ; /\* must be set \*/

AuChanMode auFlags = AU\_CHAN\_READ;

FILE \*fOut=NULL;

int bEncodeMono = 0;

int bitrate;

int nChannelsAAC, nChannelsSBR;

int sampleRateAAC;

int bandwidth = 0;

unsigned int numAncDataBytes=0;

unsigned int ancDataLength = 0;

int bDoIIR2Downsample = 0;

int bDingleRate = 0;

int useParametricStereo = 0;

int coreWriteOffset = 0;

int coreReadOffset = 0;

int envWriteOffset = 0;

int envReadOffset = 0;

int writeOffset=INPUT\_DELAY\*MAX\_CHANNELS;

struct AAC\_ENCODER \*aacEnc = 0;

int bDoUpsample = 0;

int upsampleReadOffset = 0;

int bDoIIR32Resample = 0;

const int nRuns = 4;

float \*resamplerScratch = sbr\_envRBuffer;

HANDLE\_SBR\_ENCODER hEnvEnc=NULL;

int nbyte\_ps=0;//xhl

#ifdef NEW\_SAD

SyndecidSt \*synSt = NULL;

syndecid\_init(&synSt);

#endif

/\* 初始化 \*/

f\_serial = NULL;

input\_filename = NULL;

output\_filename = NULL;

config\_filename = NULL;

/\* 版权信息 \*/

copyright();

/\* 命令行读入 \*/

parsecmdline(argc, argv, &input\_filename, &output\_filename, &config\_filename, &status\_filename, &conf, &rate);

/\* 打开输入wave文件 \*/

if ((f\_speech =

Wave\_fopen(input\_filename, "rb", &numOfChannels, &samplingRate,

&bitsPerSample, &dataSize)) == NULL)

{

fprintf(stderr, "Error opening the input file %s.\n", input\_filename);

exit(EXIT\_FAILURE);

}

/\* 设置状态变量 \*/

if (rate != -1.0)

{

if(conf.st\_mode == -2 || numOfChannels == 1)

{

GetRate(&conf, rate, MonoRate, 3\*18);

}

else

{

GetRate(&conf, rate, StereoRate, 3\*27);

}

}

/\* 检查是否为立体声输入 \*/

if ((conf.st\_mode >= 0) && (numOfChannels != 2))

{

fprintf(stderr, "Input file %s must be stereo\n", input\_filename);

exit(EXIT\_FAILURE);

}

/\* 检查是否为16位PCM \*/

if (bitsPerSample != 16)

{

fprintf(stderr, "Input file %s must be 16 bits encoded\n",

input\_filename);

exit(EXIT\_FAILURE);

}

if (conf.FileFormat == F3GP)

{

strcpy(FileFormatType,"3gp File Format");//设置生成码流为带3gpp头的3gpp文件格式

if(conf.bc == 1)

{

Create3GPAMRWB();

}

else

{

Create3GPAMRWBPlus();

}

}

else /\* raw data \*/

{

strcpy(FileFormatType,"Raw File Format");//设置生成码流为raw文件格式

if ((f\_serial = fopen(output\_filename, "wb")) == NULL)

{

fprintf(stderr, "Error opening output bitstream file %s.\n",output\_filename);

exit(EXIT\_FAILURE);

}

}

/\* 检查wb模式时输入文件采样率是否为16kHz \*/

if ((conf.extension == 0) && (samplingRate != 16000))

{

fprintf(stderr, "R-AVS work only at 16kHz\n");

exit(EXIT\_FAILURE);

}

/\* 设置采样率转换参数 \*/

fac\_up = fac\_down = 12; /\* no oversampling by default \*/

frac\_up\_right = 0;

frac\_up\_left = 0;

if (conf.fscale != 0)

{

switch (samplingRate) {

case 8000:

fac\_down = 2;

samplingRate = 48000;

break;

case 16000:

fac\_down = 4;

samplingRate = 48000;

break;

case 24000:

fac\_down = 6;

samplingRate = 48000;

break;

case 32000:

fac\_down = 8;

samplingRate = 48000;

break;

case 11025:

fac\_down = 3;

samplingRate = 44100;

break;

case 22050:

fac\_down = 6;

samplingRate = 44100;

break;

}

set\_zero(mem\_up\_right, 2\*L\_FILT\_OVER\_FS);

set\_zero(mem\_up\_left, 2\*L\_FILT\_OVER\_FS);

}

/\* 设置buff长度 \*/

set\_frame\_length(samplingRate, conf.fscale, &L\_frame, &L\_next, &L\_next\_st);

st = malloc(sizeof(Coder\_State\_P));

memset(serial, 0x42, NBITS\_MAX \* sizeof(short));

old\_bitrate = get\_bitrate(&conf);

fprintf(stderr, "%s\nEncoding @ %6.2fkbps",FileFormatType, get\_bitrate(&conf));

st->old\_bitrate = old\_bitrate;

/\* 初始化状态变量 \*/

init\_coder\_ravs\_p(st, (int) numOfChannels, conf.fscale,

conf.use\_case\_mode, 1);

stravsEnc = E\_IF\_init();

/\* 从文件读取每声道采样点 \*/

nb\_samp\_fs = ((L\_frame\*fac\_down)+frac\_up\_right) / fac\_up;

lg = read\_data(f\_speech, channel\_right, (numOfChannels \* nb\_samp\_fs));

if (lg != (numOfChannels \* nb\_samp\_fs))

{

printf("Error: file too short!\n");

exit(EXIT\_FAILURE);

}

/\* 采样率转换及分频 \*/

if (numOfChannels == 2)

{

deinterleave(channel\_right, channel\_left, channel\_right, nb\_samp\_fs);

over\_fs(channel\_left, channel\_left, L\_frame,

fac\_down, mem\_up\_left, &frac\_up\_left);

}

over\_fs(channel\_right, channel\_right, L\_frame,

fac\_down, mem\_up\_right, &frac\_up\_right);

/\* 编码初始化 \*/

if (conf.extension > 0)//avs-p10模式

{

nb\_samp =

coder\_ravs\_p\_first(channel\_right, channel\_left, numOfChannels,

L\_frame,

(numOfChannels == 1) ? L\_next : L\_next\_st,

conf.fscale, st);

