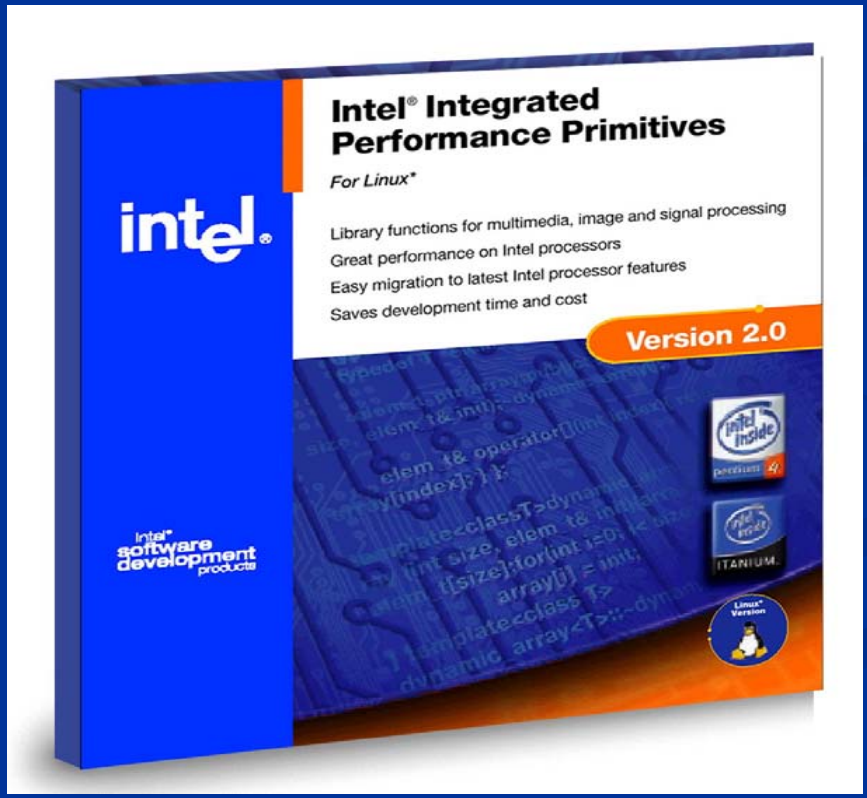
武汉大学《多核架构及编程技术》教学课程之

第 七 章

IPP程序设计

What is IPP?

Integrated Performance Primitives
集成性能基元



What Is A Primitive?

- A low level building block that
  - Abstracts low-level implementation details
  - Performs a single operation
  - Is a component piece of a larger solution

What Are IPP ?

- Huge collection of useful multimedia functions
  - Mathematics, signal, speech, audio, video, image, graphics
- Low-level and simplex
- Highly-optimized, processor-specific code
- Common across multiple platforms
  - desktop, and server processors
  - Windows*, Linux operation systems
- Widely used, well-received product

