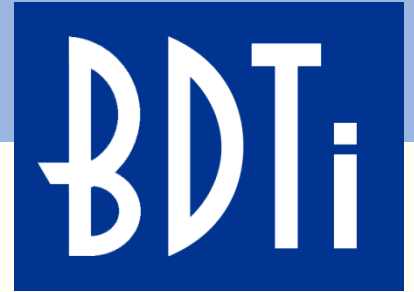


*The most trusted source of analysis, advice, and engineering  
for embedded processing technology and applications*



# **Accelerating Computer Vision Applications With the Hexagon™ DSP**

Berkeley Design Technology, Inc.  
Walnut Creek, California USA  
+1 (925) 954-1411

info@BDTI.com  
<http://www.BDTI.com>

Copyright © 2013 Berkeley Design Technology, Inc.

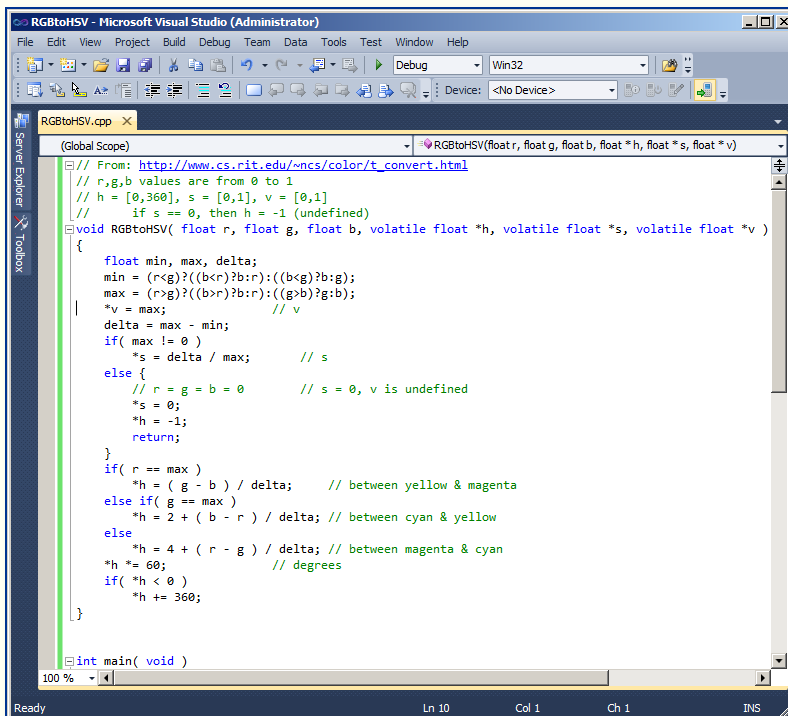
**Copyright © 2013 Berkeley Design Technology, Inc.  
Includes material copyrighted by Qualcomm Corporation  
(© 2013 Qualcomm Corporation).**

**Used by permission of Qualcomm Corporation.**

**No reproduction of this workshop material is permitted  
without the express written consent of Berkeley Design  
Technology, Inc. and Qualcomm Corporation.**

# Computer Vision Acceleration Made Easy

It took an Android/C++ developer and a DSP engineer slightly over a week to port a custom algorithm to the Hexagon DSP, implement multi-threading, and incorporate the algorithm as part of a complete Android application.



```
RGBtoHSV.cpp
(Global Scope)
// From: http://www.cs.rit.edu/~mcs/color/t\_convert.html
// r,g,b values are from 0 to 1
// h = [0,360], s = [0,1], v = [0,1]
// if s == 0, then h = -1 (undefined)
void RGBtoHSV( float r, float g, float b, volatile float *h, volatile float *s, volatile float *v )
{
    float min, max, delta;
    min = (r<g)?((b<r)?b:r):((b<g)?b:g);
    max = (r>g)?((b>r)?b:r):((g>b)?g:b);
    *v = max; // v
    delta = max - min;
    if( max != 0 )
        *s = delta / max; // s
    else {
        // r = g = b = 0 // s = 0, v is undefined
        *s = 0;
        *h = -1;
        return;
    }
    if( r == max )
        *h = ( g - b ) / delta; // between yellow & magenta
    else if( g == max )
        *h = 2 + ( b - r ) / delta; // between cyan & yellow
    else
        *h = 4 + ( r - g ) / delta; // between magenta & cyan
    *h *= 60; // degrees
    if( *h < 0 )
        *h += 360;
}
int main( void )
```

2-3  
man  
weeks

Android SDK/NDK

Hexagon DSP



## About BDTI

- The industry's trusted source of technical analysis, advice, and engineering services for embedded technology and applications
  - A highly trusted partner—consistently delivering projects that are right the first time, on time and on budget
- Extensive, hands-on experience in embedded software development
  - 20 years creating highly optimized embedded code
  - Deep skill in building embedded signal processing applications: voice, audio, video, and vision
  - Expert in vision applications, algorithms, and tools—including OpenCV
- A committed Qualcomm partner that has delivered numerous complex Hexagon projects for products on the market today

## What You Will Learn in This Presentation

- Object Segmentation Using Color—The HSV Color Space
- Overview of the Hexagon DSP and Hexagon SDK
- Introduction to FastCV (*CV = Computer Vision*)
- Passing Frames From the CPU to the DSP Using FastRPC
- Accelerating an Algorithm Using the Hexagon DSP
- Combining Android and Hexagon DSP Builds to Produce an Android Application

# Object Segmentation Using Color

## Object Segmentation Using Color

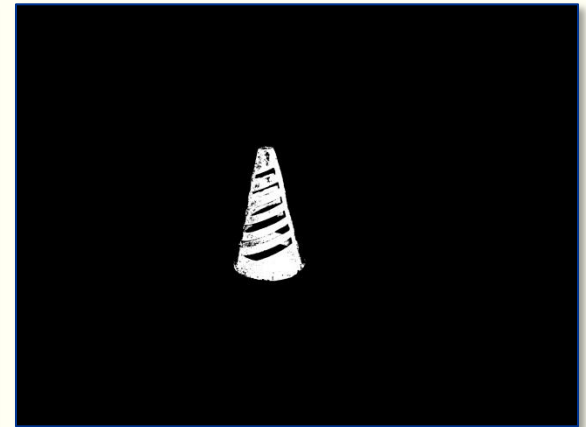
- Image segmentation is the process of changing the representation of an image into something that is easier to analyze
- It is commonly used to isolate objects or highlight boundaries in images by assigning a label to every pixel such that pixels with the same label share certain visual characteristics. (Adapted from [http://en.wikipedia.org/wiki/Image\\_segmentation](http://en.wikipedia.org/wiki/Image_segmentation))



Input RGB image



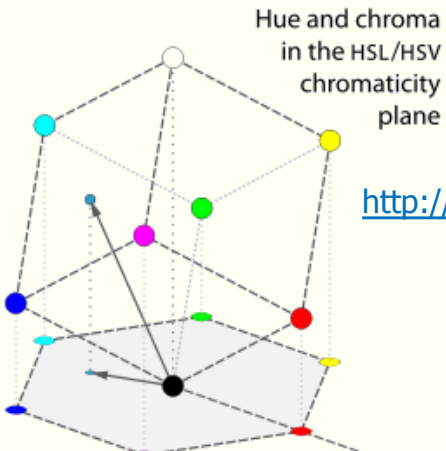
H channel after  
HSV conversion



Thresholded  
HSV image

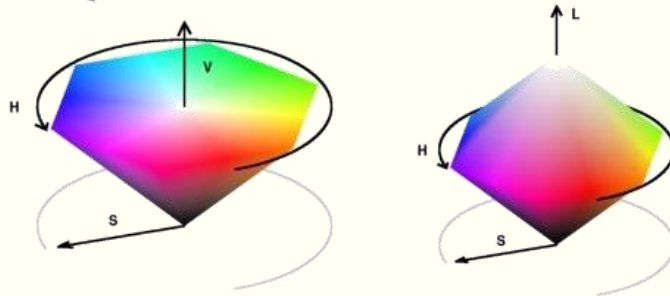
## HSV Color Space

HSV is a cylindrical color coordinate model. HSV is derived by projecting a tilted RGB cube onto the “chromaticity plane” perpendicular to the neutral axis



[http://en.wikipedia.org/wiki/HSL\\_and\\_HSV](http://en.wikipedia.org/wiki/HSL_and_HSV)

<http://www.getreuer.info>



HSV stands for *Hue, Saturation, Value*



RGB

H channel





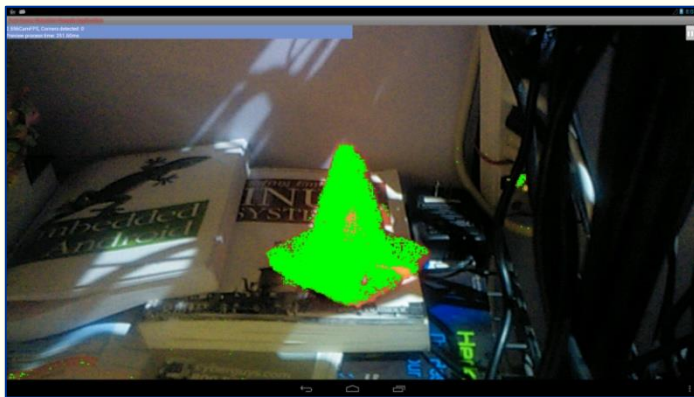
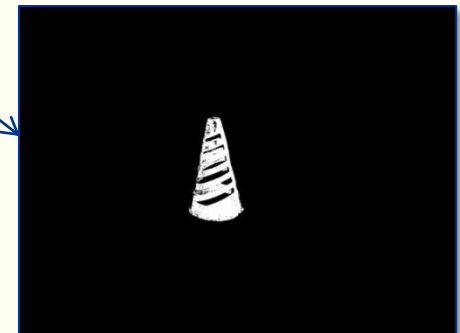
# Object Segmentation Using the HSV Color Model on Android

Segment image based on a specified color range.  
The Android framework uses YUV420sp for frame formatting