}

else//wb模式

{

moveAndRound(channel\_right, speech16, 320);

E\_IF\_encode\_first(stravsEnc, speech16);

nb\_samp = 320;

}

fprintf(stderr, "\n --- Running ---\n");

mode = conf.mode;

extension = conf.extension;

old\_st\_mode = st\_mode = conf.st\_mode;

frame = 0;

fst = conf.fscale;

if (config\_filename != 0) {

if(fac\_down != 12)

{

fprintf(stderr, "Must have 16kHz (to switch r-avs -> avs+ (mode 10@13))\n"

"or 48kHz (avs+) input\n\n");

exit(EXIT\_FAILURE);

}

if( numOfChannels == 1)

{

fprintf(stderr, "\*\* Warning \*\*\nDo not switch to stereo... \nyou have mono input\n\n");

}

f\_config = fopen(config\_filename,"r");

if (!f\_config) {

fprintf(stderr, "Error opening config file %s.\n",config\_filename);

exit(EXIT\_FAILURE);

}

while (!get\_config(f\_config,t) && !feof(f\_config)) {

printf("%2.3f \n",t[0]);

}

config\_file\_time = t[0];

rec\_time = 0;

}

nb\_sample\_to\_process = dataSize\*numOfChannels - lg;

Processed\_sample += nb\_samp;

inputInfo.bitsPerSample = 16 ; /\* only relevant if valid == 1 \*/

inputInfo.sampleRate = 44100 ; /\* only relevant if valid == 1 \*/

inputInfo.nChannels = 2 ; /\* only relevant if valid == 1 \*/

inputInfo.nSamples = 0 ; /\* only relevant if valid == 1 \*/

inputInfo.isLittleEndian = 1;

inputInfo.fpScaleFactor = AACENC\_PCM\_LEVEL ; /\* must be set \*/

inputInfo.valid = 1 ; /\* must be set \*/

inputInfo.useWaveExt = 0;

bitrate=24000;

PTR\_INIT(1); FUNC(1);

AacInitDefaultConfig(&config);

BRANCH(1); MOVE(1);

nChannelsAAC = nChannelsSBR = bEncodeMono ? 1:inputInfo.nChannels;

ADD(3); LOGIC(3); BRANCH(1);

if ( (inputInfo.nChannels == 2) && (!bEncodeMono) && (bitrate >= 16000) && (bitrate < 36000) )

{

MOVE(1);

useParametricStereo = 1;

}

BRANCH(1);

if (useParametricStereo)

{

MOVE(2);

nChannelsAAC = 1;

nChannelsSBR = 2;

}

ADD(3); LOGIC(2); BRANCH(1);

if ( (inputInfo.sampleRate == 48000) && (nChannelsAAC == 2) && (bitrate < 24000)

#if 1

||

(inputInfo.sampleRate == 48000) && (nChannelsAAC == 1) && (bitrate < 12000)

#endif

) {

MOVE(1);

bDoIIR32Resample = 1;

}

ADD(1); BRANCH(1);

if (inputInfo.sampleRate == 16000) {

MOVE(3);

bDoUpsample = 1;

inputInfo.sampleRate = 32000;

bDingleRate = 1;

}

MOVE(1);

sampleRateAAC = inputInfo.sampleRate;

BRANCH(1);

if (bDoIIR32Resample)

{

MOVE(1);

sampleRateAAC = 32000;

}

MOVE(3);

config.bitRate = bitrate;

config.nChannelsIn=inputInfo.nChannels;

config.nChannelsOut=nChannelsAAC;

MOVE(1);

config.bandWidth=bandwidth;

{

sbrConfiguration sbrConfig;

MOVE(1);

envReadOffset = 0;

MOVE(1);

coreWriteOffset = 0;

BRANCH(1);

if(useParametricStereo)

{

MOVE(3);

envReadOffset = (MAX\_DS\_FILTER\_DELAY + INPUT\_DELAY)\*MAX\_CHANNELS;

coreWriteOffset = CORE\_INPUT\_OFFSET\_PS;

writeOffset = envReadOffset;

}

sampleRateAAC = 24000;

PTR\_INIT(1); FUNC(1);

InitializeSbrDefaults (&sbrConfig);

MOVE(1);

sbrConfig.usePs = useParametricStereo;

PTR\_INIT(1); FUNC(6);

AdjustSbrSettings(&sbrConfig,

bitrate,

nChannelsAAC,

sampleRateAAC,

AACENC\_TRANS\_FAC,

24000);

PTR\_INIT(3); ADD(1); FUNC(4);

EnvOpen (&hEnvEnc,

inputBuffer + coreWriteOffset,

&sbrConfig,

&config.bandWidth);

}

while (Processed\_sample < nb\_sample\_to\_process || frame == 0)

{

fprintf(stderr, " Frames processed: %ld \r", frame);

frame++;

if (f\_config)

{

if (rec\_time >= config\_file\_time && (t[0] != -1.0) && !feof(f\_config))

{

short tmp\_mode\_index;

extension = (short)(t[1]+0.01);

tmp\_mode\_index = (short)(t[2]+0.01);

if (extension == 0)

{

mode = tmp\_mode\_index;

st\_mode = -1;

conf.mode = tmp\_mode\_index;

conf.st\_mode = -1;

conf.mode\_index = tmp\_mode\_index;

}

else

{

conf.mode\_index = tmp\_mode\_index;

get\_raw\_3gp\_mode(&(conf.mode), &(conf.st\_mode),conf.mode\_index, extension );

}

fst = (short)(t[3]\*FSCALE\_DENOM+0.5);

get\_isf\_index(&fst); /\* Use "fscale from index" \*/

while (!get\_config(f\_config,t)&& !feof(f\_config)) {}

config\_file\_time = t[0];

}

}

if (fst != conf.fscale)

