User's Manual

ADVANSEE
AdvLib

Référence Document	ADV_UM_9009_LibAdvansee
Auteurs	E.Pauchard
Révision	1.1
Date	Feb 13th. 2013

ADV_UM - 1.3 - 20120608

ADVANSEE – 9 rue Alfred Kastler – CS30750 – 44307 Nantes cedex 3– France
Tél. : 33 (0)2 4050 2173 – Email : contact@advansee.com – www.advansee.com
SARL au capital de 40 200€ - 510 580 814 RCS Nantes - SIRET 510 580 814 00034- TVAI : FR 31 510 580 814

Document history

Version	Date	Nom	Commentaires
1.0	Feb 13th, 2013	E.Pauchard	First release
1.1	Oct 4th, 2013	E. Poirier	Typo corrections.

Table of Contents

1 Introduction

Advansee's library provides all elementary functions needed to capture images from Mipsee or Seeklop. It provides portable code on all platforms, so algorithm can be developed on PC and directly ported to MIPSEE or SEEKLOP, making debug easier.

The main features of this library are:

- Portable code on PC, Mipsee or Seeklop
- Different video sources: image files, video file, ov7962,...
- Image transmission using Advansee TCP format
- Image display on Seeklop (S-Video or TFT)
- Command server for distant interaction from PC

The best way to use Advansee's library is within the Advansee's SDK, providing a 100% preconfigured project environment.

Library structure is presented in section 2: you should have all the files listed in this chapter.

For a quick start with Advansee's library, please refer to section 4 where you will find description of the most important modules of the library.

Section 5 presents modules of secondary importance. Finally, section 6 gives a complete example of Advansee's library usage.

2 Library Structure overview

```
./lib
├── documentation
├── include
│   ├── advCommandProtocol.h
│   ├── advFileAccess.h
│   ├── advSynchronize.h
│   ├── commandTcpServer.h
│   ├── trace.h
│   ├── videoSinkSeeklop.h
│   ├── videoSource.h
│   └── videoTcpFlow.h
├── libadvansee_mipsee
│   ├── debug
│   │   └── libadvansee.a
│   └── libadvansee.a
├── libadvansee_pc
│   ├── debug
│   │   └── libadvansee.a
│   └── libadvansee.a
├── libadvansee_seeklop
│   ├── debug
│   │   └── libadvansee.a
│   └── libadvansee.a
└── libadvansee_v1.10.tar.bz2
```

2.1 Binaries

The Advansee Library has been compiled statically and the binary file is called **libadvansee.a**

Binaries are provided for 3 targets :

- PC (x86 32bits)
- Seeklop (Arm cortex A-8)
- Mipsee (Arm-11)

For each target, the binary is placed in the corresponding folder :

- libadvansee_mipsee : binary for MIPSEE
- libadvansee_seeklop : binary for SEEKLOP
- libadvansee_pc : binary for PC

For each target, a « debug » version is provided in the « debug » directory. This version outputs more info on stdout during execution of programs.

2.2 Modules Overview

2.2.1 VideoSource (videoSource.h)

This module is used to capture images. Same code is nearly 100% portable on all 3 targets (mipsee, seeklop and PC).

It is possible to get image from:

- Sequence of image files (.jpg, .png, ...)
- Video file (.avi)
- Image stream using Advansee's protocol
- Webcam
- Platform camera: ov7962 (on Seeklop and Mipsee)

2.2.2 VideoSink (on Seeklop only) (videoSinkSeeklop)

This module provides functions to output images to S-VIDEO or TFT outputs of Seeklop's board.

2.2.3 Advansee TCP Image Stream (videoTcpFlow)

This module provides function to output images from Seeklop or Mipsee to a distant PC via Ethernet.

Due to limited resources on host platform or bandwith of Ethernet cable, stream is limited to following performances for VGA images:

- jpeg image type 30FPS
- grayscale images 30FPS
- color images 15 FPS

2.2.4 Command Protocol (commandTcpServer)

This module provides functions to control Seeklop or Mipsee's applications by a distant PC.