## Algorithm Steps

1. YUV to RGB
2. RGB to HSV (custom function)
3. HSV range → binary image (custom function)

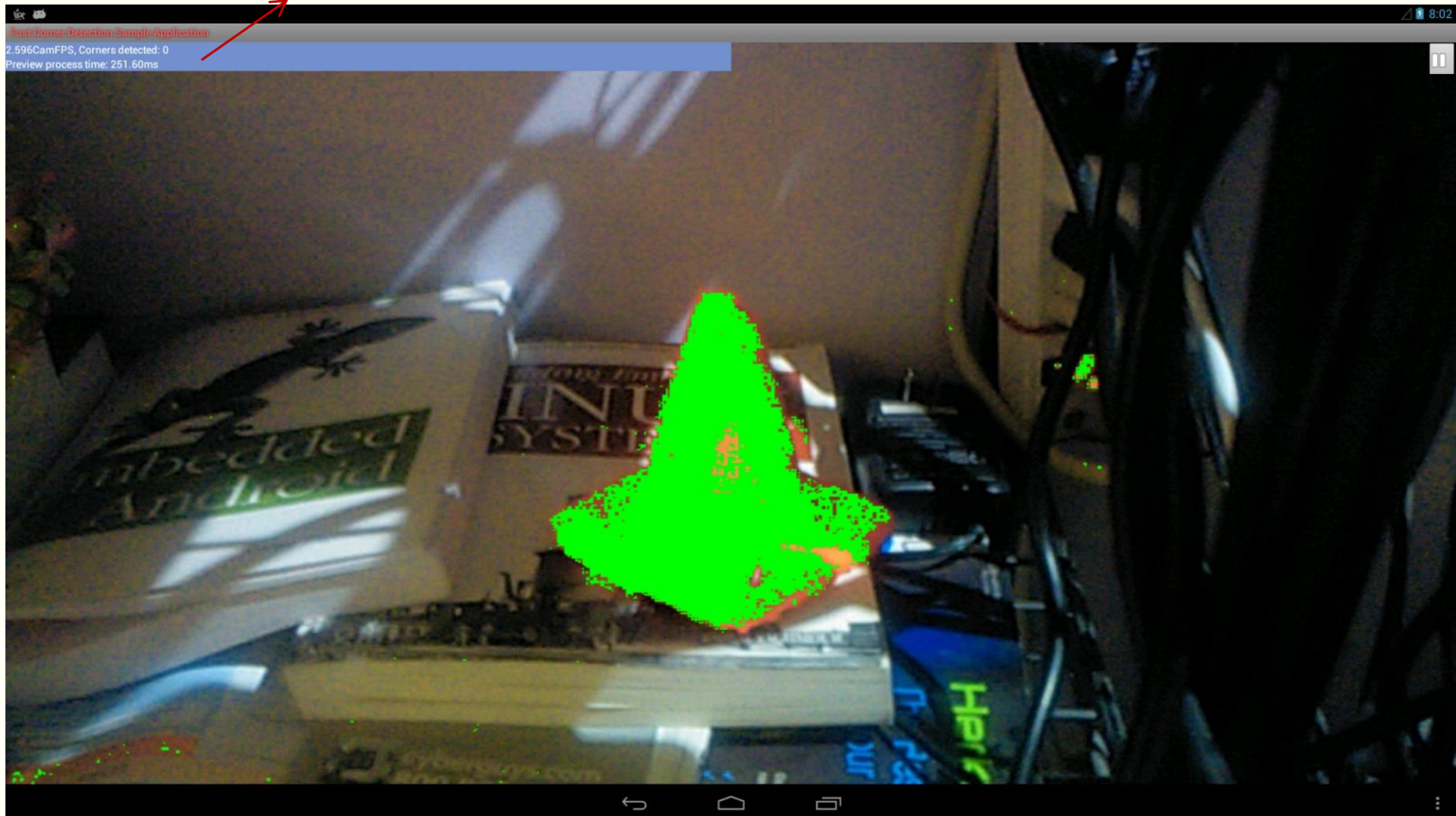
Binary image overlaid on image



# Object Segmentation Using the HSV Color Model on Android

2.596CamFPS  
Preview process time: 251.60ms

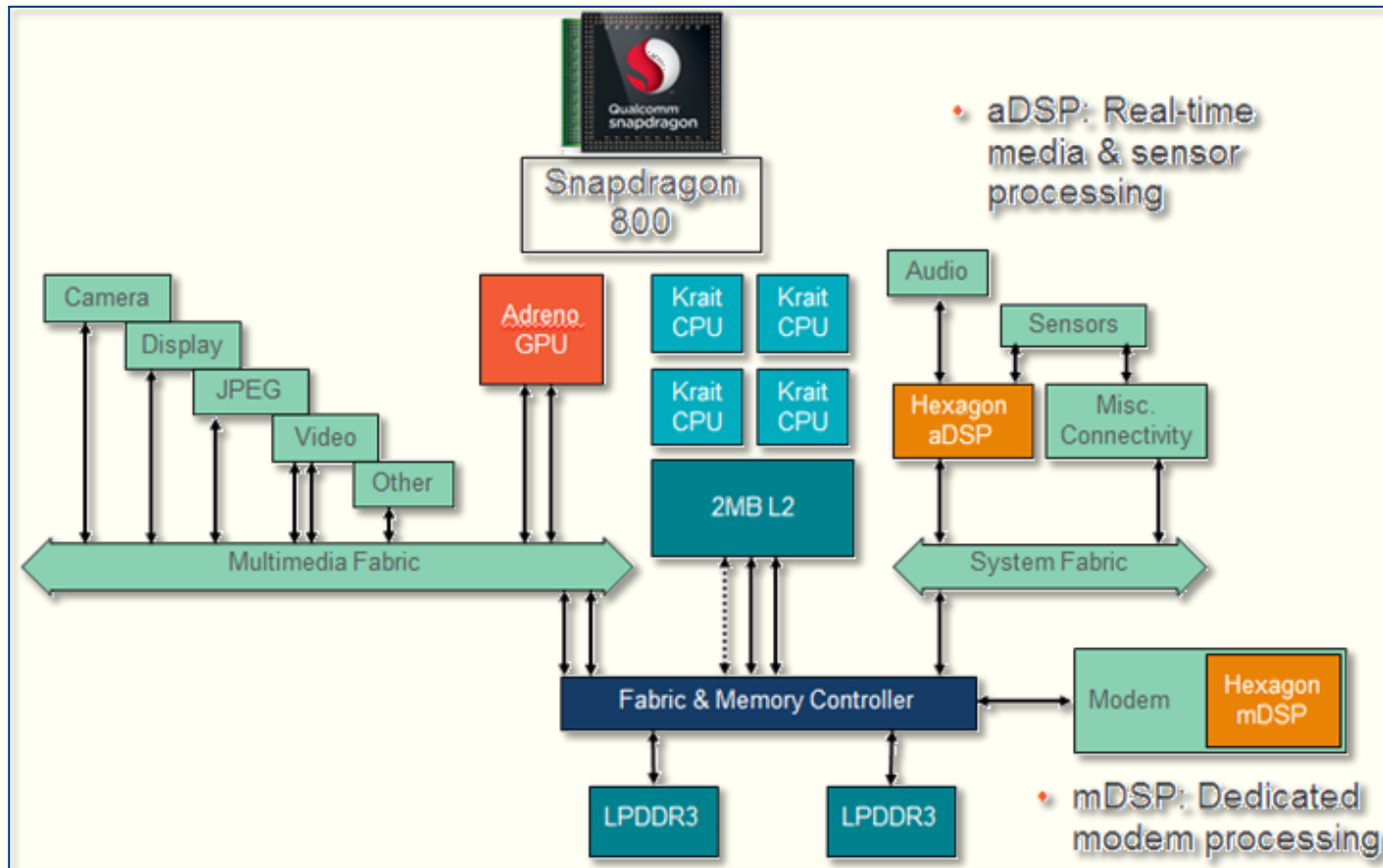
Running on CPU



# The Qualcomm Hexagon™ DSP

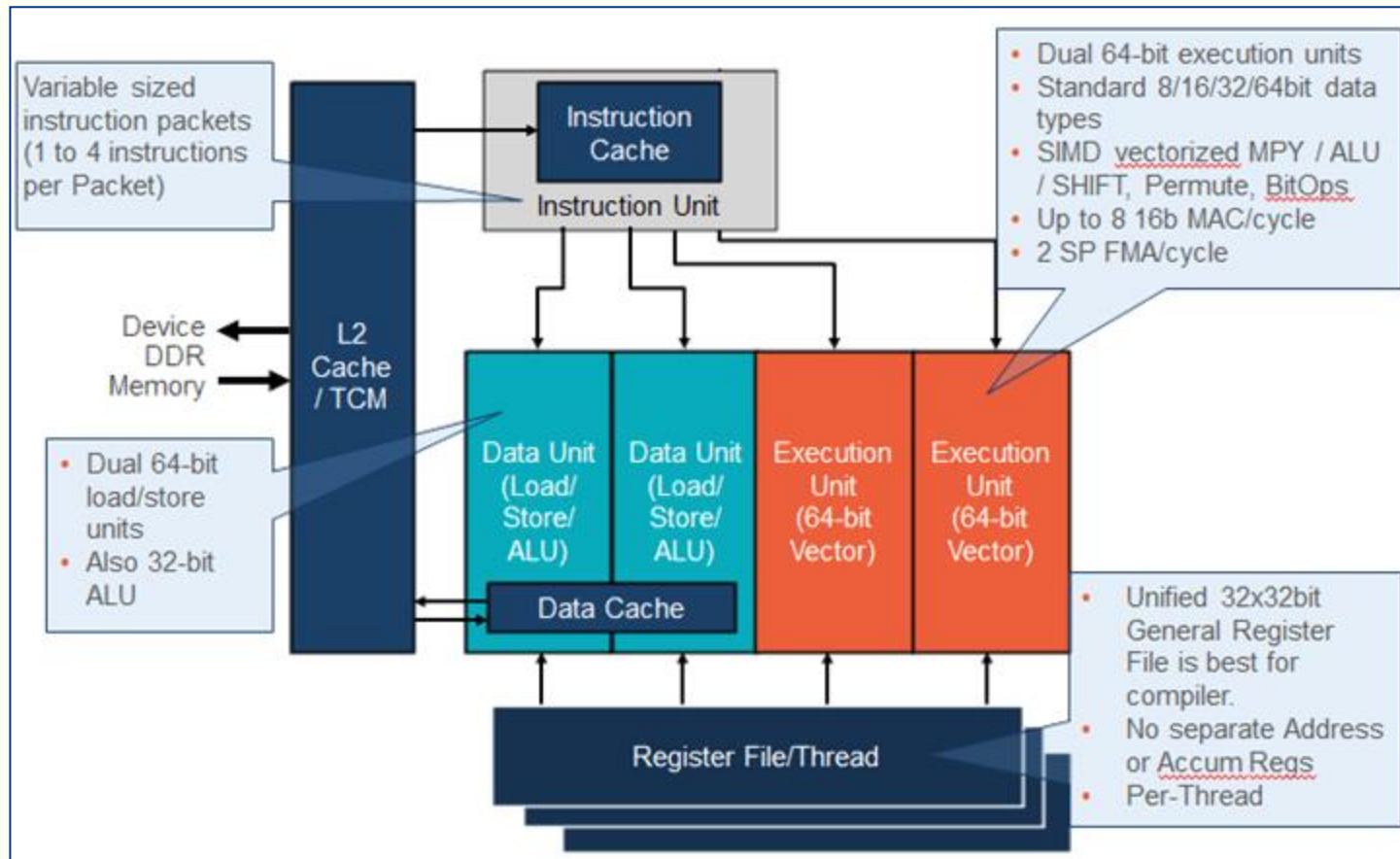
# The Qualcomm Hexagon DSP

## Hexagon DSP Processors in Snapdragon™ Products



# The Qualcomm Hexagon DSP

## VLIW: Area & Power efficient multi-issue



## Terminology and Components Used to Create This Demo

## Terminology and Components Used to Create This Demo

We used three different SDKs (Software development Kits) to create this demo:

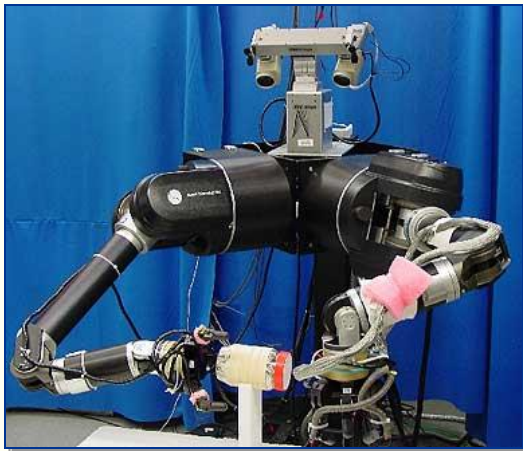
1. FastCV SDK
  - Downloaded from the Qualcomm developers support website
  - Provides a CPU only framework for capturing images from the camera and rendering images to the LCD
  - Includes the FastCV ARM libraries
2. Android SDK and NDK—Required to build the FastCV SDK
3. Hexagon SDK
  - Downloaded from the Qualcomm developers support website
  - Provides two important frameworks used for this project:
    - FastCV using a Hexagon optimized library
    - downscaleBy2—A multi-threaded example

## FastCV SDK

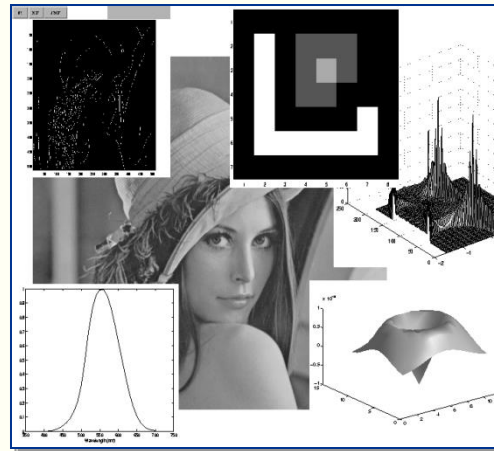


## Introduction to FastCV—FastCV Overview

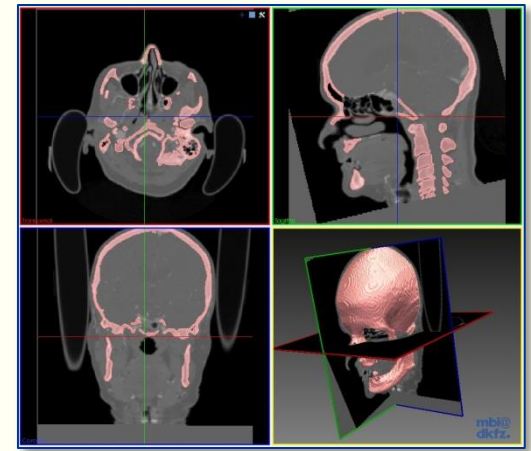
- FastCV is an API & library which is designed to enable Real-Time Computer Vision (CV) applications
- FastCV enables mobile devices to run CV applications efficiently
- FastCV allows developers to hardware accelerate their CV application
- FastCV is comparable to OpenGL ES in the rendering domain
- FastCV is a clean modular library