{

conf.fscale = fst;

init\_coder\_ravs\_p(st, (int) numOfChannels, conf.fscale,

conf.use\_case\_mode, 0);

set\_frame\_length(samplingRate, conf.fscale, &L\_frame, &L\_next,

&L\_next\_st);

conf.fscale\_index = get\_isf\_index(&(conf.fscale)); /\* Use "fscale from index" \*/

}

nb\_hold = L\_frame - nb\_samp;

mvr2r(channel\_right + nb\_samp, channel\_right, nb\_hold);

mvr2r(channel\_left + nb\_samp, channel\_left, nb\_hold);

nb\_samp\_fs = ((nb\_samp\*fac\_down)+frac\_up\_right) / fac\_up;

lg =

read\_data(f\_speech, channel\_right + nb\_hold,

(numOfChannels \* nb\_samp\_fs));

if (numOfChannels == 2)

{

deinterleave(channel\_right + nb\_hold, channel\_left + nb\_hold,

channel\_right + nb\_hold, nb\_samp\_fs);

over\_fs(channel\_left + nb\_hold, channel\_left + nb\_hold, nb\_samp,

fac\_down, mem\_up\_left, &frac\_up\_left);

}

over\_fs(channel\_right + nb\_hold, channel\_right + nb\_hold, nb\_samp,

fac\_down, mem\_up\_right, &frac\_up\_right);

if (((extension == 0) && (conf.extension == 1))

|| ((extension == 1) && (conf.extension == 0)))

{

if (((mode >= 0 && mode <= 9) || mode == 15) && (conf.extension > 0))

{

copy\_coder\_state(st, stravsEnc, 1, conf.use\_case\_mode);

}

else if ((mode >= 0 && mode <= 8) && (conf.extension == 0))

{

copy\_coder\_state(st, stravsEnc, 0, conf.use\_case\_mode);

}

/\* conf.mode = mode; \*/

conf.extension = extension;

}

if (conf.extension > 0)//avs-p10编码模式

{

nb\_bits = get\_nb\_bits\_AVS2(conf.extension, conf.mode, conf.st\_mode);//xhl

if (numOfChannels == 2)//立体声编码

{

nb\_samp =

coder\_ravs\_p\_stereo(hEnvEnc, /\* AAC need \*/

channel\_right, channel\_left,

conf.mode, L\_frame,

serial, st, conf.use\_case\_mode,

conf.fscale, conf.st\_mode,

&nbyte\_ps //PS BYTES //xhl

#ifdef NEW\_SAD

,synSt,

frame

#endif

);

}

else //单声道编码

{

nb\_samp =

coder\_ravs\_p\_mono(hEnvEnc, /\* AAC need \*/

channel\_right,

conf.mode,

L\_frame, serial, st, conf.use\_case\_mode,

conf.fscale

#ifdef NEW\_SAD

,synSt,

frame

#endif

);

}

old\_st\_mode = conf.st\_mode;

if(conf.FileFormat == F3GP)//3gpp格式码流

{

WriteSamplesAMRWBPlus( conf,serial, nb\_bits);

}

else //raw格式码流

{

for(i = 0;i < 4; i++)

{

if (numOfChannels == 2)

{

WriteHeaderPS(conf, (short)nb\_bits, (short)i, nbyte\_ps, f\_serial);//xhl

}

else

{

WriteHeader(conf, (short)nb\_bits, (short)i, f\_serial);

}

WriteBitstreamP(conf, (short)nb\_bits, (short)i, serial, f\_serial);

}

if (numOfChannels == 2)

{

WriteAvsBitstreamP(conf, (short)nb\_bits,serial, f\_serial,&nbyte\_ps);//xhl

}

else

{

WriteAvsBitstreamP(conf, (short)nb\_bits,serial, f\_serial,NBYTE\_SBR);

}

}

}

else //wb编码模式

{

for (i = 0; i < 4; i++)

{

moveAndRound(&channel\_right[i \* 320], speech16, 320);

serial\_size =

(short) E\_IF\_encode(stravsEnc, (Word16) conf.mode, speech16,

serialravs, conf.allow\_dtx);

if(conf.FileFormat == F3GP)

{

WriteSamplesAMRWBPlus(conf,serialravs, serial\_size);

}

else

{

WriteHeader(conf, (short)serial\_size, (short)i, f\_serial);

WriteBitstream(conf, (short)serial\_size, (short)i, serialravs, f\_serial);

}

}

nb\_samp = 4 \* L\_FRAME16k;

}

if (fabs(old\_bitrate-get\_bitrate(&conf)) > 0.00001) {

old\_bitrate = get\_bitrate(&conf);

fprintf(stderr, "Rectime: %2.3f Encoding @ %6.2fkbps\n", rec\_time, old\_bitrate);

}

rec\_time += nb\_samp/((float)(samplingRate));

Processed\_sample += (nb\_samp\_fs\*numOfChannels);

}

if(conf.FileFormat == F3GP)

{

Close3GP(output\_filename);

}

else

{

fclose(f\_serial);

}

Wave\_fclose(f\_speech, bitsPerSample);

if(stravsEnc != NULL)

E\_IF\_exit(stravsEnc);

close\_avsp(st, conf.use\_case\_mode);

#ifdef NEW\_SAD

syndecid\_exit(&synSt);

#endif

exit(EXIT\_SUCCESS);

}

void copyright(void)

{

fprintf(stderr, "\n");

fprintf(stderr, "\n");

}

static void usage(char \*argv)

{

fprintf(stderr,

"Usage: %s -rate <Bit rate> [-mono] | -mi <mode index> [-isf <factor>] [-lc] [-dtx] -ff <3gp/raw> -if <infile.wav> -of <outfile.avs+>\n",

argv);

fprintf(stderr, "\n");

fprintf(stderr, "-rate Bit rate between 6-36 kbps mono or 7-48 kbps stereo \n");

fprintf(stderr, "-mono Force mono encoding \n");

fprintf(stderr, "\n");

fprintf(stderr, "\n");

fprintf(stderr, "-mi Mode Index (0..15 -> R AVS\n 16..47 -> R AVS+) (see ts 26.290 Table 25) \n");

fprintf(stderr,

"-isf Internal Sampling Frequency (0.5... 1.5, default is 1.0).\n");

fprintf(stderr, "\n");

fprintf(stderr, "\n");

fprintf(stderr, "-lc low complexity (for R-AVS+ modes).\n");

fprintf(stderr,

"-dtx enables VAD/DTX functionality (for R-AVS modes).\n");

fprintf(stderr, "\n");

fprintf(stderr, "-ff 3gp File Format / raw format\n");

fprintf(stderr, "-if input audio WAV file.\n");

fprintf(stderr, "-of output RAVS+ 3gp file.\n");

fprintf(stderr, "-cf configuration file\n");

fprintf(stderr, "\n");