2.2.5 Special File Access (advFileAccess)

Provides access to special files on Seeklop or Mipsee in order to control:

- leds
- gyroscope
- luminosity sensor

2.2.6 Synchronize (advSynchronize)

Provides inter-thread synchronization functions.

2.2.7 Trace (trace.h)

Provides standard output method (wrapper for printf) that can be easily turned on or off at compile time using defines macro.

2.3 HTML Documentation

AdvLib source code is fully documented. This documentation is available in HTML format in “documentation” directory.

To open AdvLib Source code documentation, open file:

```
./lib/documentation/html/index.html
```

3 Dependencies

The library has been compiled statically in C. You need to have some packages on your computer in order to compile your programs with Advansee Library:

3.1 Ubuntu

AdvLib has been developped on Ubuntu.

3.2 Packages

AdvLib needs the following packages to be installed :

- pthread
- opencv (2.4.2)

For Seeklop target, AdvLib needs two additional librairies :

- libipu
- libvpu

These libraries are included in Advansee's SDK. If you use the library in the SDK context, it should not be a problem.

4 Main Modules description

4.1 Video Source

You need to include “videoSource.h” to your source code to use this module.

4.1.1 Generalities

This module allows image capture from multiple sources. Code using this module is nearly 100% portable on all Advansee's platform (Seeklop, Mipsee or PC).

All capture parameters are directly accessible in the video structure **AdvVideoSource**.

To retrieve image from video source:

- declare and alloc an IplImageHeader structure
- pass a pointer of this header to the **adv_videoSourceGetFrame** function
- use this image as a standard IplImage (opencv compatible)

4.1.2 Basic Usage

1. Create video structure
2. Set video source type by setting **videoSourceType** variable
3. Set all other capture parameters
4. Initialize video source using this structure
5. Retrieve as many frames as needed
6. Close video source

4.1.3 Parameters

4.1.3.1 General parameters of video source structure

These parameters of **AdvVideoSource** should be set for any video source type:

- **width:** expected width *(default: 640)*
- **height**: expected height *(default: 480)*
- **imageType**: set to FORMAT_ADV_GRAY to load grayscale images, otherwise opencv color BGR image will be loaded

4.1.3.2 Platform's Camera Source: VIDEO_SOURCE_PLATFORM_CAMERA

This video mode is recommended for use on Seeklop or Mipsee.

- **filename**: device name (usually /dev/video0)
- **fps**: capture framerate (possible values 30-15-10-6-5-3-2-1)
- **rotation:** rotation of image source (supported values: ROTATION_VFLIP on Seeklop, ROTATION_VFLIP and ROTATION_HFLIP on Mipsee)

Note: On **Seeklop**, it is possible to use **FORMAT_ADV_JPG** to get jpeg compressed images from video source module. Images are captured using camera, compressed to jpeg using Seeklop's VPU and buffer is sent to user.

The buffer pointer is stored in provided IplImage's **imageData** parameter, and buffer size is stored in provided IplImage's **imageSize** parameter (see examples below).

4.1.3.3 File source Usage: VIDEO_SOURCE_STILLS

This video mode is recommended for use on PC.

In this mode, images are sequentially loaded from a succession of images. Images should be named using a prefix (including directory location), an image number and an extension.

Images are decoded with opencv, so many formats are supported (at least: bmp, jpeg, png).

Additionally to parameters described in 4.1.3, following variables of **AdvVideoSource** must be set :

- **filename**: set images prefix
- **extName**: set images extension
- **start**: set first image number *(default: 0)*
- **stop**: set last image number *(default: 10000)*
- **stillFactor**: set to number increment (to load one of X images) *(default: 1)*

4.1.3.4 Video File Source: VIDEO_SOURCE_VIDEO

In this mode, a video file is opened and images are extracted from this file. Opencv is used to open the video file so many formats are supported.

Additionally to parameters described in 4.1.3, following variables of **AdvVideoSource** must be set :

- **filename**: complete video file name

4.1.3.5 Webcam Source: VIDEO_SOURCE_WEBCAM

Webcam is opened using v4l2 standard API and opencv. Framerate is not settable in this mode.