[Source](#)



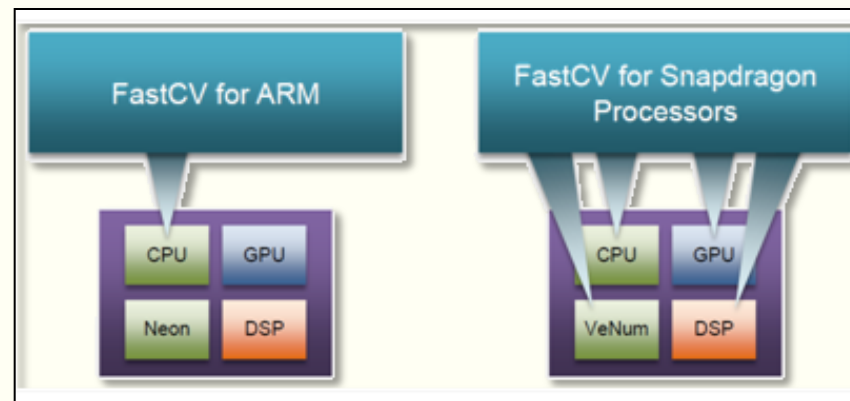
[Source](#)



[Source](#)

## Introduction to FastCV—FastCV Overview

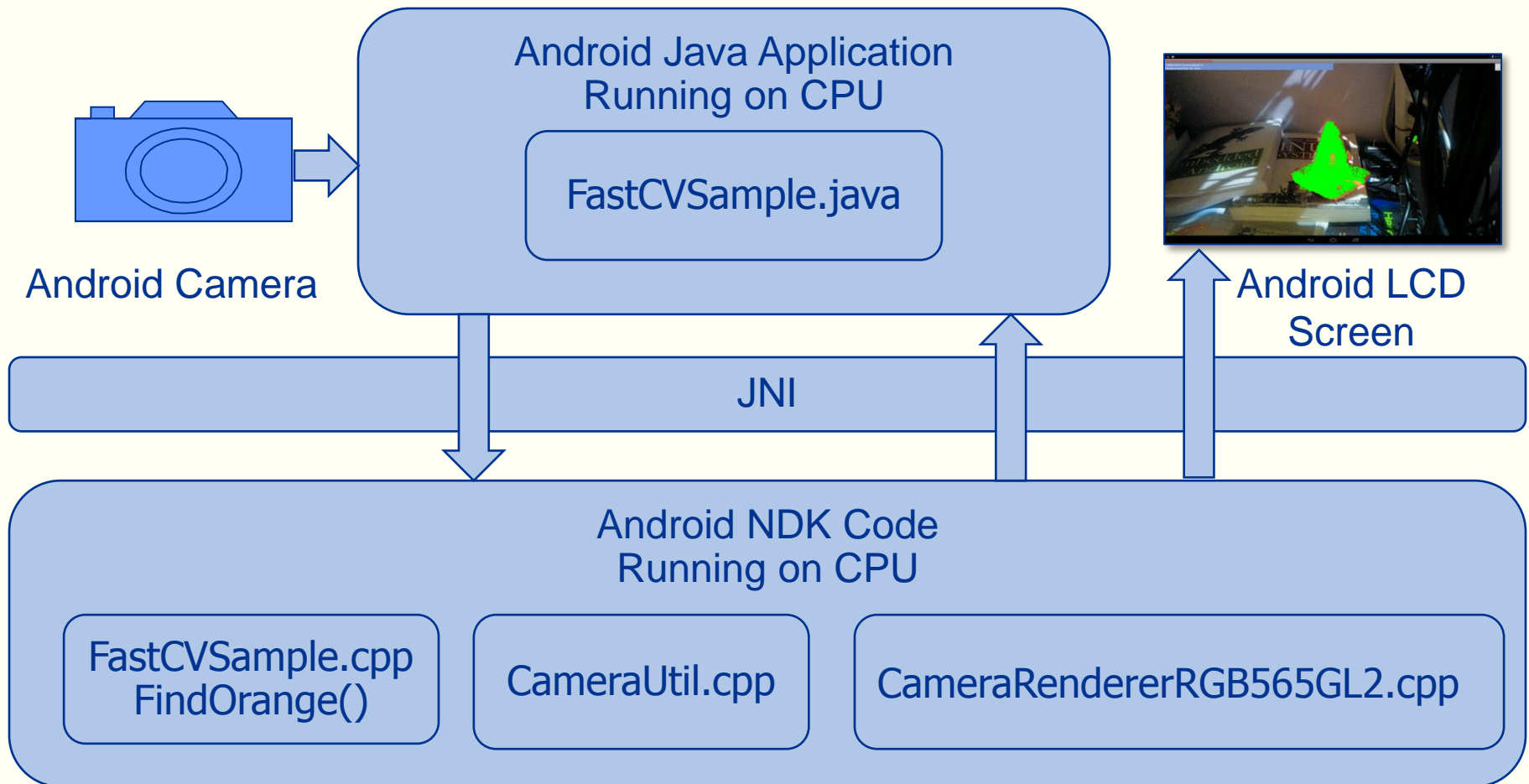
- FastCV is a low-level CV library optimized for ARM and Snapdragon
- The FastCV library contains 391 CV operators, all have been ported to Hexagon
- 174 of the 391 CV operators have been hand-optimized for the Hexagon
- The FastCV SDK includes examples/frameworks for integrating FastCV into Android applications



One Development Kit—Two Implementations

## **Implementing the Algorithm on the Android FastCV Framework (CPU Only)**

## FastCV SDK Example

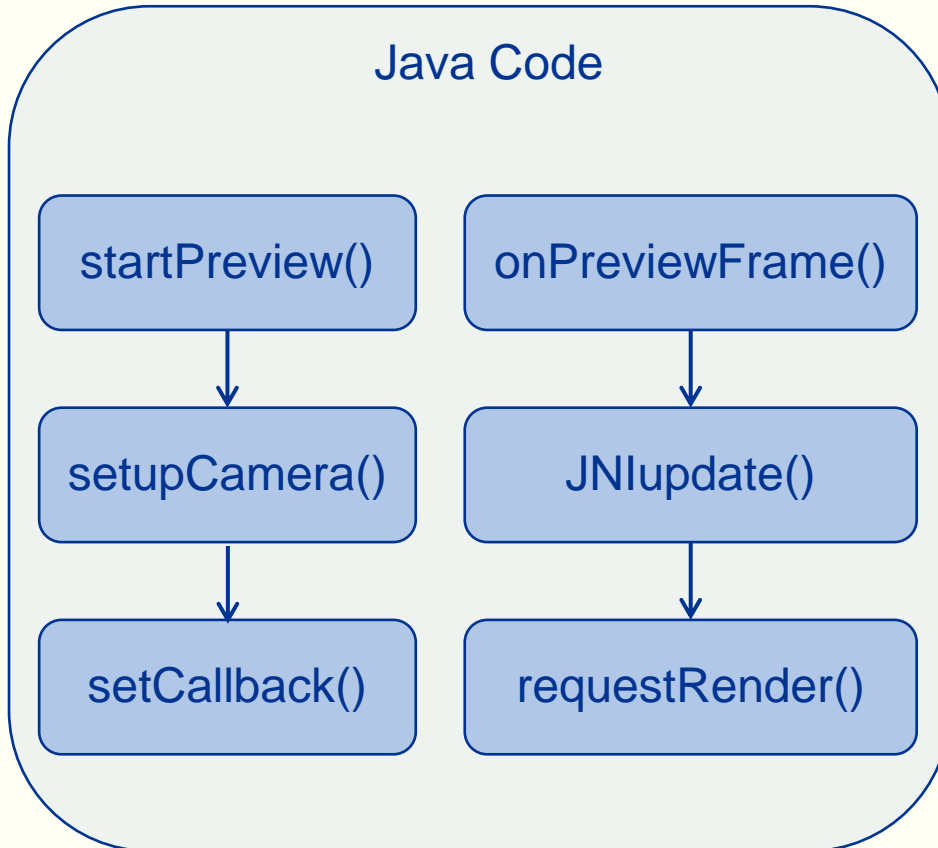


## Implementing the Algorithm on the Android FastCV Framework

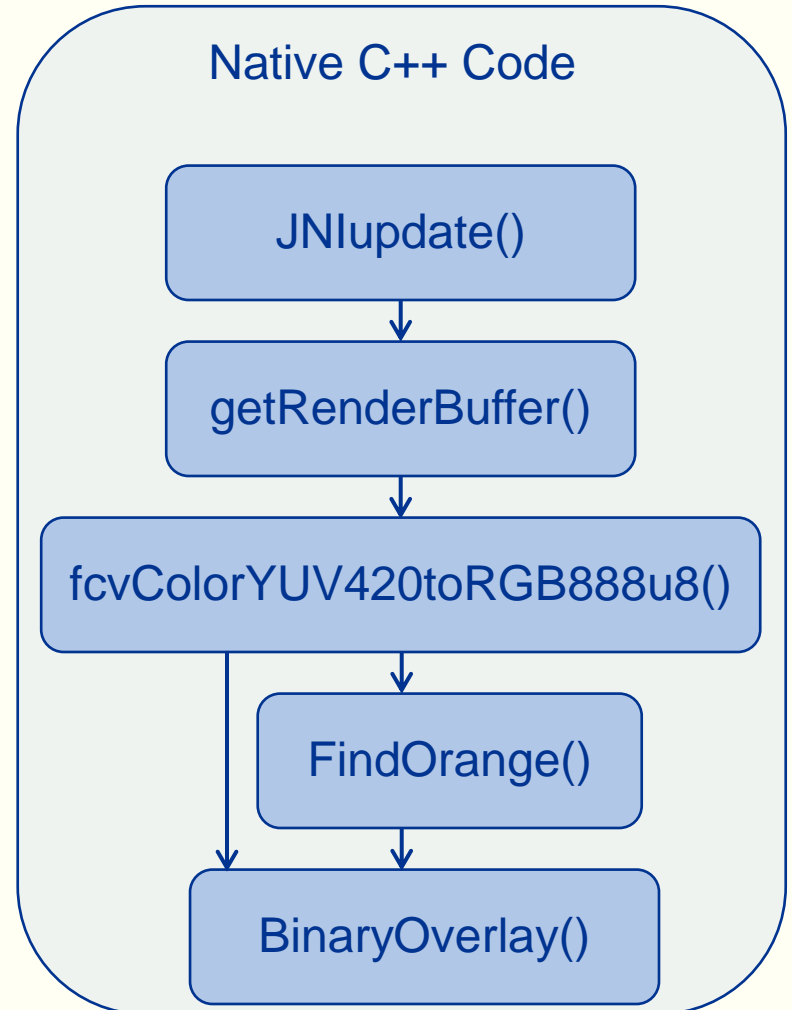
Module	Function
FastCVSample.java	setupCamera(), onPreviewFrame(), setCallback()
FastCVSample.cpp	FindOrange(), JNIupdate(), JNIinit(), JNIconeup()
CameraUtil.cpp	OpenGL shader
CameraRendererRGB565GL2.cpp	RGB565 renderer