}

float get\_bitrate(EncoderConfig \* conf)

{

if (conf->fscale != 0)

{

return (float) (get\_nb\_bits(conf->extension, conf->mode, conf->st\_mode) \* conf->fscale) / (80.0f \*

FSCALE\_DENOM);

}

else

{

return (float) get\_nb\_bits(conf->extension, conf->mode, conf->st\_mode) / 80.0f;

}

}

int get\_core\_mode(float bitrate)

{

float min\_dist = 1e16f;

int mode;

int i;

mode = 0;

for (i = 0; i < 8; i++)

{

if (fabs(bitrate - (float) (NBITS\_CORE[i] + NBITS\_BWE) / 80.0f) <

min\_dist)

{

min\_dist = (float)fabs(bitrate - (float) (NBITS\_CORE[i] + NBITS\_BWE) / 80.0f);

mode = i;

}

}

return mode;

}

int get\_stereo\_mode(float bitrate)

{

float min\_dist = 1e16f;

int mode;

int i;

if (bitrate == 0)

return -1;

mode = 0;

for (i = 0; i < 16; i++)

{

if (fabs(bitrate - (float) (StereoNbits[i] + NBITS\_BWE) / 80.0f) <

min\_dist)

{

min\_dist =

(float)fabs(bitrate - (float) (StereoNbits[i] + NBITS\_BWE) / 80.0f);

mode = i;

}

}

return mode;

}

void get\_raw\_3gp\_mode(short \*mode, short \*st\_mode, short raw\_3gp\_mode ,short extension )

{

short index;

if( raw\_3gp\_mode <= 8 )

{

if (extension != 0)

{

fprintf(stderr, "-isf is not supported by r\_avs\n");

exit(EXIT\_FAILURE);

}

\*mode = raw\_3gp\_mode;

\*st\_mode = -1;

}

else if(raw\_3gp\_mode== 10) /\* 14m \*/

{

\*mode = 2;

\*st\_mode = -1;

}

else if (raw\_3gp\_mode== 11) /\* 18s \*/

{

\*mode = 2;

\*st\_mode = 6;

}

else if (raw\_3gp\_mode== 12) /\* 24m\*/

{

\*mode = 7;

\*st\_mode = -1;

}

else if (raw\_3gp\_mode == 13) /\*24s\*/

{

\*mode = 5;

\*st\_mode = 7;

}

else if(raw\_3gp\_mode >= 16 && raw\_3gp\_mode < 24)

{

\*mode = raw\_3gp\_mode - 16;

\*st\_mode = -1;

}

else if(raw\_3gp\_mode >= 24 && raw\_3gp\_mode <= 47)

{

index = raw\_3gp\_mode - 24;

\*mode = miMode[2\*index];

\*st\_mode = miMode[2\*index+1];

}

else

{

printf("Invalid Mode Index\n");

exit(EXIT\_FAILURE);

}

}

short get\_isf\_index(short \*fscale)

{

short index, i ;

float dist = 512.0f, ftmp;

index = 0;

for (i = 0;i < 14; i++)

{

ftmp = (float)fabs(\*fscale-isfIndex[i]);

if(ftmp < dist)

{

dist = ftmp;

index = i;

}

}

\*fscale = isfIndex[index];

return index;

}

static void parsecmdline(int argc, //

char \*argv[],

char \*\*input\_filename,

char \*\*output\_filename,

char \*\*config\_filename,

char \*\*status\_filename,

EncoderConfig \* conf,

float \*rate)

{

int simple\_mode,r\_avs, r\_avsp\_carac, mi\_mode;

float srate;

float mrate;

if (argc == 1)

{

usage(argv[0]);

exit(EXIT\_FAILURE);

}

conf->extension = 0;

conf->allow\_dtx = 0;

conf->use\_case\_mode= USE\_CASE\_A;

conf->fscale = 0;

conf->mode = -1;

conf->st\_mode = -1;

conf->FileFormat = F3GP;

conf->mode\_index = -1;

conf->fscale\_index = 0;

conf->bc = 0;

simple\_mode = 0;

r\_avs = 0;

mi\_mode = 0;

r\_avsp\_carac = 0;

mrate = -1;

srate = -1;

\*rate = -1;

argc--;

argv++;

\*status\_filename = NULL;

while (argc > 0)

{ if (!strcmp(\*argv, "-mi"))

{

if (simple\_mode)

{

fprintf(stderr, "Can't use -rate with -mi\n");

exit(EXIT\_FAILURE);

}

mi\_mode = 1;

argv++;

argc--;

conf->mode\_index = (short)atoi(\*argv);

if (conf->mode\_index < 0 || conf->mode\_index > 47)

{

fprintf(stderr, "Unknown Mode Index (see TS 26.290 Table 25)\n");

exit(EXIT\_FAILURE);

}

else if (conf->mode\_index == 9 || conf->mode\_index == 14 || conf->mode\_index == 15)

{

fprintf(stderr, "Mode Index %d is reserved (see TS 26.290 Table 21)\n", conf->mode\_index);

exit(EXIT\_FAILURE);

}

else

{

if ( (conf->mode\_index >= 0) && (conf->mode\_index <= 8) ) /\* r\_avs modes \*/

{

get\_raw\_3gp\_mode(&(conf->mode), &(conf->st\_mode),(short) atoi(\*argv), conf->extension = 0);

r\_avs = 1;

}

else if ( (conf->mode\_index >= 10) && (conf->mode\_index <= 13) ) /\* AVS+ tested modes \*/