Additionally to parameters described in 4.1.3, following variables of **AdvVideoSource** must be set :

- **filename**: device name (usually /dev/video0)

4.1.3.6 Advansee TCP stream Source: VIDEO_SOURCE_TCP

This mode will receive images sent using Advansee's TCP stream protocol.

Additionally to parameters described in 4.1.3, following variables of **AdvVideoSource** must be set :

- **filename**: server hostname (ex: mipsee000245)

4.1.4 Example

```
// Allocate source
struct AdvVideoSource *videoSourceConf = adv_videoSourceCreate();
if (videoSourceConf == NULL)
{
      TRACE_ERROR("Could not create video source");
      return EXIT_FAILURE;
}

// Set Camera Parameters for PLATFORM acquisition of VGA @30fps
videoSourceConf->filename = "/dev/video0";
videoSourceConf->videoSourceType = VIDEO_SOURCE_PLATFORM_CAMERA;
videoSourceConf->width = 640;
videoSourceConf->height = 480;
videoSourceConf->imageType = FORMAT_ADV_GRAY;
```

```
videoSourceConf->fps = 30;

// init source with these values
if (adv_videoSourceInit(videoSourceConf)<0)
{
	TRACE_ERROR("Could not initialize video source");
	return EXIT_FAILURE;
}

// Warning: Init an opencv image HEADER
if ((grabFrame = cvCreateImageHeader( cvSize(videoSourceConf->width, videoSourceConf->height),
IPL_DEPTH_8U, videoSourceConf->imageType == FORMAT_ADV_GRAY ? 1 : 3)) == NULL)
	ALLOC_FAILURE("grabFrame");

while(1)
{
	// get image
	if (adv_videoSourceGetFrame(videoSourceConf, grabFrame)<0)
	{
		TRACE_INFO("Image Capture Error");
		localSystemState = UNINIT;
		break;
	}
	// New image is available in grabFrame!

	// Use image as any IplImage with Opencv
	cvCanny(grabFrame, grabFrame, 100, 200, 3);
}

//// Release Memory
adv_videoSourceClose(videoSourceConf);
cvReleaseImageHeader(&grabFrame);
```

4.2 Video Sink

This module is only operational on Seeklop.

To use this module you need to include “**videoSinkSeeklop.h**” to your code.

4.2.1 Generalities

This code provides functions to output images on Seeklop' S-VIDEO or TFT output. Images are scaled, color-converted by IPU. Image rotation is also available.

4.2.2 Basic usage

1. Create Video Sink Structure **AdvVideoSinkSeeklop** with **adv_videoSinkCreate**
2. Manually fill structure parameters
3. Initialize Video Sink with **adv_videoSinkInit**
4. Send an image to the sink (image will be copied to IPU's buffers) with **adv_videoSinkDisplay**
5. Stop video sink with **adv_videoSinkStop**
6. Free video sink with **adv_videoSinkFree**

4.2.3 Parameters

4.2.3.1 Input image parameters

- **ImageType**: Advansee's input image color format (see videoSource.h)
  - Following formats should be compatible:
    - FORMAT_ADV_YUV420 (recommended)
    - FORMAT_ADV_NV12
    - FORMAT_ADV_RGB565
    - FORMAT_ADV_UYVY
  - To convert from a RGB image, please consider **adv_convertRgb2Yuv420** from VideoSource module (see html documentation).
- **input_width** : input image width in pixels
- **input_height** : input image height in pixels

4.2.3.2 Output image parameters: image transform

- **crop_left**: crop pixels from left of input image
- **crop_right**: crop pixels from right of input image
- **crop_top**: crop pixels from top of input image
- **crop_bottom**: crop pixels from bottom of input image
- **rotate**: Image rotation (see videoSource.h)

4.2.3.3 Output image parameters: display

- **axis_left**: Left border of image axis on screen
- **axis_top**: Top border of image axis on screen
- **disp_width**: width of image on screen

- **disp_height**: height of image on screen
- **tv_out**: Set to 1 to use S-VIDEO output, 0 to use TFT output
- **tv_mode**: 0: NTSC, 1: PAL, 2: 720p