主要内容

- IPP简介
- 通过 IPP 获得更高的性能
- 编程基础
- 编程示例

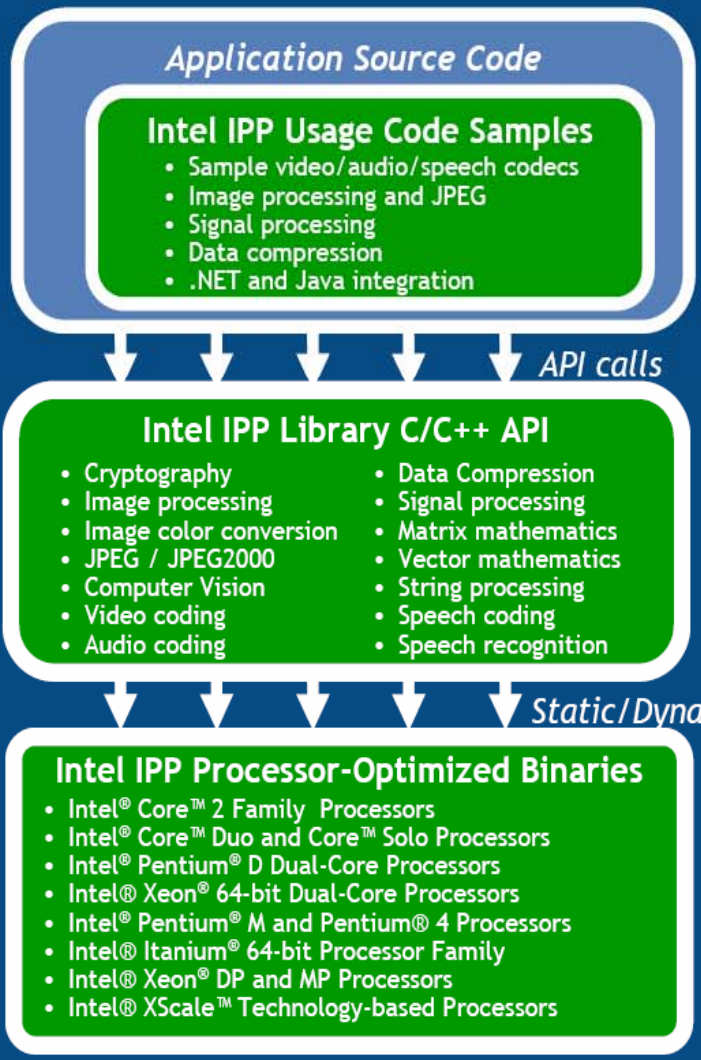
IPP简介

- 面向Intel处理器和芯片的函数库

  信号处理，图像处理，多媒体，向量处理等
- 具有跨平台和操作系统的通用 API
- 提供高性能的代码


应用
数字媒体
Web/企业数据
嵌入式
通讯
科技
可实现代码
复用的跨平台
C/C++ API
英特尔® 集成性能基元
免费代码范例
15 个函数域
图像和视频
✓ 图像处理
✓ 颜色转换
✓ JPEG / JPEG2000
✓ 视频编码
✓ 计算机视觉
✓ 射线跟踪/渲染
通讯和信号处理
✓ 信号处理
✓ 音频编码
✓ 语音编码
✓ 语音识别
✓ 矢量运算
数据处理
✓ 数据压缩
✓ 加密技术
✓ 字符串处理
✓ 矩阵/矢量代数
优化的 32 位和 64 位多核性能

Intel® Integrated Performance Primitives (Intel® IPP) — Overview and Benefits



Free Code Samples

Cross-platform Application Programming Interface (API)

Processor-Optimized Implementation

Benefits

Rapid Application Development

Cross-platform Compatibility and Code Re-Use

Outstanding Performance

The ~~14~~ 15 Domains in IPP

1. **Static Pictures**
   - Image Processing
   - Computer Vision
   - JPEG (Image coding)
   - Ray-Tracing/Rendering*
2. **Motion Pictures**
   - Video Coding
   - Color Conversion
3. **Sound and Speech**
   - Audio Coding
   - Speech Coding
   - Speech Recognition
4. **Computer Data**
   - Data Compression
   - Cryptography
   - Text string processing
5. **Mathematics**
   - Digital Signal Processing
   - Small matrices
   - Vectors

* new in Intel IPP 5.2 (Q2-2007)

IPP 5.x Domains and Functions

	Domain	IA	IXP
Image Processing	ippi	2570	1574
Signal Processing	ipps	1865	783
Speech Recognition	ippsr	618	618
Computer Vision	ippcv	417	417
Color Conversion	ippcc	410	410
Cryptography	ippcp	307	307
Speech Coding	ippsc	297	247
Video Coding	ippvc	219	219
Image Coding	ippj	201	201
Audio Coding	ippac	152	152
Data Compression	ippdc	73	73
Char Processing	ippch	72	72
Small Matrix	ippm	669	0
Vector Math	ippvm	136	0
	Totals	8006	5082