# Implementing the Algorithm on the Android FastCV Framework

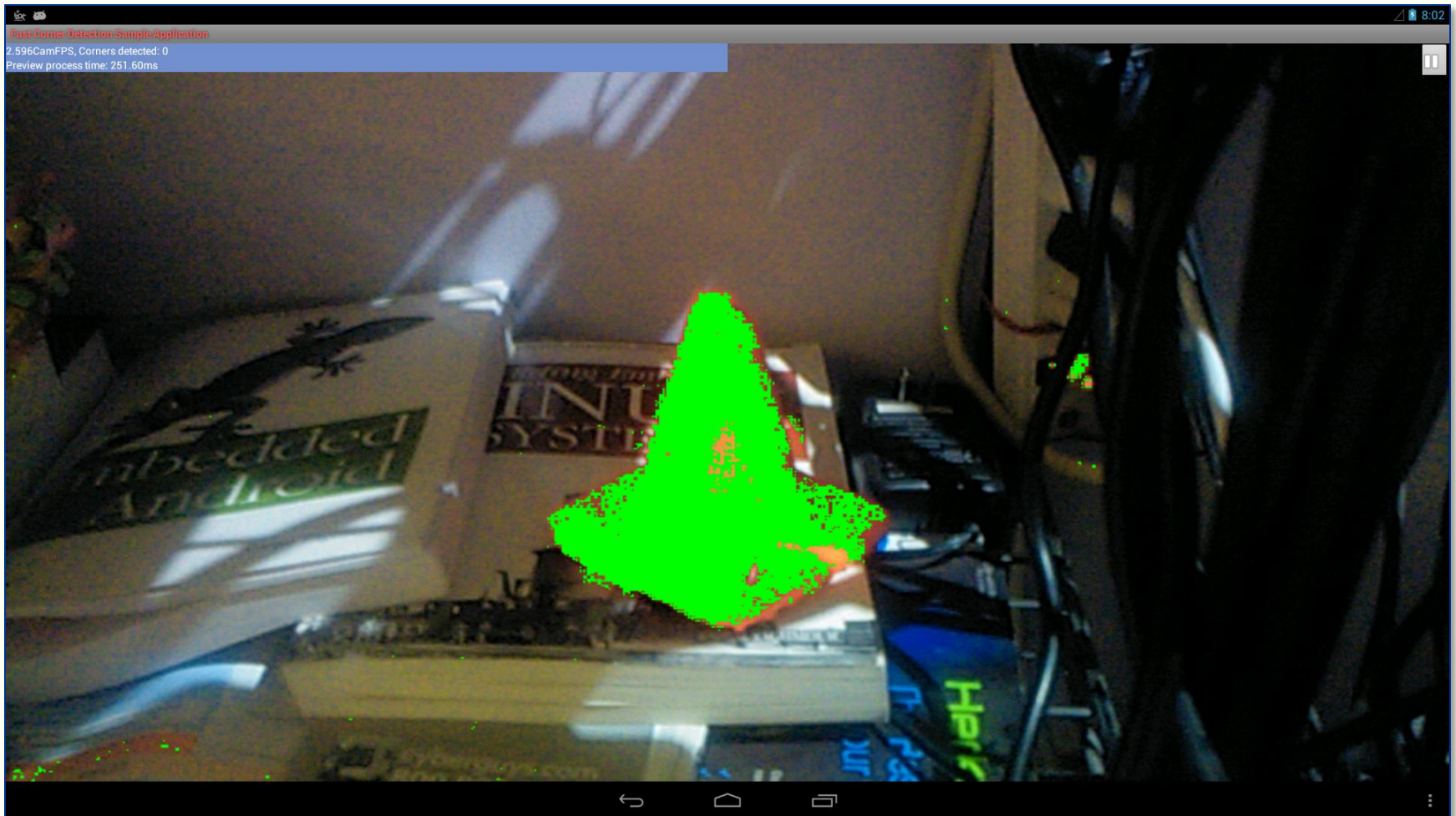
## Java Code



## Native C++ Code



## Demo Running on CPU



## Passing Frames From the CPU to the DSP Using FastRPC



## Passing Frames From the CPU to the DSP Using FastRPC

Android Java Application  
Running on CPU

FastCVSample.java

CameraSurface.java

FastCVSampleRenderer.java

JNI

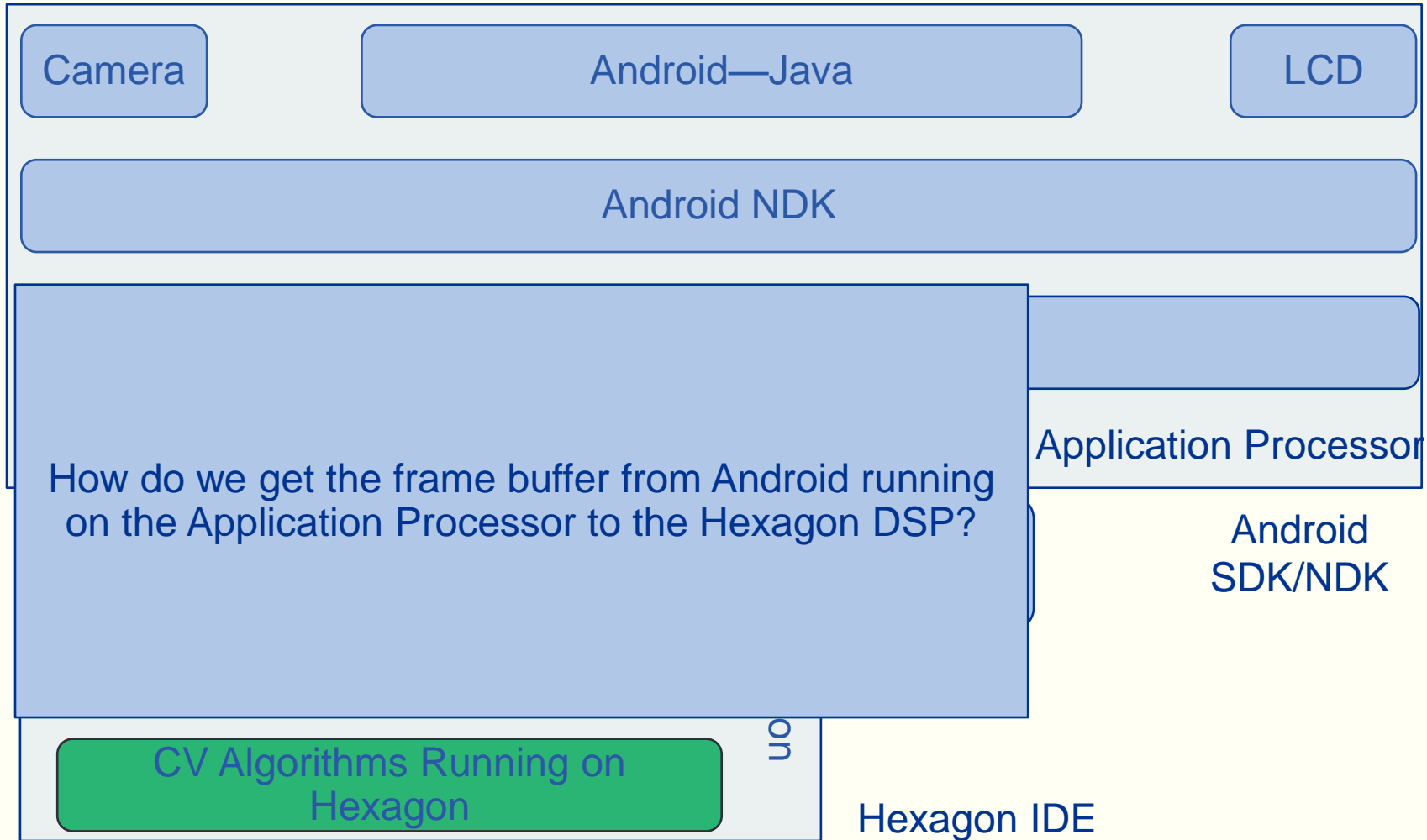
Android NDK Code  
Running on CPU

FastCVSample.cpp  
FindOrange()