4.2.4 Example

```
#include "videoSinkSeeklop.h"
#include "trace.h"

// For image conversion:
#include "videoSource.h"

// Create and init video display pipeline
struct AdvVideoSinkSeeklop* videoSink = adv_videoSinkCreate();
if (videoSink == NULL)
{
      TRACE_ERROR("%s: could not init sink", __func__);
      goto srcFail;
}
videoSink->input_width = 640;
videoSink->input_height = 480;
videoSink->imageType = FORMAT_ADV_YUV420;
videoSink->disp_width = 720;
videoSink->disp_height = 480;
videoSink->tv_out = tv_out;
videoSink->tv_mode = SEEKLOP_SINK_TV_NTSC;
videoSink->rotate = ROTATION_HFLIP;

if (adv_videoSinkInit(videoSink)<0)
{
      TRACE_ERROR("%s: could not init sink", __func__);
      exit(-1);
}

// Load VGA image from jpeg
IplImage *myImage = cvLoadImage("image.jpg", CV_LOAD_IMAGE_COLOR);
if (!myImage)
{
      TRACE_ERROR("%s: could not load image", __func__);
      exit(-1);
}

// Convert image to YUV420 type (12 bytes per pixel)
unsigned char imageConverted[640*480*3/2];
adv_convertRgb2Yuv420(myImage->imageData, imageConverted, 640, 480, 1);

if (adv_videoSinkDisplay(videoSink, imageConverted, 640*480*3/2) != eOK)
{
      TRACE_ERROR("%s: Error in displaying frame", __func__);
}

// Image is displayed until a new image comes in or until sink is stopped
// In that case, wait 10s then stop
sleep(10);

TRACE_INFO("%s: Stopping Video Sink...", __func__);
```

```
adv_videoSinkStop((*videoMg)->videoSink);
adv_videoSinkFree((*videoMg)->videoSink);
```

4.3 Advansee TCP Image Stream

To use this module you need to include “videoTcpFlow.h” to your code.

4.3.1 Generalities

Transfer images over network. Images can be sent in sent in several formats:

- Grayscale (up to 30FPS in VGA)
- Raw UYVY (up to 10FPS in VGA)
- Raw YUV420 or NV12 (up to 15FPS in VGA)
- Jpeg (up to 30FPS in 720p)

This module creates a server listening on a specified port (recommended use of macro STREAM_PORT to use port 5412). When clients connect to this port (using server's machine hostname), server starts sending data.

First int sent will be, for each frame, ADV_MAGIC_NUMBER (0xAD1A15EE).

All values of header are converted to network endianness.

4.3.2 Basic usage

1. Create Streaming Server **AdvStreamServer** using **adv_streamCreate**
2. Create buffer **AdvStreamBuffer** using **adv_streamCreateImageHeader**
    - AdvStreamBuffer includes allocated space for image data and Advansee's image header
3. Copy your image to allocated space pointed to by **AdvStreamBuffer->imagePtr**
4. Send image over network using **adv_streamNewData**
5. Close server to end image stream using **adv_streamClose**
6. Release buffer using **adv_streamReleaseImageHeader**

4.3.3 Parameters

4.3.3.1 AdvStreamBuffer

- **sheader:** Header structure, can be updated for each frame
- **imagePtr:** Pointer to memory allocated in **adv_streamCreateImageHeader** function. To reallocate this memory (in case of dynamic image size change for example), please use **adv_streamUpdateStreamHeader** (see html documentation).

4.3.3.2 AdvStreamHeader

These parameters are initialized using **adv_streamCreateImageHeader** (all values are automatically converted to network endianness):

- magic: will always be the first integer transferred
- usadvformat: Advansee's image format
- uswidth: image width
- usheight: image height
- dataSize: number of bytes to allocate for buffer

These parameters are updated for each new frame using **adv_streamNewData** (all values are automatically converted to network endianness):

- usNumTrame: frame counter
- dataSize: actual number of bytes of image data
- lSec: image timestamp (number of seconds since epoch)
- lMSec: image timestamp (number of milliseconds since epoch)
- cLuminosity: string of 16 characters containing luminosity information (not on all platforms)
- cPosition: string of 16 characters containing gyroscopic information (not on all platforms)