性能优化的函数

函数领域列表		
视频编码	JPEG 编码	音频编码
图像处理	语音编码	语音识别
计算机视觉	信号处理	矢量/矩阵运算
颜色转换	数据压缩	加密
字符串处理	射线跟踪/渲染 （新！）	

Intel IPP:
Codec and Data Processing Standards Support

- Video Codecs
  - H.264
  - H.263
  - H.261
  - MPEG-4
  - MPEG-2
  - Motion JPEG
  - DV
- Audio Codecs
  - MP3
  - AAC
  - AC3
- Image Codecs
  - JPEG
  - JPEG2000

- Speech Codecs
  - AMR-WB
  - G.711 / I / II
  - G.722.1 G.722.2 (GSM-AMR)
  - G.723.1 / A
  - G.726 / A
  - G.728 G/I/H
  - G.729, G.729A, G.729D, G.729E, G.729I
  - GSM-FIR
  - GSM 06.90-06.94
  - GSM 06.10-06.12
  - GSM 06.31-06.32
- Echo Cancellation
  - G.168-2000
  - G.167
- Speech Recognition
  - Aurora
  - Advanced Aurora
  - Gaussian Mixture

- Data Compression
  - Huffman encoding/decoding
  - RLE encoding/decoding
  - MoveToFront (MTF)
  - Burrows-Wheeler Transformations (BWT)
  - General Interval Transform (GIT)
  - Lempel-Ziv-Storer-Szymanski (LZSS) functions
- Cryptography
  - Rijndael, DAARijndael
  - DES, DAA-DES
  - Triple DES, DAA-TDES
  - Twofish, DAA-Twofish
  - Blowfish, DAABlowfish
  - SHA1, SHA256/384/512, HMAC-SHA1
  - MD5, HMAC-MD5
  - Digital Signature Algorithm (DSA)

Higher-Level APIs

Examples in the Market Today:

- Data Compression:
  - zlib, libbzip2
- Medical/Document Imaging:
  - Pegasus Imaging
  - LeadTools
  - Accusoft
  - Snowbound Software
  - libPNG
- Encrypted communications:
  - OpenSSL
- Computer Vision
  - OpenCV
- Signal Processing
  - VSIPL++