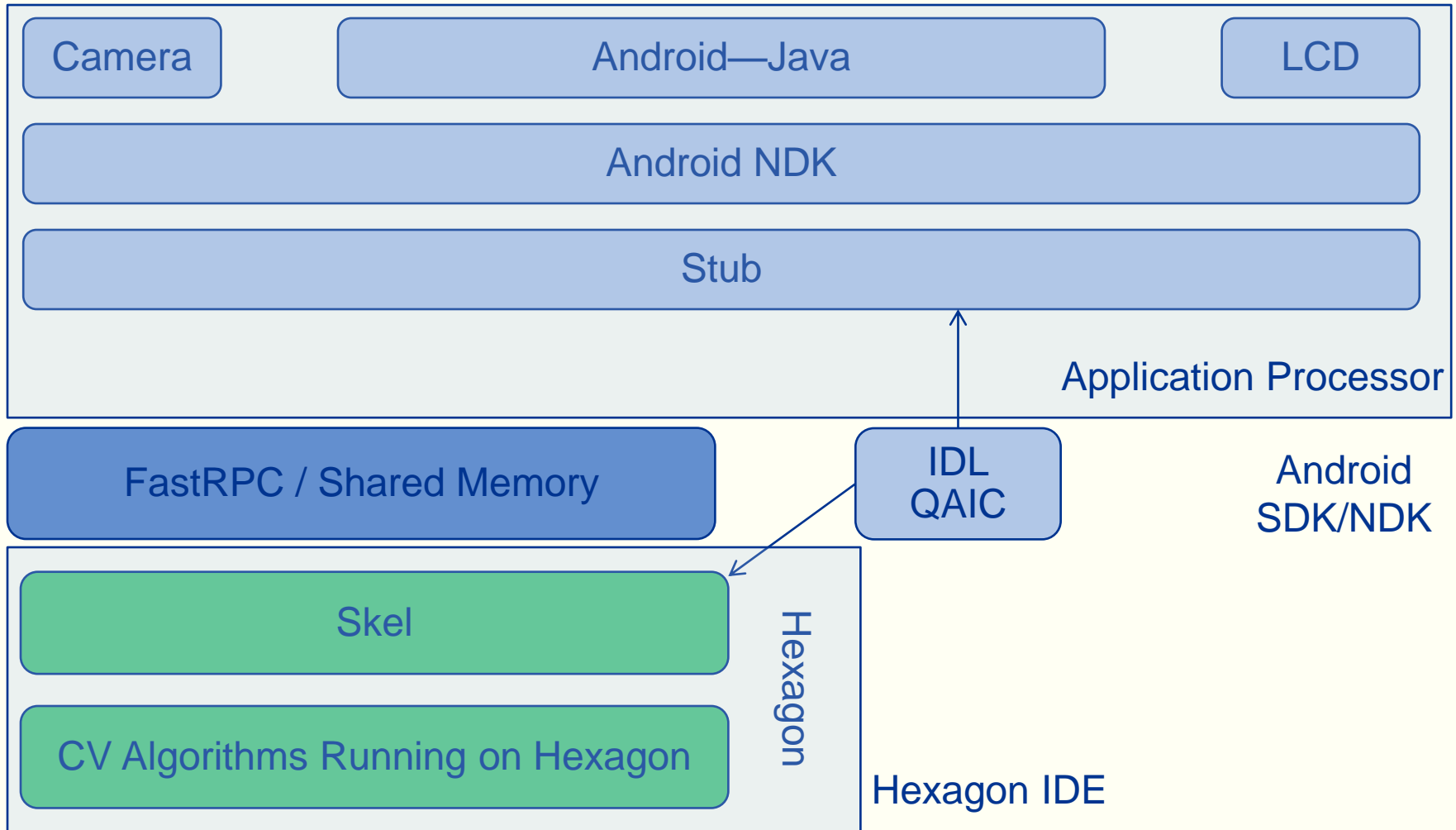
CameraUtil.cpp

CameraRendererRGB565GL2.cpp

# Hexagon Accelerated Computer Vision Block Diagram

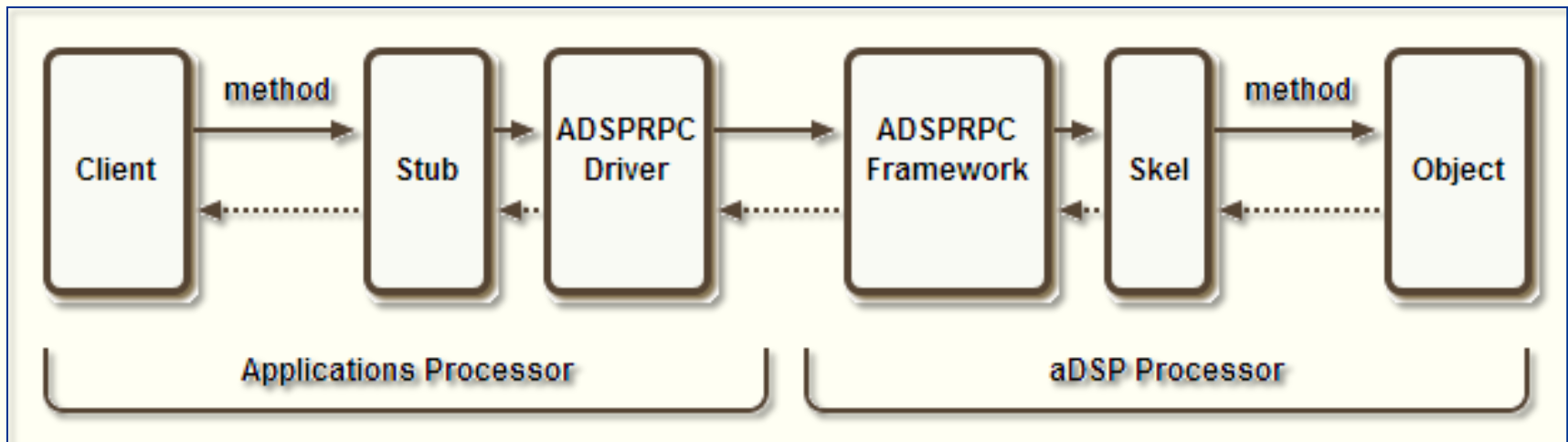


# Hexagon Accelerated Computer Vision Block Diagram

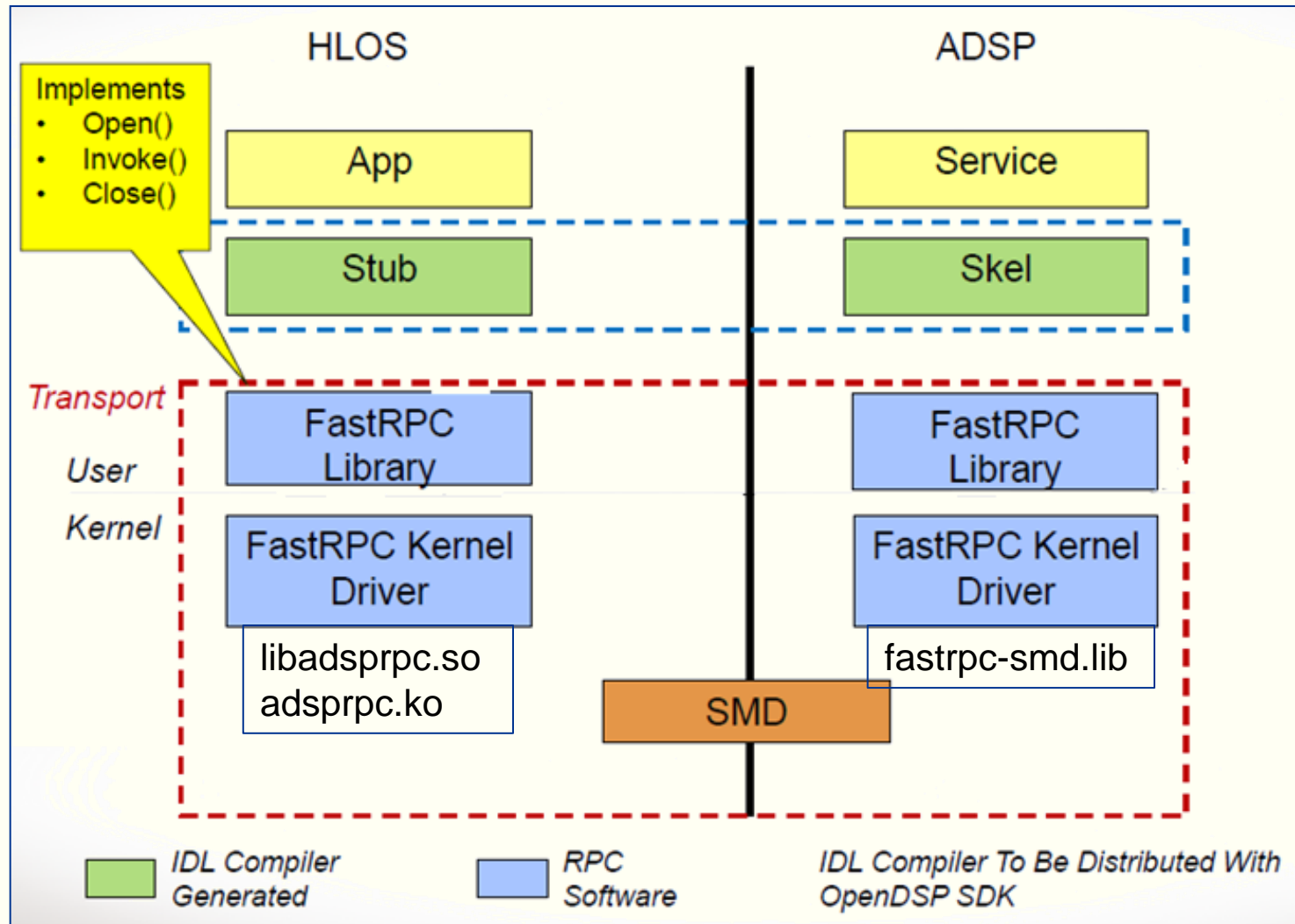


## What is Fast RPC?

- Enables calling functions between application processor and DSP
- Synchronous
- Zero Copy – Only when using ION buffers
- Ensures cache coherency for input and output buffers
- Each HLOS thread is handled on its own unique thread on the DSP
- Threads are automatically cleaned up on HLOS process exit



## What is Fast RPC?



## Dynamic Loading

The Remote File System is used by the loader on the Hexagon aDSP to read shared object files. The shared object files are stored on the HLOS's file system

- On these Android builds the remote file system is implicitly implemented by libadsprpc.so
- It uses the calling processes context to open files
- The default file system root directory is /system/lib/rfsa/adsp
- The optional environment variable ADSP\_LIBRARY\_PATH can be used to override the default file system root directory. It contains a list of directories to search when dlw\_Open(<library>) is called

# Implementing the Algorithm on the Hexagon Framework Using the Hexagon SDK

## Modifying the Example to Produce Your Own CV Primitive

### Two BDTI engineers involved in this project

- An Android/C++ developer with computer vision algorithm experience
- A DSP expert with some Hexagon experience

We were able to put together this demo with very little direct Qualcomm support. The documentation and examples were of very high quality and easy to modify to fit our requirements

### Steps to Creating Demo

1. Create and validate algorithm using MATLAB, Visual Studio, and OpenCV
2. Port algorithm into Hexagon downscaleBy2 example and test using simulator and file I/O (Load and Store static images to filesystem)
3. Merge algorithm+downscaleBy2 code into cornerApp example to include FastCV library and Android FastCV framework support

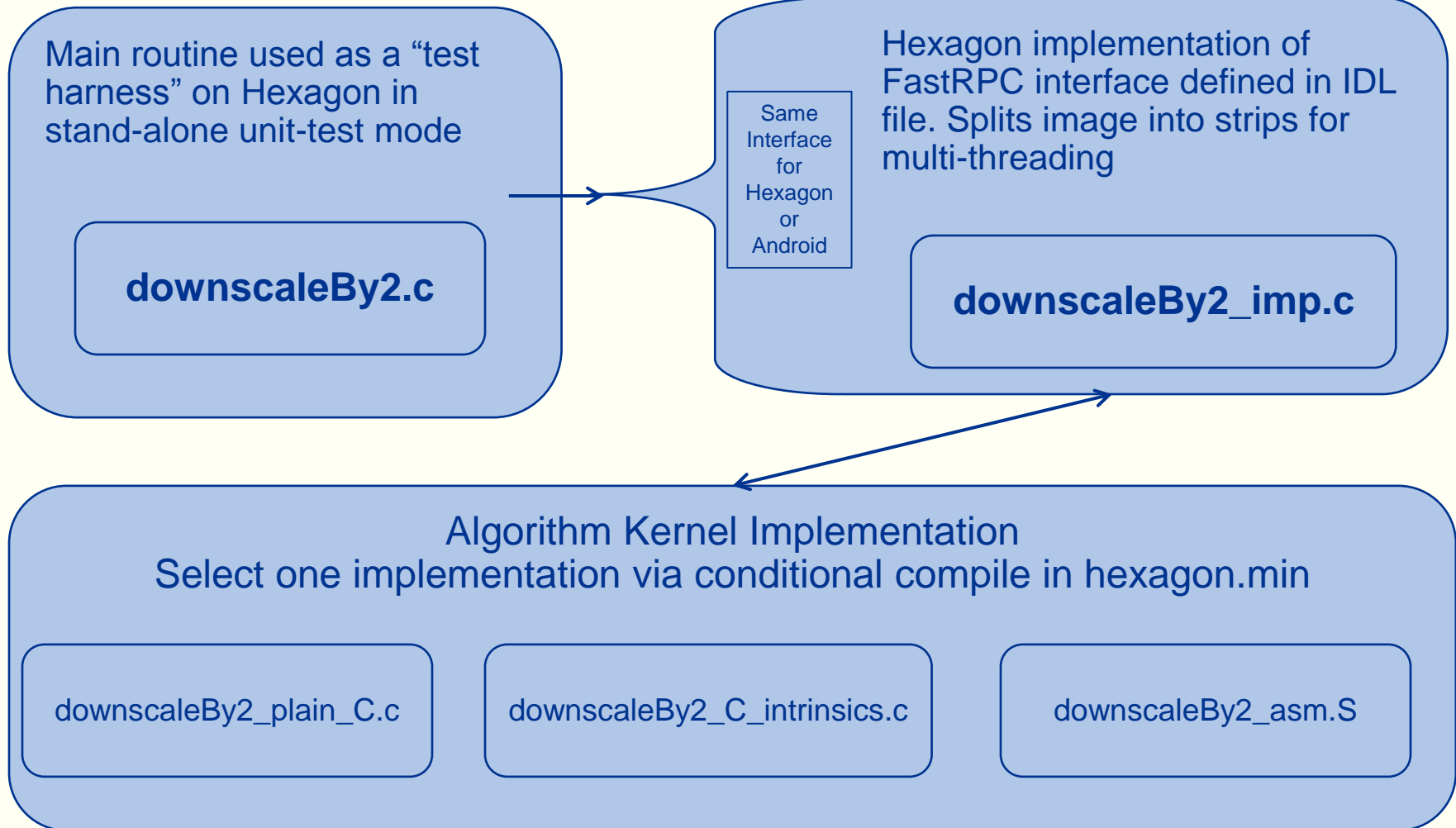


## downscaleBy2 Example

The downscaleBy2 example shows the following from a Hexagon perspective:

- Integration of a custom algorithm onto Hexagon (using FastRPC)
  - A function not currently available in the FastCV library
- Build environment where the (main) downscaleBy2.c file can be:
  - Compiled for Hexagon only—Creates unit-testable, stand-alone Hexagon implementation
  - Compiled on ARM—Creates ARM + Hexagon implementation where algorithm is invoked via RPC call
- Example algorithm Hexagon implementations: Plain C, C with Intrinsic, assembly language
- Optimization utilizing multiple threads
- Use Hexagon simulator to assess performance for various scenarios in Hexagon stand-alone unit-test mode

## downscaleBy2 Example—Hexagon Stand-Alone Unit-Test



## Using the Hexagon Simulator, File I/O, and Static Images for Debugging

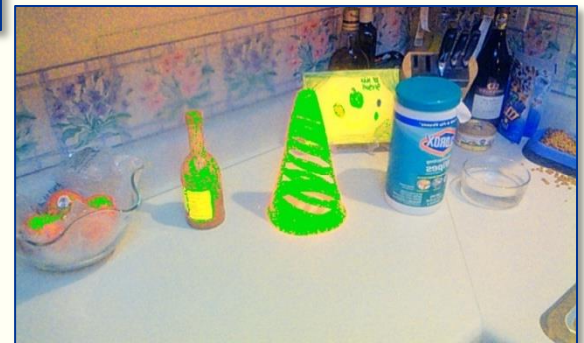


Add file I/O to `downscaleBy2.c` to use BMP files as the input and output

Hexagon simulator supports standard `fopen()`, `fclose()`, `fprintf()`, `fscanf()`



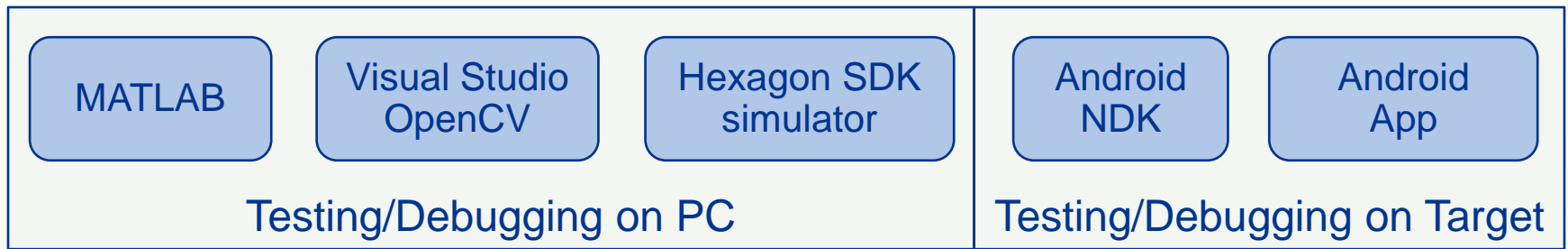
Hexagon unit-test, stand-alone mode on the Hexagon simulator