{

get\_raw\_3gp\_mode(&(conf->mode), &(conf->st\_mode),(short) atoi(\*argv), conf->extension = 0);

conf->extension = 1;

r\_avsp\_carac = 1;

}

else

{

conf->extension = 1;

get\_raw\_3gp\_mode(&(conf->mode), &(conf->st\_mode),conf->mode\_index, conf->extension );

if(conf->fscale == 0)

{

conf->fscale = FSCALE\_DENOM;

conf->fscale\_index = 8;

}

}

}

}

else if (!strcmp(\*argv, "-isf"))

{

if (simple\_mode)

{

fprintf(stderr, "Can't use -rate with -isf\n");

exit(EXIT\_FAILURE);

}

if (r\_avs)

{

fprintf(stderr, "-isf is not supported by r\_avs\n");

exit(EXIT\_FAILURE);

}

argv++;

argc--;

mi\_mode = 1; /\* -isf is only allow with -mi \*/

conf->extension = 1;

if ((atof(\*argv) >= 0.5) && (atof(\*argv) <= 1.5))

{

conf->fscale = (short) ((atof(\*argv) \* FSCALE\_DENOM) + 0.5f);

conf->fscale = (conf->fscale >> 1) << 1;

if (conf->fscale > FAC\_FSCALE\_MAX)

{

conf->fscale = FAC\_FSCALE\_MAX;

}

if (conf->fscale < FAC\_FSCALE\_MIN)

{

conf->fscale = FAC\_FSCALE\_MIN;

}

conf->fscale\_index = get\_isf\_index(&(conf->fscale)); /\* Use "fscale from index" \*/

}

else

{

fprintf(stderr, "Unknown Inernal Sampling Frequency factor\n");

exit(EXIT\_FAILURE);

}

}

else if (!strcmp(\*argv, "-rate"))

{

if(mi\_mode)

{

fprintf(stderr, "Can't use -rate with -mi or -isf \n");

exit(EXIT\_FAILURE);

}

argv++;

argc--;

simple\_mode = 1;

\*rate = (float)atof(\*argv);

conf->extension = 1;

if(\*rate < 6.0 || \*rate > 48.0)

{

fprintf(stderr, "Minimum rate is 6.0kbps and maximum rate is 48.0 kbps\n");

exit(EXIT\_FAILURE);

}

}

else if (!strcmp(\*argv, "-mono"))

{

conf->st\_mode = -2; /\* indicate mono is forced \*/

if(\*rate > 36.0)

{

fprintf(stderr, "Maximum mono rate is 36.0 kbps\n");

exit(EXIT\_FAILURE);

}

}

else if (!strcmp(\*argv, "-lc"))

{

conf->use\_case\_mode = USE\_CASE\_B;

}

else if (!strcmp(\*argv, "-dtx"))

{

conf->allow\_dtx = 1;

}

else if (!strcmp(\*argv, "-bc"))

{

conf->bc = 1;

}

else if (!strcmp(\*argv, "-if"))

{

argv++;

argc--;

\*input\_filename = \*argv;

}

else if (!strcmp(\*argv, "-of"))

{

argv++;

argc--;

\*output\_filename = \*argv;

}

else if (!strcmp(\*argv,"-cf")) {

argv++;

argc--;

\*config\_filename = \*argv;

}

else if (!strcmp(\*argv, "-ff"))

{

argv++;

argc--;

if(!strcmp(\*argv, "raw"))

{

conf->FileFormat = FRAW;

}

else

{

conf->FileFormat = F3GP;

}

}

else if (!strcmp(\*argv, "-sf"))

{

argv++;

argc--;

\*status\_filename = \*argv;

}

else

{

fprintf(stderr, "Unknown option %s\n", \*argv);

exit(EXIT\_FAILURE);

}

argv++;

argc--;

}

if (r\_avsp\_carac && conf->fscale != 0)

{

fprintf(stderr, "-isf is not supported with R AVS caracterized modes\n");

exit(EXIT\_FAILURE);

}

if (conf->st\_mode == -2 && simple\_mode != 1)

{

fprintf(stderr, "Choose right Mode Index to encode mono File\n-mono option is only supported with -rate\n");

exit(EXIT\_FAILURE);

}

if (\*status\_filename == NULL){

char\* pos;

pos = strrchr(\*input\_filename, '/');

if (!pos)

pos = strrchr(\*input\_filename, '\\');

if (!pos)

pos = \*input\_filename;

else

pos++;

\*status\_filename = malloc(strlen(pos)+8);

if (\*status\_filename == NULL)

printf("malloc error\r\n");

strcpy(\*status\_filename, pos);

strcat(\*status\_filename, "\_status");

}

}

static void set\_frame\_length(int samplingRate,

int fscale,

int \*L\_frame, int \*L\_next, int \*L\_next\_st)

{

if (fscale != 0)

{

switch (samplingRate)

{

#ifdef FILTER\_48kHz

case 8000:

case 16000:

case 24000:

case 32000:

case 48000:

\*L\_frame = 2 \* L\_FRAME48k;

break;

#endif

#ifdef FILTER\_44kHz

case 11025:

case 22050:

case 44100:

\*L\_frame = 2 \* L\_FRAME44k;

break;

#endif

default:

fprintf(stderr, "error in sampling frequency. choice of filter are: \n");

#ifdef FILTER\_44kHz

fprintf(stderr, " 11, 22, 44 kHz \n");

#endif

#ifdef FILTER\_48kHz

fprintf(stderr, " 8, 16, 24, 32, 48 kHz \n");

#endif

exit(EXIT\_FAILURE);

break;

}

}

else {

switch (samplingRate)

{

case 8000:

\*L\_frame = L\_FRAME8k;

\*L\_next = L\_NEXT8k;

\*L\_next\_st = L\_NEXT\_ST8k;

break;

case 16000:

\*L\_frame = L\_FRAME16kP;

\*L\_next = L\_NEXT16k;

\*L\_next\_st = L\_NEXT\_ST16k;

break;

case 24000:

\*L\_frame = L\_FRAME24k;

\*L\_next = L\_NEXT24k;