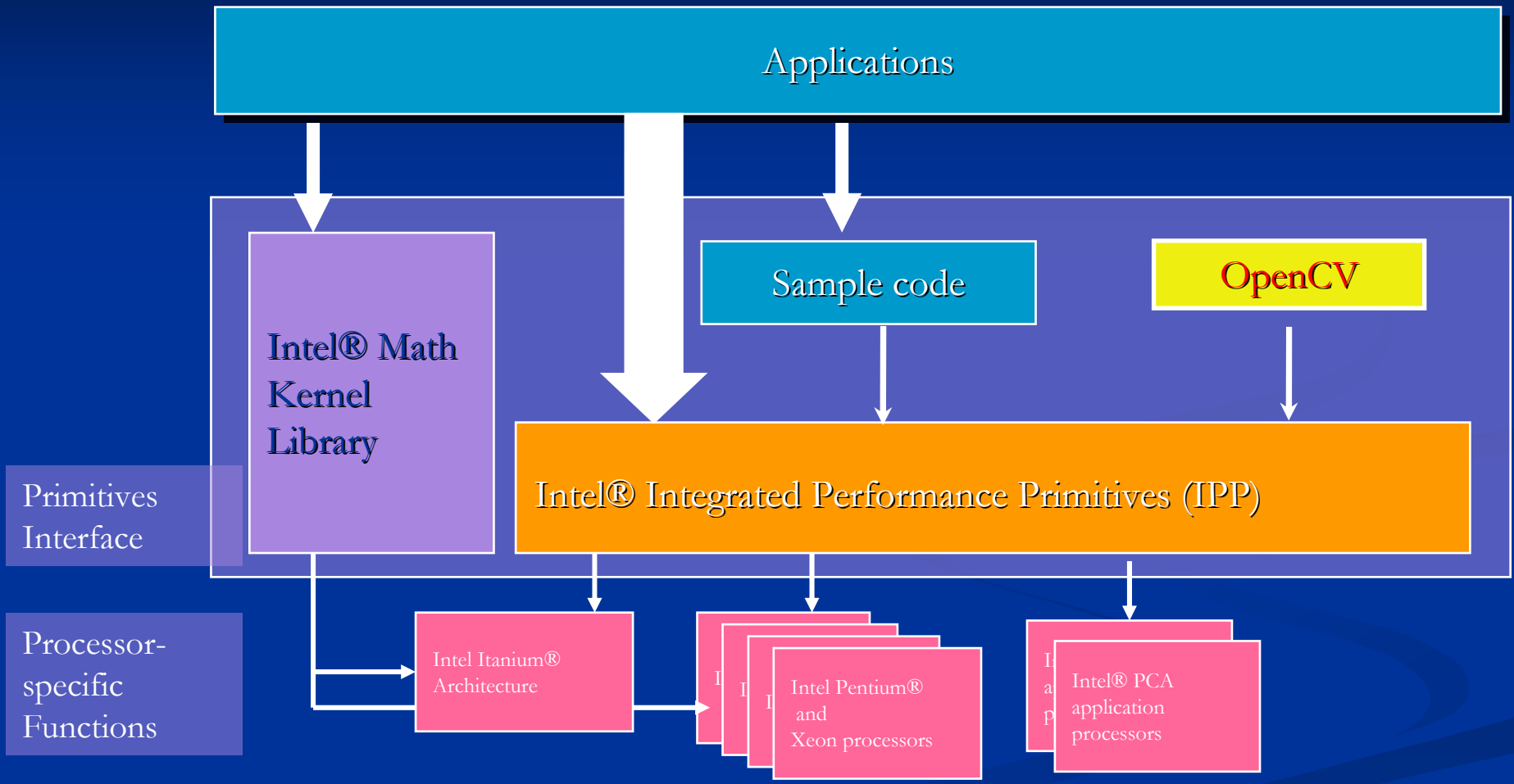
Proposed Strategy:

- Use Code Samples to enter higher-level API market
- "Promote" code samples into binary-only, new libraries
  - higher performance than code samples

Question:

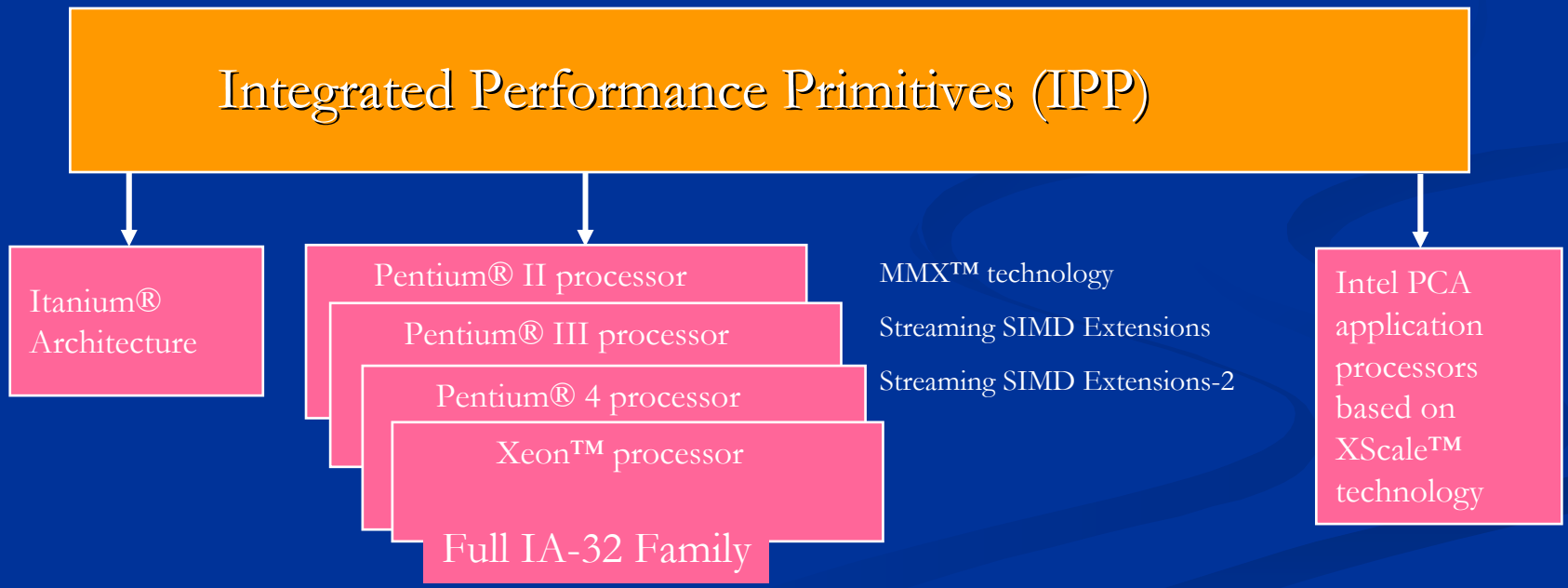
- Will ISVs object to not having source code?