4.3.4 Example

```
#include "videoTcpFlow.h"
#include "videoSource.h"
#include <sys/time.h>

// Create a streaming server
struct AdvStreamServer *streamingServer = adv_streamCreate();
if (streamingServer == NULL){
      printf("Error: unable to create streaming server"); exit(-1);}

// Create an image data buffer structure with enough space to hold entire image data (no
verification performed).
struct AdvStreamBuffer *dataBuffer = adv_streamCreateImageHeader(FORMAT_ADV_JPG, 640, 480,
640*480*sizeof(char)*3/2);

// Use your usual function to read an image and retrieve a pointer to its data and its size (may be
varying ie for JPEG).
unsigned char *myImg = read_an_image(file);
int dataSize = get_image_size();

// Copy image to your structure.
memcpy(dataBuffer->imagePtr, myImg, dataSize);

// Set timestamp if not available (NOTE: need to #include <sys/time.h>).
struct timeval tv;
gettimeofday(&tv, NULL);

// Send image.
if (adv_streamNewData(streamingServer, dataBuffer, dataSize, &tv)<0)
      printf("Error while streaming image");

// Close server and release memory.
adv_streamClose(streamingServer);

// release memory used by image header.
adv_streamReleaseImageHeader(dataBuffer);
```

4.4 Trace

To use this module you need to include “trace.h” to your code.

4.4.1 Generalities

Output messages to STDOUT. Messages can be belong to different categories: DEBUG, INFO, WARNING, ERROR, FATAL.

At compile time it is possible to exclude some message categories. (example: exclude all DEBUG messages)

4.4.2 Basic usage

- Use TRACE_DEBUG, TRACE_INFO, TRACE_WARNING, TRACE_ERROR, TRACE_FATAL instead of printf (same arguments)
- At compile time, define TRACE_LEVEL to exclude all inferior categories from build:
  - TRACE_LEVEL_DEBUG, TRACE_LEVEL_INFO, TRACE_LEVEL_WARNING, TRACE_LEVEL_ERROR, TRACE_LEVEL_FATAL
- In Advansee's SDK, “debug” build defines TRACE_LEVEL as TRACE_LEVEL_DEBUG
- In Advansee's SDK, “release” build defines TRACE_LEVEL as TRACE_LEVEL_INFO

4.4.3 Parameters

4.4.4 Example

```
// Do not display DEBUG nor INFO messages:
#define TRACE_LEVEL TRACE_LEVEL_WARNING
#include "trace.h"

// For errno and strerror:
#include <string.h>
#include <errno.h>

TRACE_DEBUG("Debug message\n"); // will not be displayed if TRACE_LEVEL >= TRACE_LEVEL_DEBUG
TRACE_INFO("Info message %d\n", 1); // will not be displayed if TRACE_LEVEL >= TRACE_LEVEL_INFO

TRACE_FATAL("Fatal error: %s\n", strerror(errno));
```

5 Secondary Modules Description

5.1 Command Protocol

To use this module you need to include “commandTcpServer.h” to your code.

5.1.1 Generalities

Commands server for remote control of applications on Advansee's Seeklop or Mipsee.

Clients can send flags to the server. On platform, user can retrieve the value of flags at any time. For each flag, 2 bytes of data can be stored.

A message is made of 3 bytes in following order:

```
FLAG_NB DATA_1 DATA_2
```

Where:

- FLAG_NB is flag identification (0-31)
- DATA_1 is the MSB of data
- DATA_2 is the LSB of data

After message reception, server will answer following acknowledge message:

```
0xff DATA_1 DATA_2
```

With DATA_1 and DATA_2 same as received.

Acknowledge only means that message has been completely received, not necessarily understood!

Server will close connection after sending acknowledge message.