\*L\_next\_st = L\_NEXT\_ST24k;

break;

default:

fprintf(stderr, "error in sampling freq: without fsratio(isf) only 8, 16 or 24 kHz are allowed\n");

exit(EXIT\_FAILURE);

break;

}

}

}

static void GetRate(EncoderConfig \*conf,float rate, const short \*TableRate, short lenght)

{

short index, i ;

float dist = 512.0f, ftmp;

index = 0;

for (i = 0;i < lenght; i+=3)

{

ftmp = (float)fabs(rate\*2 - TableRate[i]);

if(ftmp < dist)

{

dist = ftmp;

index = i;

}

}

conf->mode\_index = TableRate[index+1];

conf->fscale\_index = TableRate[index+2];

get\_raw\_3gp\_mode(&conf->mode, &conf->st\_mode, conf->mode\_index, conf->extension);

conf->fscale = isfIndex[conf->fscale\_index];

}

static void moveAndRound(float \*in, short \*out, int n)

{

int i;

float temp;

for (i = 0; i < n; i++)

{

temp = \*in++;

if (temp >= 0.0)

temp += 0.5;

else

temp -= 0.5;

if (temp > 32767.0)

temp = 32767.0;

if (temp < -32767.0)

temp = -32767.0;

\*out++ = (short) temp;

}

}

static void deinterleave(float \*buf, float \*left, float \*right, int length)

{

int i;

for (i = 0; i < length; i++)

{

left[i] = buf[i \* 2];

right[i] = buf[(i \* 2) + 1];

}

}

int get\_config(FILE \*fp, float t[])

{

int OK = 0;

if (!fp || feof(fp))

return 0;

while (!OK && !feof(fp)) {

char s[100], \*sp;

int ix = 0;

t[0] = 0.0;

fgets(s,99,fp);

sp = strtok(s," \t");

while (sp && ix < 4)

{

t[ix++] = (float)atof(sp);

sp = strtok(0," \t");

}

if (t[0] != 0.0) {

return 1;

}

}

return 0;

}

void close\_avsp(Coder\_State\_P \*st, Word16 UseCaseB)

{

if(st->stClass != NULL && UseCaseB > 0 )

{

free(st->stClass);

st->stClass = NULL;

}

if(st->vadSt != NULL && UseCaseB > 0 )

avs\_vad\_exit(&st->vadSt);

if(st != NULL)

{

free(st);

st = NULL;

}

}

void init\_cod\_hi\_stereo(Coder\_State\_P \*st)

{

int i;

set\_zero(st->old\_exc\_mono,HI\_FILT\_ORDER);

st->filt\_energy\_threshold= 0.0f;

set\_zero(st->old\_wh,HI\_FILT\_ORDER);

set\_zero(st->old\_wh\_q,HI\_FILT\_ORDER);

set\_zero(st->old\_gm\_gain,2);

cos\_window(st->w\_window,0,L\_SUBFR);

for(i=0;i<L\_SUBFR;i++)

{

st->w\_window[i] = st->w\_window[i]\*st->w\_window[i];

}

}

/\* gain quantizer \*/

static void quant\_gain(float gain\_left, /\* i/o \*/

float gain\_right, /\* i/o \*/

float old\_gain[],

int \*\*prm,

const PMSVQ \*gain\_hi\_pmsvq

)

{

float tmp[2];

tmp[0] = gain\_left;

tmp[1] = gain\_right ;

pmsvq(tmp, prm, tmp, old\_gain, gain\_hi\_pmsvq);

}

/\* filter quantizer \*/

static void quant\_filt(float h[], /\* i/o \*/

float old\_h[], /\* i/o \*/

int \*\*prm,

const PMSVQ \*filt\_hi\_pmsvq

)

{

pmsvq(h, prm, h, old\_h, filt\_hi\_pmsvq);

}

/\* filter smoother \*/

static void smooth\_ener\_filter(float \*filter,

float \*threshold)

{

float tmp, ener,old\_ener;

int i;

/\* compute energy over subframe \*/

ener = 0.0001f;

for (i=0; i<HI\_FILT\_ORDER; i++)

{

ener += filter[i]\*filter[i];

}

/\* in any case limit the filter energy \*/

old\_ener = ener;

if (ener > 16.0f){

ener= 16.0f;

}

tmp = ener;

if (tmp < \*threshold)

{

tmp = tmp\*1.414f;

if (tmp > \*threshold)

{

tmp = \*threshold;

}

}

else {

tmp = tmp/1.414f;

if (tmp < \*threshold)

{

tmp = \*threshold;

}

}

\*threshold = tmp;

tmp = (float)sqrt(tmp/old\_ener);

for (i=0; i<HI\_FILT\_ORDER; i++)

{

filter[i] \*= tmp;

}

return;

}

void cod\_hi\_stereo(float speech\_hi[],

float right\_hi[],

float AqLF[],

int param[],

Coder\_State\_P \*st)

{

float \*exc\_mono = speech\_hi-M;

float \*exc\_side = right\_hi-M;

int i\_subfr,i,k,t,j;

float \*p\_Aq;

/\* covariance matrix \*/

float r[HI\_FILT\_ORDER][HI\_FILT\_ORDER];

float c[HI\_FILT\_ORDER];

/\* estimated LMS filters \*/

float wh[NB\_DIV\*HI\_FILT\_ORDER];

float \*p\_h;

/\* signal big subframe pointers \*/

float \*x,\*y;

float buf[L\_DIV+L\_SUBFR];

float gain\_left[NB\_SUBFR];

float gain\_right[NB\_SUBFR];

/\* energies \*/

float energy\_right;

float energy\_right\_q;

float energy\_left;

float energy\_mono;

float energy\_left\_q;

float corr\_left\_right;

float gain\_fact;

int \*prm;

mvr2r(st->old\_exc\_mono,exc\_mono-HI\_FILT\_ORDER,HI\_FILT\_ORDER);

/\* compute the residual of the hi mono and right \*/

p\_Aq = AqLF;

for (i\_subfr=0; i\_subfr<L\_FRAME\_P; i\_subfr+=L\_SUBFR)