IPP与Intel其它组件的关系



易于在多种平台上开发

- 面向处理器和芯片组
  - 自动选择处理器相关的 DLL
  - 和体系结构相关的指令集



跨平台和操作系统

- 支持多种平台
  - MMX™, Streaming SIMD Extensions (SSE) and Streaming SIMD Extensions 2 (SSE-2) Technologies
  - IA-32 (including Intel® Xeon™ processor)
  - Itanium® architecture
  - Intel® XScale™ micro-architecture
- 支持多种操作系统
  - Windows NT* 4.0 / Windows 2000* /Windows XP*
  - Windows XP* 64-bit
  - Linux* & Linux-64
  - Windows CE*, Linux in embedded device

不需要写底层（汇编）代码，获得优化的应用程序

更多的特性

- 支持:
  - SPL (Signal Processing Library)
  - IPL (Image Processing Library)
  - IJL (Intel® JPEG Library)
  - RPL (Recognition Primitives Library)

…supporting migration to new platforms

更多的特性

- 编解码原语:
  - MPEG-1, MPEG-2, MPEG-4, JPEG2000
  - G.729, G.723, GSM AMR
- 编解码采样:
  - MPEG-1, MPEG-2, MPEG-4
  - G.729, G.723, GSM AMR

…supporting video, image and speech encoding/decoding

更多的特性

- 支持更多的颜色模式和转换
- …and much, much more!

…tell us what you want!

通过 Intel® IPP 获得更高的性能

对IPP性能的评论

Leo Volfson, President and Chief Technology Officer, Inetcam, Inc.

"The Intel ® Integrated Performance Primitives (IPP) has enhanced the iVISTA* application to be more in line with customers' expectations. For example, it enables us to dynamically rescale, in real-time, a video stream without loss of performance. This capability would not be possible without IPP."

"The Intel IPP provided a 300% improvement in the number of users who can simultaneously participate in a webcast. In addition, the migration from the Intel Pentium III to the Intel Pentium 4 took only a day."

对IPP性能的评论

Bryan Cook, Software Architect, AuSIM Inc, Los Altos, California October 2001

"AuSIM Inc. delivers the most advanced audio simulation technology for mission-critical aural displays and simulations. With Intel's Integrated Performance Primitives (IPP), AuSIM has leveraged 4X performance gains within its AuSIM3D* audio simulation technology. …directly enhances AuSIM's ability to provide the ultimate audio solutions for simulations, team communications, audio production, tele-conferences, and aural information displays."