5.1.2 Basic usage

1. Call function **adv_startCommandServer**
   - This function will start a thread listening for incoming messages
   - server will update an int variable with **flags sent by clients**
2. User can ask the value of current flag variable at any time using **adv_getCommandFlag**
3. Last data sent by client is stored for each flag, user can get its value with **adv_getFlagData**
4. To reset the command flag, use **adv_resetCommandFlag**
5. Call function **adv_stopCommandServer** to stop the server thread and free memory

5.1.3 Parameters

- Use defined macro FLAG to convert from int to flag mask
- **adv_getFlagData** will return received data converted to an integer of the form: 0x0000XXYY where XX is DATA_1 and YY is DATA_2

5.1.4 Example

```
#include “commandTcpServer.h”
#define FLAG_ONE 1
#define FLAG_TWO 2
#define FLAG_THREE 3

struct AdvServerData *commandConf = adv_startCommandServer(4305);
if (!commandConf)
```

```
{
	TRACE_ERROR("Could not initialize command server");
	exit(EXIT_FAILURE);
}
// Every second, check flags and exit if FLAG_ONE is received. Print data associated with FLAG_ONE
int loop = 1;
while(loop)
{
	sleep(1);
	// Get current flag
	int commands = adv_getCommandFlag(commandConf);

	if (commands & FLAG(FLAG_ONE))
	{
		int data = adv_getFlagData(commandConf, FLAG_ONE);
		adv_resetCommandFlag(commandConf, FLAG_ONE);
		printf("Received DATA for FLAG_ONE is : %d", data);
		loop=0;
	}
}

adv_stopCommandServer(commandConf);
```

5.2 Special File Access

To use this module you need to include “advFileAccess.h” to your code.

5.2.1 Generalities

On Seeklop: access to green led, red led and luminosity value

On Mipsee with OV7962: access to green led, red led, gyroscopic and luminosity values

On Mipsee with MT9M131: access to green led, red led. (No gyroscope nor luminosity sensor)

To pass defined argument to functions, please use ASTRINGZ macro to correctly expand values.

(ex: **adv_writeIntToFile(**ASTRINGZ(LEDR), 1))

5.2.2 Basic usage

1. Specify which camera board you are using by defining **OV7962** or **MT9M131** before including **advFileAccess.h**
2. For LED control use **adv_writeIntToFile**
   - argument **LEDG** will control green led, **LEDR** will control red led
   - On Seeklop: writing 0 will turn led on, 1 will turn led off
   - On Mipsee: writing 1 will turn led on, 0 will turn led off
3. To retrieve values from gyroscope use **adv_copyCharFromFile**
   - Allocate a buffer of 16 bytes
   - use **adv_copyCharFromFile** to copy value to this buffer
   - **in case of error, buffer will be filled with “0\0”**
4. To retrieve values from luminosity sensor use **adv_readIntFromFile**
   - in case of file error, -1 will be returned

- in case of number conversion error, UINT_MAX will be returned

5.2.3 Parameters

5.2.4 Example

On MIPSEE with OV7962, turn green led on and read luminosity and gyroscopic values.

```
#define OV7962
#include "advFileAccess.h"
// Turn green led ON
adv_writeIntToFile(ASTRINGZ(LEDG), 0);
// Read lum value
int lum = adv_readIntFromFile(ASTRINGZ(LUM));
if (lum < 0)
      printf("Error in reading");
else if (lum == UINT_MAX)
      printf("Error in number conversion");
else
{
      printf("Luminosity is %d\n", lum);
}

char rotation[16];
if (adv_copyCharFromFile(ASTRINGZ(GYRO),  rotation, 16) < 0)
      printf("Could not read gyroscopic value\n");
else
{
      printf("Rotation angles are: %s\n", rotation);
}
```

5.3 Synchronization

To use this module you need to include “advSynchronize.h” to your code.

5.3.1 Generalities

Useful functions to synchronize different threads. This module is needed by several other modules but can also be used directly by user.

Calling threads can be blocked until a specific signal is sent. All blocked threads are then released sequentially.

5.3.2 Basic usage

1. Create and initialize a synchronization structure using **adv_syncInit**
2. Block for specific signal using **adv_syncWaitEvent** (with a possible timeout)
3. When thread is released, it will have ownership of synchronization mutex and should release it once not needed.
4. To signal a flag, use **adv_syncNotifyEvent**. This will release all threads blocked on this flag.

5.3.3 Parameters

5.3.3.1 AdvSynchroConf

- mutexAccess: this mutex is owned by the thread returning from **adv_syncWaitEvent**. It should be manually released.