## Testing and Debugging

Testing and debugging was done in multiple stages:

1. Using MATLAB to verify the algorithm
2. Using Visual Studio and OpenCV on a PC to verify the C implementation of the algorithm
3. After porting to the Hexagon downscaleBy2 example, the Hexagon simulator was used with static images
4. Testing on hardware as an Android NDK application (called from the Android shell)
5. After porting to the cornerApp example, the algorithm was tested using the Android camera and the FastCV Android framework



## Using Everything We Learned to Create the Demo

## Incorporating Your CV Primitive Into the Android Framework

- Combining the **fastcorner** example from the FastCV SDK and the **cornerapp** example from the Hexagon SDK
- FastCV fastcorner SDK provides Android camera to NDK video stream path. After initializing the camera, it uses a JNI call to pass the frame to the NDK level where the FastCV function RUNS ON THE CPU
- We replace the JNI function in the NDK from the FastCV SDK example with the JNI function from the NDK in the Hexagon SDK cornerapp example. The Hexagon SDK example RUNS ON THE HEXAGON

# Incorporating Your CV Primitive Into the Android Framework

Android Java Application  
Running on CPU

FastCVSample.java

CameraSurface.java

FastCVSampleRenderer.java

JNI

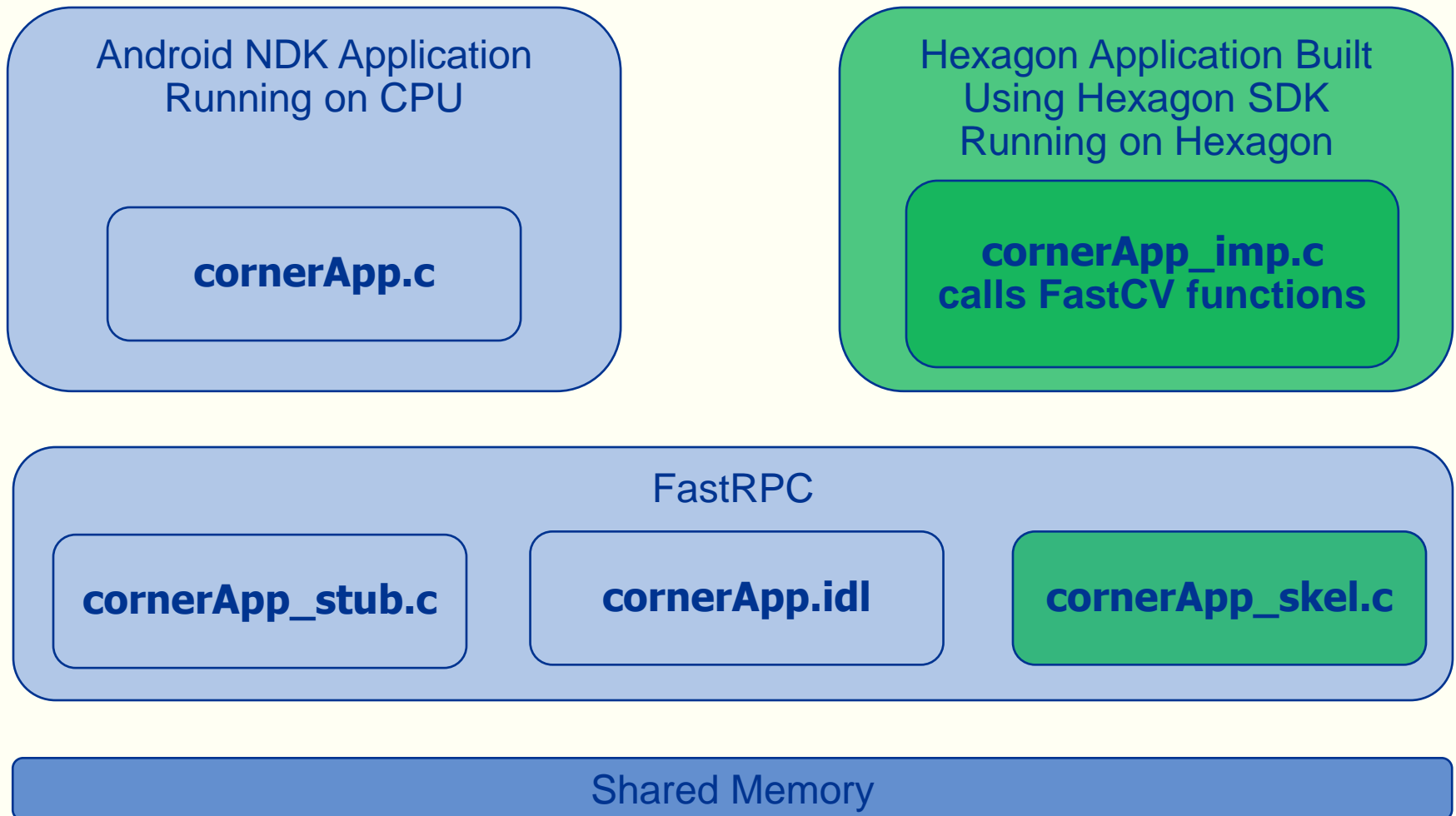
Android NDK Code  
Running on CPU

FastCVSample.cpp  
FindOrange()

CameraUtil.cpp

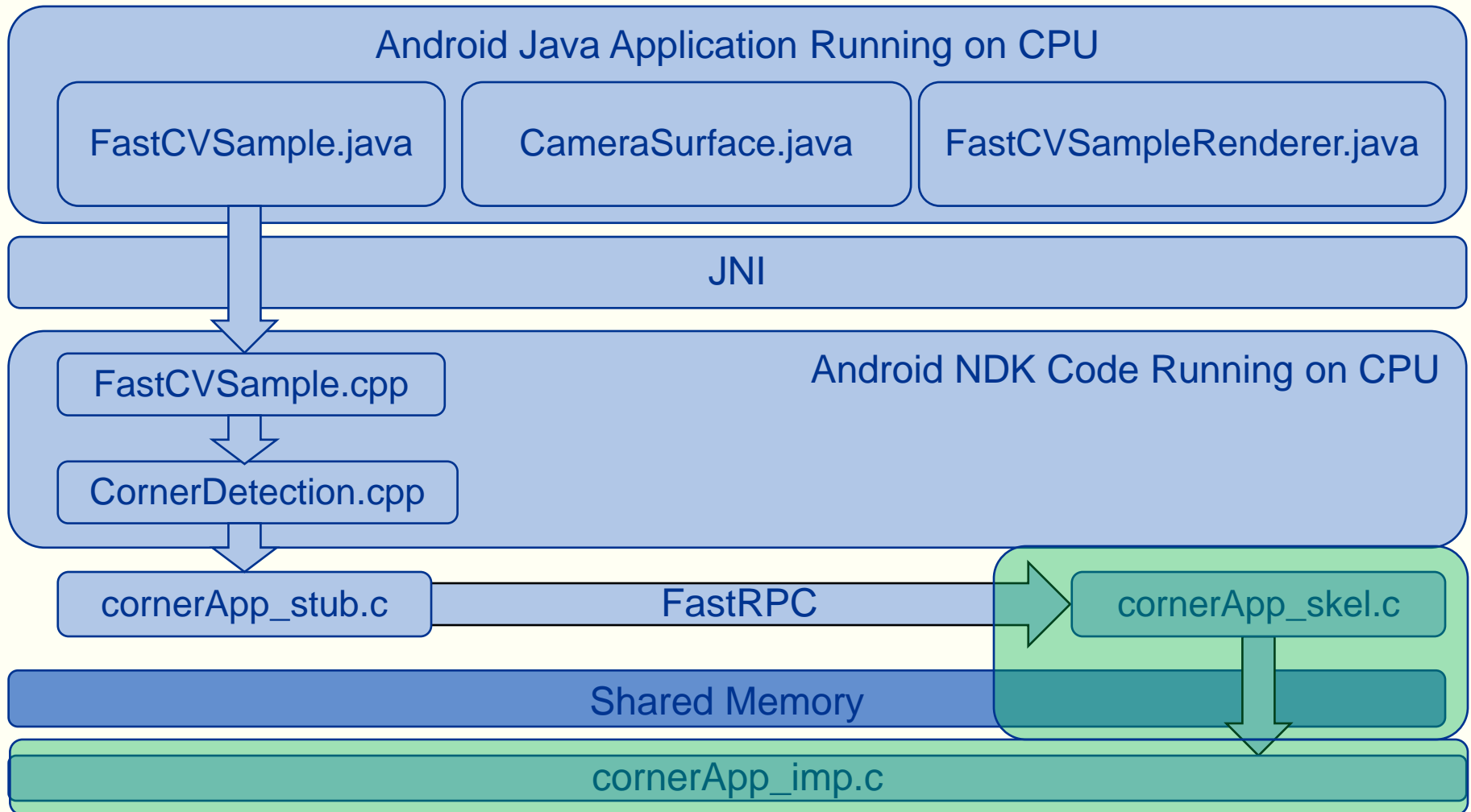
CameraRendererRGB565GL2.cpp

# Incorporating Your CV Primitive Into the Android Framework

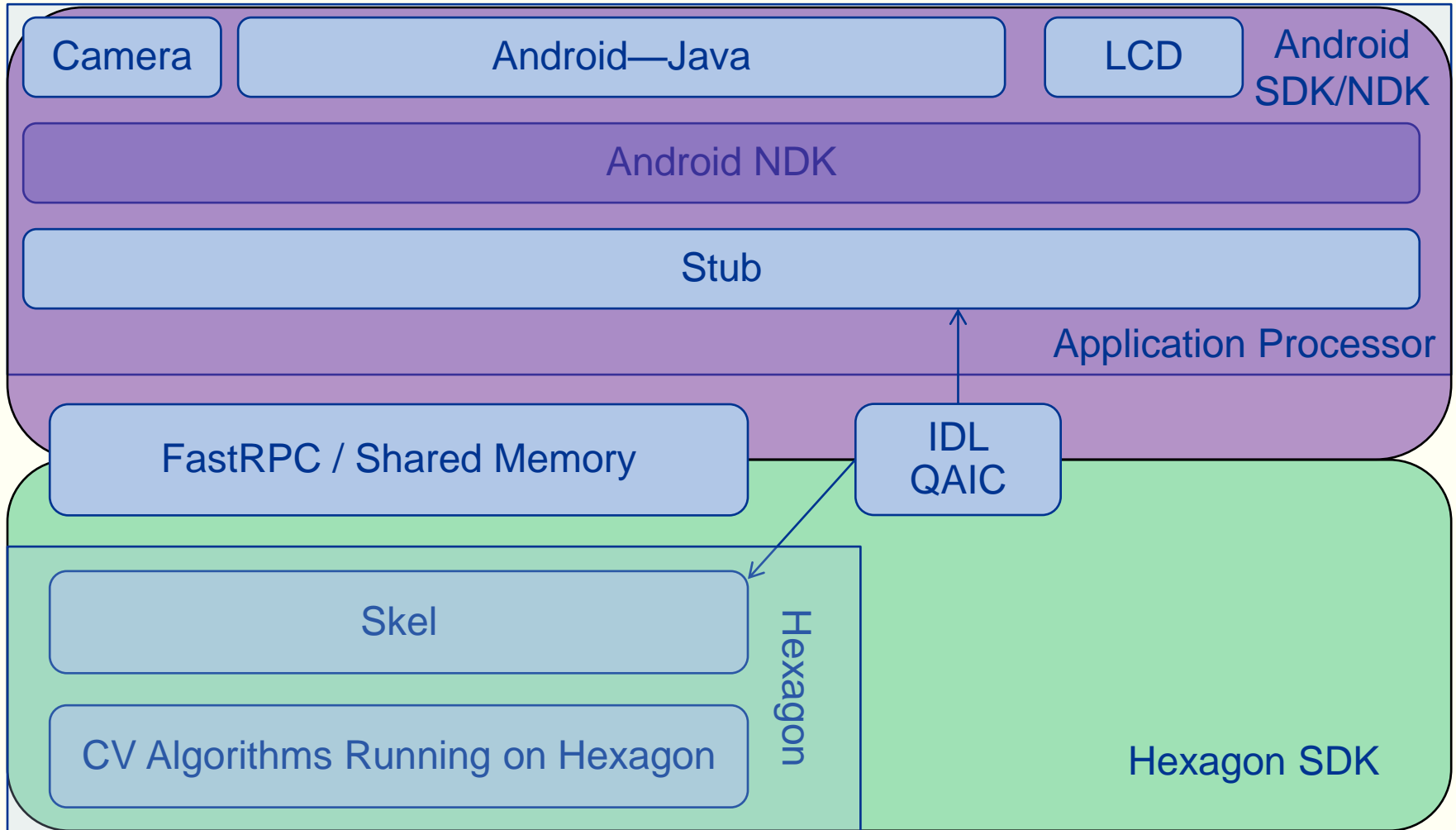




# Incorporating Your CV Primitive Into the Android Framework



# Incorporating Your CV Primitive Into the Android Framework



## Code Highlights

# Incorporating Your CV Primitive Into the Android Framework

```
/**
 * Actual callback function for camera frames. Does per frame
 * processing.
 *
 * @param data buffer for preview data, in YUV420sp format.
 * @param c Camera object.
 */
public void onPreviewFrame( byte[] data, Camera c )
{
    if( c != null )
    {
        // with buffer requires addbuffer each callback frame.
        c.addCallbackBuffer( mPreviewBuffer );
        c.setPreviewCallbackWithBuffer( this );
    }

    // Increment FPS counter for camera.
    cameraFrameTick();

    // Perform processing on the camera preview data.
    update( data, mDesiredWidth, mDesiredHeight, mCameraDisplayOrientation );

    // Simple IIR filter on time.
    mProcessTime = getFastCVProcessTime();
    Log.e( TAG, "FastCVProcessTime(): " + mProcessTime);

    // Mark dirty for render.
    requestRender();
}
```

## FastCVSample.java

The `onPreviewFrame()` method takes in a video frame from the camera in YUV420sp format (byte array data)

The JNI call 'update' performs the computer vision algorithm and copies the preview image to a render buffer.