Demo

Minor effort, major gains

通过Intel® IPP，很小的代码改变可以获得极大的性能

通过将运行时的函数调用替换为IPP，应用程序模块的性能可以得到极大的改进

1x

```
pFlTmp=flArgSin[thrd].algPtr;
for(t=0; t<4*iWvMeshSize; t++)
{
pFlTmp[t]=(float)sin(pFlTmp[t]);
}
```

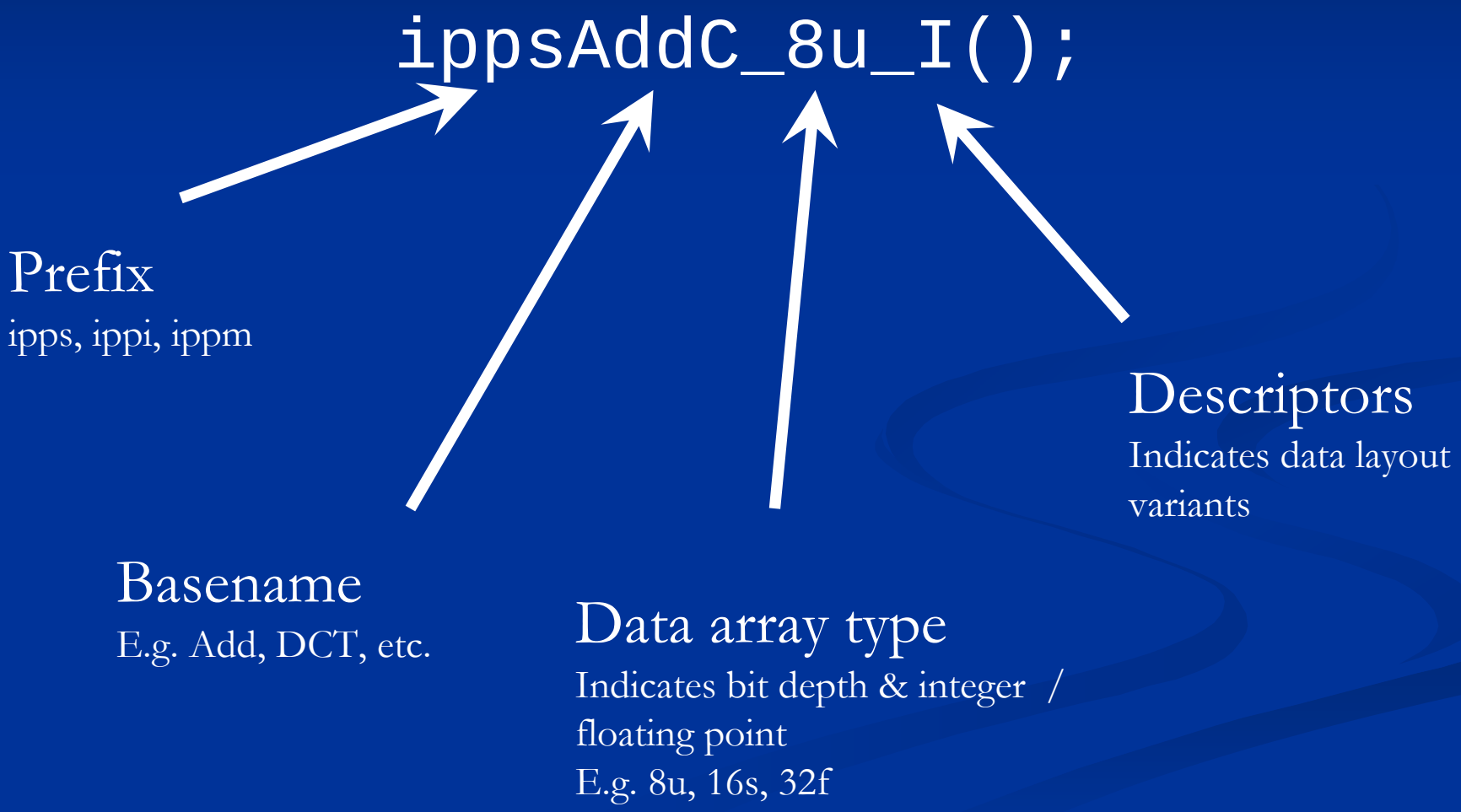
4x

```
ippsSin_32f_A11(flArgSin[thrd].
algPtr,flArgSin[thrd].algPtr,
4*iWvMeshSize);
```