5.3.4 Example

```
// Initialize sync. structure with default flags:
struct AdvSynchroConf strFlags = adv_syncInit(0);
if (videoSource->strFlags == NULL)
{ ////Manage error
}

// From one thread, block until the flag is signaled
adv_syncWaitEvent(strFlags, FLAG_ON, 0);

// When control is gained, don't forget to unlock mutex
pthread_mutex_unlock(&strFlags->mutexAccess);

// Same example with timeout
if (adv_syncWaitEvent(strFlags, FLAG_ON, 1000)<0)
{
      // 1000ms Time out - release mutex and leave
      pthread_mutex_unlock(&strFlags->mutexAccess);
      return -1;
}

// From an other thread, signal a new flag:
adv_syncNotifyEvent(strFlags, FLAG_ON);

// This will cause all threads calling adv_waitEvent on specified flag to return as owner of the
mutex.

// release synchronization structure
adv_syncFree(strFlags);

// See http://linux.die.net/man/3/pthread_cond_signal for more information
```

6 Complete Example

We will demonstrate the portability of Advansee's library by using nearly the same source code on Seeklop and PC. On Seeklop, it will capture an image, compress it to JPEG and send it using Advansee's image stream protocol. On PC, it will receive this stream and display it on screen.

6.1 Source code: main.c

Create a text file named "**main.c**" and paste following text in it. Please replace **seeklopAABBCC** of **SEEKLOP_HOSTNAME variable** with your Seeklop's hostname.

```
// Standard lib
#include <stdio.h>
#include <unistd.h>
#include <getopt.h>
#include <signal.h>
#include <time.h>
#include <sys/time.h>

// Opencv
#include "opencv/cv.h"
#include "opencv/highgui.h"

// Advansee lib headers:
#include "trace.h"
#include "videoSource.h"
#include "videoTcpFlow.h"

#define SEEKLOP_HOSTNAME      "seeklopAABBCC"

int main(int argc, char *argv[])
{

      printf("****************************************\n");
      printf("*              Advansee                *\n");
      printf("* Template for Image Processing Project *\n");
      printf("*                                       *\n");
      printf("****************************************\n");


      /* Alloc structures and variables       */
      struct AdvVideoSource *videoSourceConf = adv_videoSourceCreate();
      if (videoSourceConf == NULL)
      {
            TRACE_ERROR("Could not create video source\n");
            return EXIT_FAILURE;
      }

      IplImage *grabFrame = NULL;
      struct timeval tv;

      struct AdvStreamServer *streamingConf = NULL;
      struct AdvStreamBuffer *dataHeader = NULL;

      /* Frame counter        */
      int imageCnter = 0;
```

```
	/**-------------------------------------------------------------------**/
	/**             INIT STATE: Initialize all variables                  **/
	/**             Only called once at first execution                   **/
	/**-------------------------------------------------------------------**/
	TRACE_INFO("System Init\n");

	/* Set Camera Parameters VGA 30FPS*/
	videoSourceConf->width = 640;
	videoSourceConf->height = 480;
	videoSourceConf->fps = 30;
#ifndef PC
	// On Seeklop or Mipsee, set camera source on video0
	videoSourceConf->filename = "/dev/video0";
	videoSourceConf->videoSourceType = VIDEO_SOURCE_PLATFORM_CAMERA;
	videoSourceConf->imageType = FORMAT_ADV_JPG;
#else
	// On PC, set video source to Tcp Stream
	videoSourceConf->filename = SEEKLOP_HOSTNAME;
	videoSourceConf->videoSourceType = VIDEO_SOURCE_TCP;
	videoSourceConf->imageType = FORMAT_ADV_BGR;
#endif

	/* Init structures */
	/* Ipl Image to receive color BGR*/
	if ((grabFrame = cvCreateImageHeader( cvSize(videoSourceConf->width, videoSourceConf->height),
IPL_DEPTH_8U, 3)) == NULL)
	{
		TRACE_ERROR("Memory allocation failed for %s in %s\n", "grabFrame", __func__);
		return (-1);
	}

	/* Video Source */
	if (adv_videoSourceInit(videoSourceConf)<0)
	{
		TRACE_ERROR("Could not initialize video source\n");
		return EXIT_FAILURE;
	}
#ifndef PC
	// On mipsee or Seeklop:

	/* Create Streaming server */
	streamingConf = adv_streamCreate(STREAM_PORT);

	/* Streaming data structure */
	dataHeader = adv_streamCreateImageHeader(videoSourceConf->imageType,
                                             videoSourceConf->width,
                                             videoSourceConf->height,
                                             (videoSourceConf->height*videoSourceConf->width*3)
                                             );
#endif

	/* Go to next state */
	TRACE_INFO("System Operationnal\n");

/**-------------------------------------------------------------------------------------------**/
/**           OPERATIONAL STATE: Perform image processing loop                                **/
```

```
/**           This state is called for each image processed using the while(1) loop           **/
/**-------------------------------------------------------------------------------------------**/
int exit=0;
while(!exit)
{

      /* Get Frame and check error code */
      if (adv_videoSourceGetFrame(videoSourceConf, grabFrame)!=0)
      {
              TRACE_INFO("Image Capture Error\n");
              // Try again
              break;
      }

#ifndef PC
      /* On Mipsee or Seeklop, send image via TCP */
      memcpy(dataHeader->imagePtr, (unsigned char*)grabFrame->imageData, grabFrame->imageSize);
      gettimeofday(&tv, NULL);
      adv_streamNewData(streamingConf, dataHeader, grabFrame->imageSize, &tv, NULL, NULL);
#endif


#ifdef PC
      /* On PC, display image */
      cvShowImage("Test", grabFrame);
      char esc ='1';
      esc = cvWaitKey(33);
      if (esc == 'q')
              exit = 1;
#endif

      /* Increment image counter      */
      imageCnter++;
}


/**-------------------------------------------------------------------------------------------**/
/**   UNINIT STATE : Release all memory allocated during                                      **/
/**           the init phase and exit program                                  **/
/**-------------------------------------------------------------------------------------------**/
      TRACE_INFO("System Uninit\n");
      /* release all structures */
      adv_videoSourceClose(videoSourceConf);
#ifndef PC
      adv_streamClose(streamingConf);
      adv_streamReleaseImageHeader(dataHeader);
#endif
      cvReleaseImageHeader(&grabFrame);

      printf("Application Ends:\n");

      /* end application */
      return 0;

} // main
```