A call to `requestRender()` displays the render buffer to the screen

# Incorporating Your CV Primitive Into the Android Framework

```
JNIEXPORT void
JNICALL Java_com_qualcomm_fastcorner_FastCVSample_update
(
    JNIEnv*     env,
    jobject     obj,
    jbyteArray  img,
    jint        w,
    jint        h,
    jint        rotation
)
{
    jbyte*      jimgData = NULL;
    jboolean    isCopy = 0;
    uint32_t*   curCornerPtr = 0;
    uint8_t*    renderBuffer;
    uint64_t    timeOverallStart;
    float       timeOverallMs;
    float       runningTime;

    // Get data from JNI
    jimgData = env->GetByteArrayElements( img, &isCopy );

    renderBuffer = getRenderBuffer( w, h );

    lockRenderBuffer();
}
```

## FastCVSample.cpp

The JNI call from Java to C++. This is the C++ side

getRenderBuffer() gets a blank buffer from the OpenGL pool

getRenderBuffer() is in FastCVSampleRenderer.cpp

# Incorporating Your CV Primitive Into the Android Framework

```

timeOverallStart = getTimeMicroSeconds();

// jimgData might not be 128 bit aligned.
// fcvColorYUV420toRGB888u8() and other fcv functionality inside
// Hexagon code require 128 bit memory aligned. In case of jimgData
// is not 128 bit aligned, it will allocate memory that is 128 bit
// aligned and copies jimgData to the aligned memory.

uint8_t* pJimgData = (uint8_t*)jimgData;

// Perform FastCV and Custom Algorithm on DSP
AlgorithmObject->RunAlgorithmOnDSP(pJimgData, renderBuffer);

timeOverallMs = ( getTimeMicroSeconds() - timeOverallStart ) / 1000.f;
totalTimeFilteredMs =
    ((totalTimeFilteredMs*(29.f/30.f)) + (float)(timeOverallMs/30.f));
unlockRenderBuffer();

// Let JNI know we don't need data anymore. this is important!
env->ReleaseByteArrayElements( img, jimgData, JNI_ABORT );

```

## FastCVSample.cpp

In JNI update

FastCV is used to convert the YUV420 image to RGB565 for display on the LCD

Note, 128 bit alignment required

updateCorners contains the computer vision algorithm

# Incorporating Your CV Primitive Into the Android Framework

```
void AlgorithmWrapper::RunAlgorithmOnDSP(uint8_t *YUVinput,
                                         uint8_t *RGBbuffer)
{
    memcpy(YUVimgBuf, YUVinput, YUVarrayNumElements);

    // Copy from camera buffer allocated on heap into ion allocated buffer.
    int retVal = algorithmApp_RunAlgorithmOnDSP
        (
            YUVimgBuf, YUVarrayNumElements, // input image
            imgWidth, // input image width
            imgHeight, // input image height
            RGB888Buffer, RGB888BufferNumElements,
            imgStride, // srcStride
            (uint32*)thresholdArray, thresholdArrayNumElements,
            (uint32*)RGBbuffer, RGBarrayNumElements);

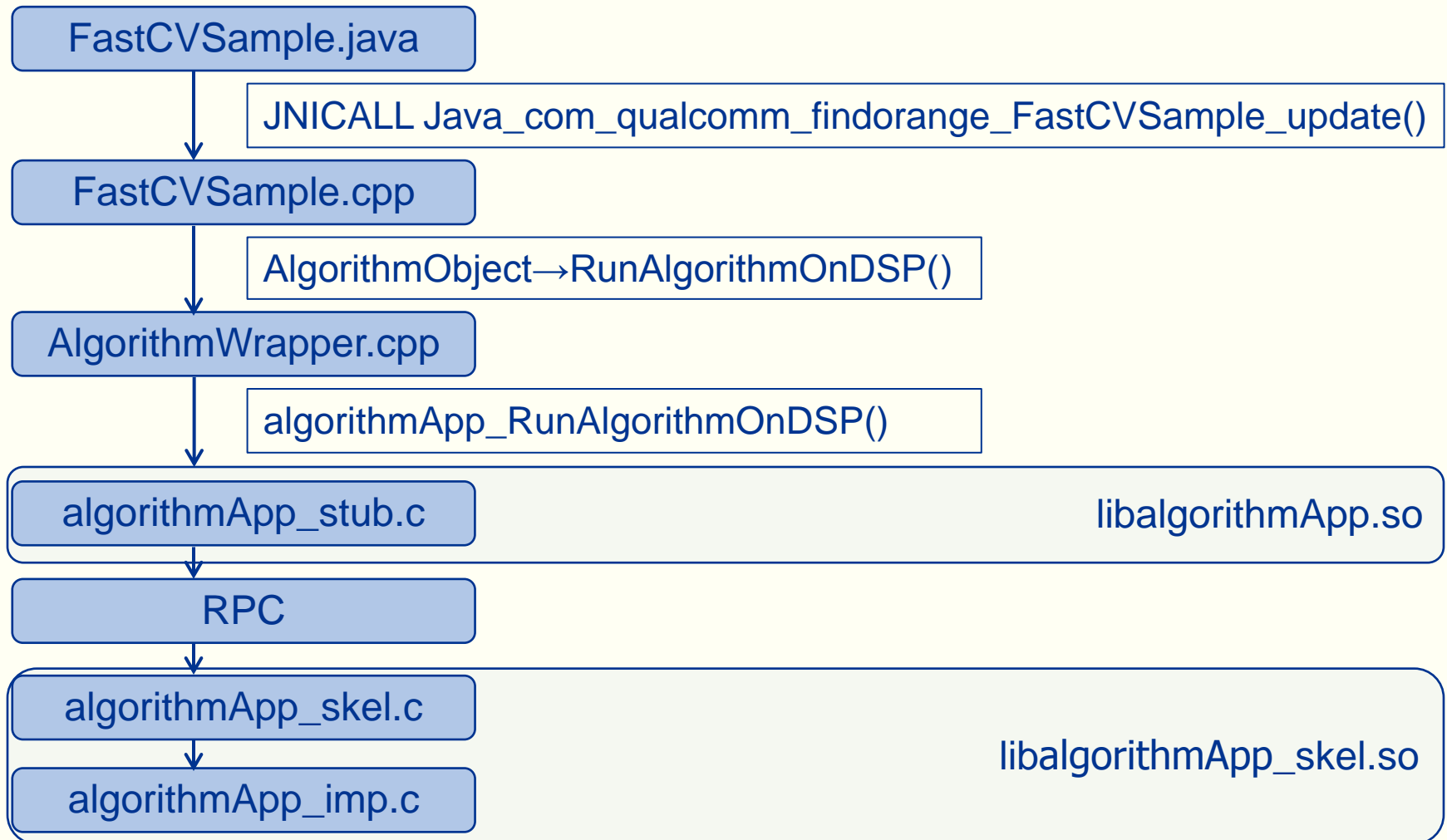
    DPRINTF("retVal from Algorithm: %d", retVal);
    if (retVal!=0)
        EPRINTF("fastRPC call failed; return value: %d", retVal);
}
```

## Actual call to DSP

The YUVimgBuf, RGB888Buffer, thresholdArray, and RGBbuffer have all been allocated using the fastRPC library

These are ION buffers

# Incorporating Your CV Primitive Into the Android Framework





# Incorporating Your CV Primitive Into the Android Framework

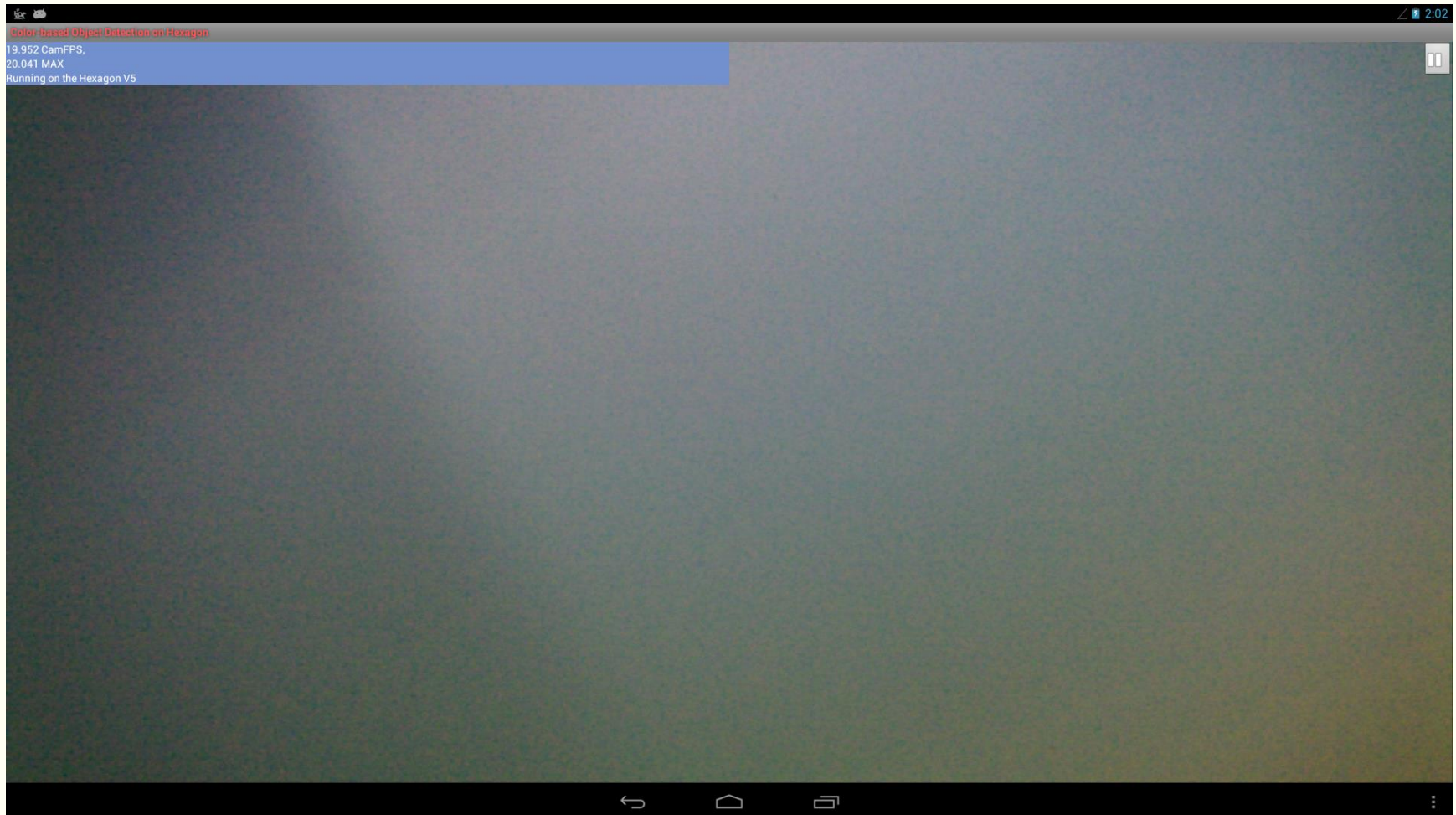
```
/*=====
  REMOTED FUNCTION - This code is built using the Hexagon SDK
                    and executes on The Hexagon DSP.
  =====*/
int algorithmApp_RunAlgorithmOnDSP( const uint8* imgSrcY,
                                   int srcYLen,
                                   uint32 srcWidth,
                                   uint32 srcHeight,
                                   uint8* imgDstRGB888,
                                   int dstRGB888Len,
                                   uint32 srcYStride,
                                   uint32* Threshold_ptr,
                                   int thresholdCLen,
                                   uint32* imgDstRGB565,
                                   int dstRGB565Len)
{
    MultiTask( imgSrcY,
               srcYLen,
               srcWidth,
               srcHeight,
               imgDstRGB888,
               dstRGB888Len,
               srcYStride,
               Threshold_ptr,
               thresholdCLen,
               imgDstRGB565,
               dstRGB565Len);

    return 0;
}
```