{

residu(p\_Aq,M, &speech\_hi[i\_subfr], &exc\_mono[i\_subfr], L\_SUBFR);

residu(p\_Aq,M, &right\_hi[i\_subfr], &exc\_side[i\_subfr], L\_SUBFR);

p\_Aq += (M+1);

}

residu(p\_Aq,M, &speech\_hi[i\_subfr], &exc\_mono[i\_subfr], L\_SUBFR);

residu(p\_Aq,M, &right\_hi[i\_subfr], &exc\_side[i\_subfr], L\_SUBFR);

/\* compute real side signal \*/

for(i=0;i<L\_FRAME\_P+L\_SUBFR;i++){

exc\_side[i] = exc\_mono[i] - exc\_side[i];

}

/\* save fir state for next frame \*/

mvr2r(exc\_mono+L\_FRAME\_P-HI\_FILT\_ORDER,st->old\_exc\_mono,HI\_FILT\_ORDER);

/\* compute the wiener filters, raw on each frame with covariance method\*/

p\_h = wh;

for(i=0;i<NB\_DIV;i++){

/\* set the pointer to parameters \*/

prm = param + i\*NPRM\_STEREO\_HI\_X;

/\* set signal pointers \*/

x = exc\_mono + i\*L\_DIV;

y = exc\_side + i\*L\_DIV;

/\* compute cross-correlation terms \*/

for(k=0;k<HI\_FILT\_ORDER;k++){

c[k] =0.0f;

for(t=0;t<L\_DIV;t++)

{

c[k] += y[t]\*x[t-k];

}

for(t=L\_DIV;t<L\_DIV+L\_SUBFR;t++)

{

c[k] += y[t]\*x[t-k];

}

}

/\* compute correlation matrix \*/

for(k=0;k<HI\_FILT\_ORDER;k++)

{

for(j=k;j<HI\_FILT\_ORDER;j++)

{

r[k][j] =0.0f;

for(t=0;t<L\_DIV;t++)

{

r[k][j] += x[t-k]\*x[t-j];

}

for(t=L\_DIV;t<L\_DIV+L\_SUBFR;t++)

{

r[k][j] += x[t-k]\*x[t-j];

}

}

}

/\* compute a solution to the linear system \*/

if(cholsolc(r,c,p\_h,HI\_FILT\_ORDER))

{

/\* cholesky failed use panning \*/

for(k=1;k<HI\_FILT\_ORDER;k++){

p\_h[k] = 0.0f;

}

p\_h[0] = c[0]/(r[0][0]+1.0f);

}

/\* wiener filter energy smoothing \*/

smooth\_ener\_filter(p\_h,&st->filt\_energy\_threshold);

/\* quantize the filters\*/

quant\_filt(p\_h,st->old\_wh\_q,&prm,st->filt\_hi\_pmsvq);

/\* local synthesis \*/

fir\_filt(p\_h,HI\_FILT\_ORDER,x,buf,L\_DIV+L\_SUBFR);

/\* compute the gain matching figures in the excitation domain \*/

energy\_left = 0.001f;

energy\_right = 0.001f;

energy\_left\_q = 0.001f;

energy\_right\_q = 0.001f;

energy\_mono = 0.001f;

corr\_left\_right = 0.00f;

for(t=0;t<L\_DIV+L\_SUBFR;t++)

{

/\* mono + side \*/

energy\_left += (x[t] + y[t])\*(x[t]+y[t]);

energy\_left\_q += (x[t] + buf[t]) \*(x[t]+buf[t]);

/\* mono \*/

energy\_mono += x[t] \* x[t];

/\* mono - side \*/

energy\_right += (x[t] - y[t])\*(x[t]-y[t]);

energy\_right\_q += (x[t]- buf[t])\*(x[t]-buf[t]);

corr\_left\_right += (x[t] - y[t])\*(x[t]+y[t]);

}

/\* compute the gain matching \*/

gain\_right[i] = 10.0f\*(float)log10(energy\_right/energy\_right\_q+0.001f);

gain\_left[i] = 10.0f\*(float)log10(energy\_left/energy\_left\_q +0.001f);

gain\_fact = (4.0f\*energy\_mono)/(energy\_left+energy\_right);

if(gain\_fact < 1.0f){

/\* we have signal cancelation, take no risks \*/

if(gain\_right[i] >0.0f){

gain\_right[i] = my\_max(gain\_right[i]+10.0f\*(float)log10(gain\_fact),0.0f);

}

if(gain\_left[i] > 0.0f){

gain\_left[i] = my\_max(gain\_left[i]+10.0f\*(float)log10(gain\_fact),0.0f);

}

}

/\* quantize the gains \*/

quant\_gain(gain\_left[i],gain\_right[i],st->old\_gm\_gain,&prm,st->gain\_hi\_pmsvq);

/\* next frame \*/

p\_h += HI\_FILT\_ORDER;

}

/\* save last filter\*/

mvr2r(&wh[(NB\_DIV-1)\*HI\_FILT\_ORDER],st->old\_wh,HI\_FILT\_ORDER);

}

void init\_tvc\_stereo\_encoder(Coder\_State\_P \*st)

{

st->mem\_stereo\_ovlp\_size = 0;

set\_zero(st->mem\_stereo\_ovlp,L\_OVLP\_2k);

}

int q\_gain\_pan( /\* output: return quantization index \*/

float \*gain /\* in/out: quantized gain \*/

)

{

int index;

index = (int)floor((\*gain +2.0f)\*(32.0f) + 0.5f);

if (index < 0) index = 0;

if (index > 127 ) index = 127;

\*gain = (float) index / 32.0f - 2.0f;

return(index); /\* 0...127\*/

}

void ctvc\_stereo(float side[], /\* input: speech[-M..lg] \*/

float mono[],

float synth[], /\* in/out: synth[-M..lg] \*/

float wovlp[], /\* i/o: wovlp[0..127] \*/

int ovlp\_size, /\* input: 0, 64 or 128 (0=acelp) \*/

int L\_frame, /\* input: frame length \*/

int nb\_bits, /\* input: number of bits allowed \*/

int prm[], /\* output: tvc parameters \*/

int pre\_echo)