6.2 Project compilation

Please create an executable file named "**compile.sh**" and paste following text in it. Set all variables in bold with correct path to indicated directory:

```
#!/bin/bash

# Directory containing Advansee's library headers
LIBINCLUDEDIR=

# Compile For PC
# Directory containing libadvansee.a for PC
LIBBINPCDIR=
gcc -c -DPC -I$LIBINCLUDEDIR  main.c
gcc -o mainPc main.o -L$LIBBINPCDIR  -ladvansee -lpthread -lopencv_highgui -lopencv_imgproc
-lopencv_core

# Compile for Seeklop
# Directory containing libadvansee.a for Seeklop
LIBBINSEEKDIR=
# Seeklop staging dir:
SYSROOT=
# Seeklop compile toolchain:
CC=arm-advseeklop-linux-gnueabi
$CC-gcc -c -DSEEKLOP -I$LIBINCLUDEDIR  --sysroot=$SYSROOT main.c
$CC-gcc -o mainSeeklop main.o --sysroot=$SYSROOT -L$LIBBINSEEKDIR  -lipu -lvpu -ladvansee -lpthread
-lopencv_highgui -lopencv_imgproc -lopencv_core
```

6.3 Project test

Put main.c and compile.sh in same directory. Execute "compile.sh" to compile code.

Copy executable "mainSeeklop" to Seeklop platform using command (replace **seeklopAABBCC** by your Seeklop hostname):

```
scp -p mainSeeklop root@seeklopAABBCC:/root
```

Connect to your Seeklop platform in SSH and execute application:

```
ssh root@seeklopAABBCC
./mainSeeklop
```

Start PC application and receive frames:

```
./mainPC
```