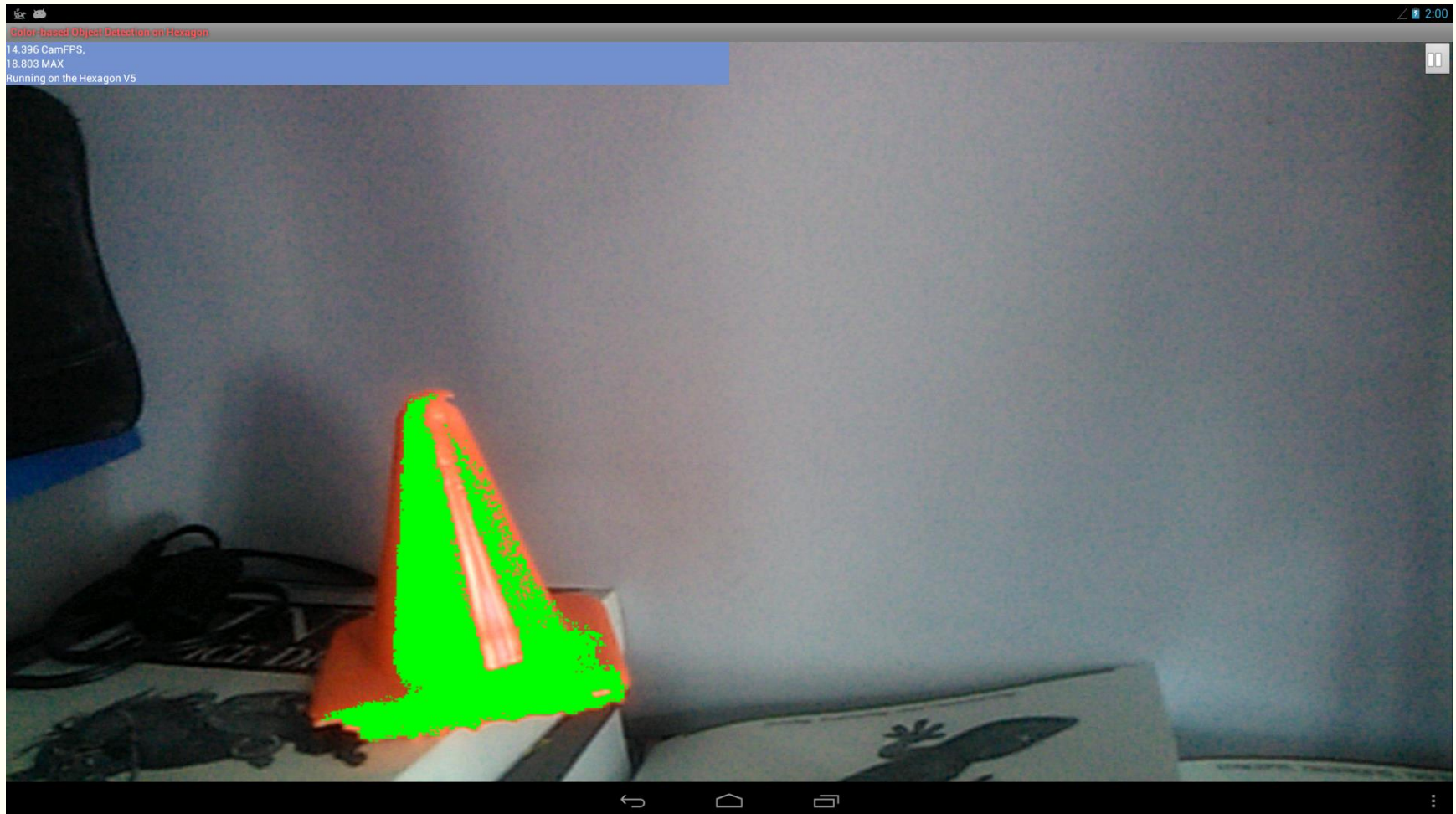
## Interface function on Hexagon DSP side

This code is built using the Hexagon SDK, compiled into a shared object, and executed on the Hexagon DSP

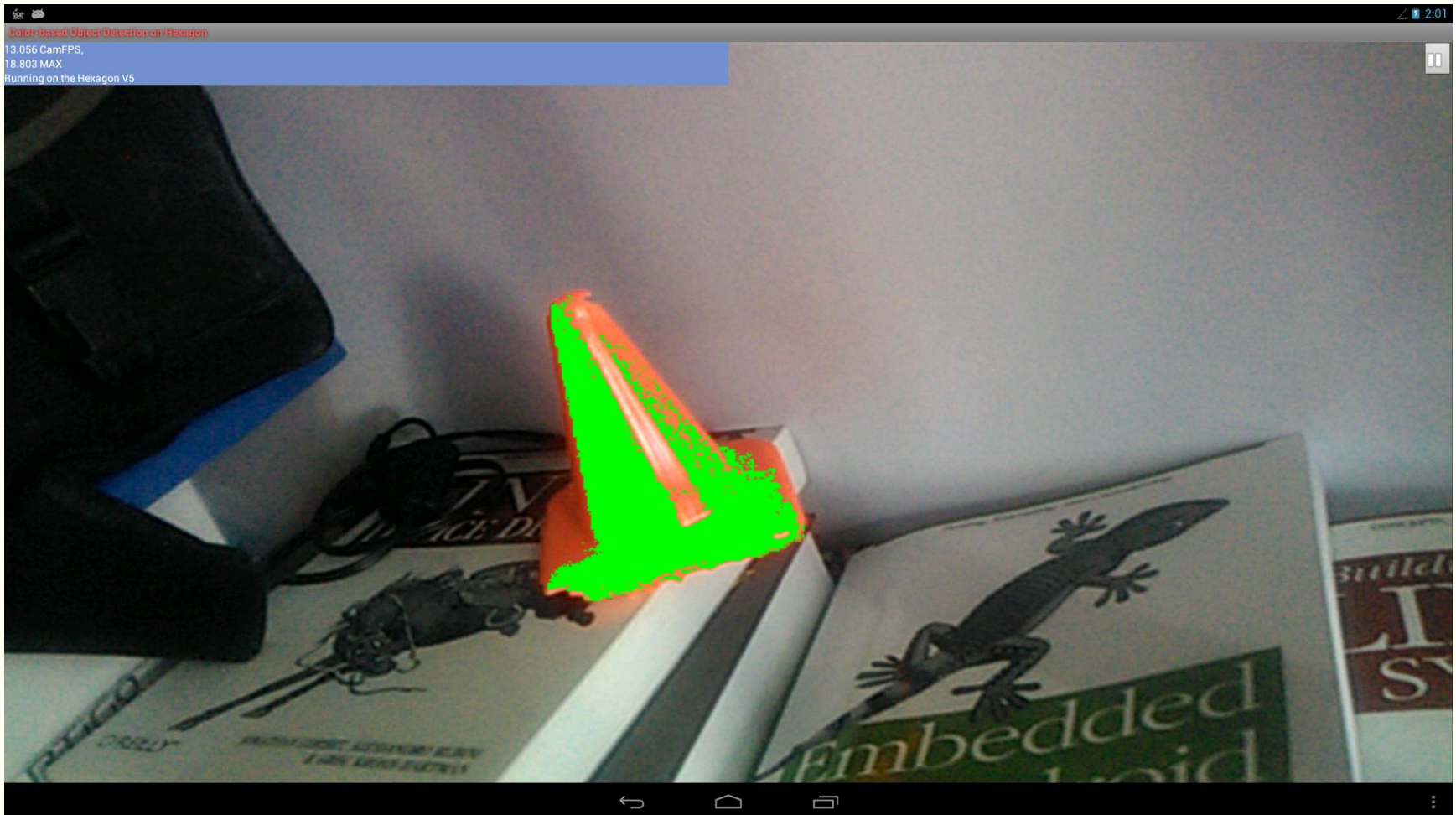
# Demo Performance While Running on Hexagon



## Demo Performance While Running on Hexagon



## Demo Performance While Running on Hexagon



## Demo Performance While Running on Hexagon



## Performance Assessment

	Relative Performance
Krait CPU (un-optimized plain float C) (single-core)	1X
Hexagon DSP (un-optimized plain float C) (single-threaded)	1.12X
Hexagon DSP (un-optimized plain float C) (three threads)	2.8X
Hexagon DSP (optimized plain fixed C) (three threads)	11.2X

No intrinsic or vectorization were used for optimization

## Summary

- FastCV is a Qualcomm computer vision library with functions ported to both ARM and the Hexagon DSP
- The FastCV SDK provides an android example that passes video frames from the camera to C/C++ code via JNI
- FastRPC provides a mechanism for calling functions on the DSP and passing data to the DSP from C/C++ code
- Hexagon is a powerful multi-threaded DSP that can be used to offload compute intensive functions from the CPU
- The Hexagon SDK and FastRPC library make it easy to move functionality from the CPU to the DSP
- The Hexagon SDK includes multiple, well documented examples that provide solid foundations for moving computer vision algorithms from the CPU to the DSP

## To Explore Further—Additional Resources From Qualcomm

### Qualcomm FastCV Website

<https://developer.qualcomm.com/mobile-development/mobile-technologies/computer-vision-fastcv>

### Qualcomm Hexagon Website

<https://developer.qualcomm.com/hexagon-processor>



## BDTI Expertise Will Speed Your Product to Market

BDTI provides expert optimized software development for Qualcomm Snapdragon processors:

- Hexagon DSP
- Krait CPU
- Adreno GPU

Use BDTI's software engineering services to accelerate time-to-market, reduce risk, and improve software performance

For a confidential discussion of your requirements, contact:

Jeremy Giddings  
Director of Business Development  
+1 (925) 954-1411  
giddings@BDTI.com

# embedded VISION ALLIANCE

Inspiring and empowering engineers  
to design systems that see and understand

[www.Embedded-Vision.com](http://www.Embedded-Vision.com)



## The Embedded Vision Alliance

- The Embedded Vision Alliance is helping transform embedded vision's potential into reality by enabling and empowering design engineers to incorporate vision capabilities into products
- The Alliance offers a growing array of free resources for design engineers, including online seminars, technical articles, downloadable demos, and discussion forums
- Visit [www.Embedded-Vision.com](http://www.Embedded-Vision.com):
  - Become a registered user and sign up for the free newsletter and new-content notifications (via RSS, LinkedIn, Twitter)
  - Access technical articles, as well as video tutorials, interviews, demos, and news
  - Participate in discussion forums and comment on articles
  - Use the Embedded Vision Academy



## Embedded Vision Summit East 2013

### October 2, 2013

- A full day of “how-to” presentations and demos of technology in one of the hottest area of technology today—embedded vision
- Talks by industry luminaries
  - Keynote by Marion Munich, VP of iRobot
  - Special presentation by Mike Geertsen, Program Manager for DARPA’s Visual Media Reasoning program
- Presentations and demos from embedded technology leaders
  - Analog Devices • Apical • Avnet • BDTI • Cadence • CEVA • CogniVue • Geo Semiconductor • MathWorks • National Instruments • PathPartner • Qualcomm • Synopsys • Texas Instruments • videantis • Xilinx • and others
- Register online today at [www.Embedded-Vision.com](http://www.Embedded-Vision.com)