{

int i, k, i\_subfr, lg;

int lext=32;

float tmp, gain, fac\_ns,gain\_pan;

float xri[L\_TVC\_LB];

float xn[L\_TVC\_LB];

float wm[L\_TVC\_LB];

float xnq[L\_TVC\_LB];

#ifndef COS\_FAC

float window[L\_TVC\_LB];

#else

float tmpfloat;

#endif

float gain\_shap[8];

/\*------ set length of overlap (lext) and length of encoded frame (lg) -----\*/

switch (L\_frame) {

case 40:

lext = 8;

break;

case 80:

lext = 16;

break;

case 160:

lext = 32;

break;

};

lg = L\_frame + lext;

/\* built window for overlaps section \*/

#ifndef COS\_FAC

cos\_window(window, ovlp\_size, lext);

#endif

for (i=0; i<lg; i++){

xn[i] = side[i];

wm[i] = mono[i];

}

for (i=0; i<ovlp\_size; i++)

{

#ifndef COS\_FAC

xn[i] \*= window[i];

wm[i] \*= window[i];

#else

tmpfloat=cos\_fac(i,ovlp\_size,lext);

xn[i]\*=tmpfloat;

wm[i] \*= tmpfloat;

#endif

}

for (i=0; i<lext; i++) {

#ifndef COS\_FAC

xn[L\_frame+i] \*= window[ovlp\_size+i];

wm[L\_frame+i] \*= window[ovlp\_size+i];

#else

tmpfloat=cos\_fac(ovlp\_size+i,ovlp\_size,lext);

xn[L\_frame+i] \*= tmpfloat;

wm[L\_frame+i] \*= tmpfloat;

#endif

}

if(pre\_echo)

{

tmp = 0.0f;

for(i=0;i<lg;i+=16){

gain\_shap[i/16] = 0.001f;

for(k=0;k<16;k++)

{

gain\_shap[i/16] += wm[i+k]\*wm[i+k];

}

tmp += (float)log10(gain\_shap[i/16]);

}

tmp /= (float)(lg/16);

for(i=0;i<lg;i+=16){

gain\_shap[i/16] = (float) sqrt(gain\_shap[i/16]\*pow(10.0f,-tmp));

gain\_shap[i/16] = my\_min(2.0,my\_max(0.5,gain\_shap[i/16]));

for(k=0;k<16;k++)

{

xn[i+k] /= gain\_shap[i/16];

wm[i+k] /= gain\_shap[i/16];

}

}

}

gain\_pan = get\_gain(xn, wm, lg);

nb\_bits -= 7;

prm[0] = q\_gain\_pan(&gain\_pan);

for(i=0;i<lg;i++){

xn[i] -= gain\_pan \* wm[i];

}

fft3(xn, xri, (short)lg);

xri[1] = 0.0; /\* freq bin at 1000 Hz zeroed \*/

adap\_low\_freq\_emph\_stereo(xri, 4\*lg);

nb\_bits -= (7); /\* gain = 7 bits \*/

fac\_ns = AVQ\_cod\_stereo(xri, prm+2, nb\_bits, lg/8);

for(i=0; i<lg; i++)

{

xri[i] = (float)prm[i+2];

}

adap\_low\_freq\_deemph\_stereo(xri,4\*lg);

xri[1] = 0.0; /\* freq bin at 6400 Hz zeroed \*/

ifft3(xri, xnq, (short)lg);

if (pre\_echo) {

for(i=0;i<lg;i+=16){

for(k=0;k<16;k++) {

xnq[i+k] \*= gain\_shap[i/16];

xn[i+k] \*= gain\_shap[i/16];

wm[i+k] \*= gain\_shap[i/16];

}

}

}

gain = get\_gain(xn, xnq, lg);

prm[1] = q\_gain\_tvc(xnq, lg, &gain);

for (i=0; i<lg; i++) {

xnq[i] = gain \*xnq[i] + gain\_pan \* wm[i];

}

for (i=0; i<ovlp\_size; i++)

{

#ifndef COS\_FAC

xnq[i] \*= window[i];

#else

tmpfloat=cos\_fac(i,ovlp\_size,lext);

xnq[i] \*= tmpfloat;

#endif

}

for (i=0; i<lext; i++)

{

#ifndef COS\_FAC

xnq[i+L\_frame] \*= window[ovlp\_size+i];

#else

tmpfloat=cos\_fac(ovlp\_size+i,ovlp\_size,lext);

xnq[i+L\_frame] \*= tmpfloat;

#endif

}

for (i=L\_frame+lext; i<lg; i++)

{

xnq[i] = 0;

}

/\*-----------------------------------------------------------\*

\* TVC overlap and add. Update memory for next overlap. \*

\*-----------------------------------------------------------\*/

for (i=0; i<L\_OVLP\_2k; i++)

{

xnq[i] += wovlp[i];

}

/\* save overlap for next frame \*/

for (i=0; i<lext; i++)

{

wovlp[i] = xnq[i+L\_frame];

}

for (i=lext; i<L\_OVLP\_2k; i++)

{

wovlp[i] = 0.0;

}

/\*-----------------------------------------------------------\*

\* find excitation and synthesis \*

\*-----------------------------------------------------------\*/

for (i\_subfr=0; i\_subfr<L\_frame; i\_subfr++)

{

synth[i\_subfr] = xnq[i\_subfr];

}

return;

}

/\*-----------------------------------------------------------------\*

\* Funtion c\_stereo \*

\* ~~~~~~~~~~~~~~~~~~~~ \*

\*-----------------------------------------------------------------\*/

void cod\_tvc\_stereo(float mono\_2k[],

float right\_2k[],

int param[],

int brMode,

int mod[],

int fscale,

Coder\_State\_P \*st)

{

float sig\_buf\_mono[TVC\_L\_FFT\_2k];

float sig\_buf\_right[TVC\_L\_FFT\_2k];

float sig\_buf\_side[TVC\_L\_FFT\_2k];

float synth\_tvc[L\_FRAME\_2k];

float synth[L\_FRAME\_2k];

int ovlp\_size[4+1];