编程基础

- 编程环境设置与约定
- 编程示例

IPP 函数命名


ippsAddC_8u_I();
Prefix
ipps, ippi, ippm
Basename
E.g. Add, DCT, etc.
Data array type
Indicates bit depth & integer / floating point
E.g. 8u, 16s, 32f
Descriptors
Indicates data layout variants

数据类型和布局

- 对数据类型的特殊优化 (8u, 16s, 32f)
- 对数据布局的特殊优化 (pixel, planer…)
- 对数据类型和布局的转换函数的优化

```
ippsConvert_8u_32f()
ippsInterleave_16s()
ippiRGBToYUV_8u_C3P3R()
```

Interleaving:

- L/R/L/R/L/R… stereo
- RGBRGBRGB… color image
- RGBARGBA… color +alpha

Color Models:

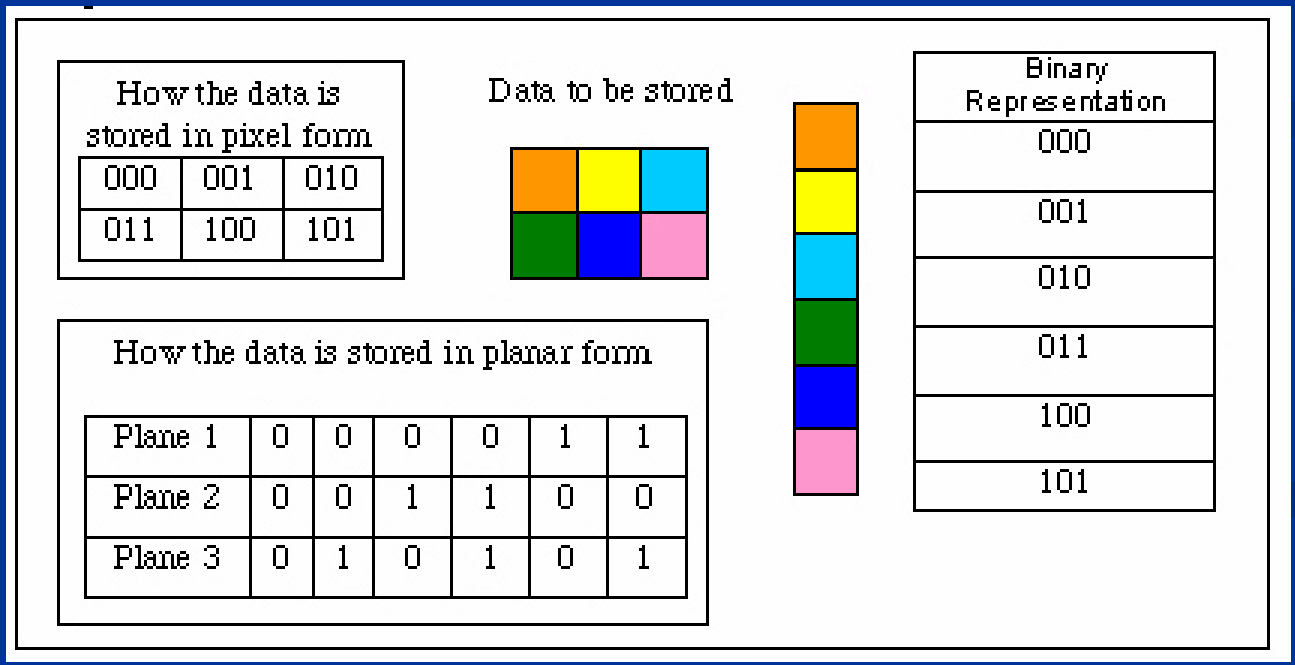
- YCbCr (4:4:4, 4:2:2)
- YUV (4:2:2, 4:2:0)
- YCC, HLS, RGB, RGBA …

IPP基础类型

代码	Intel IPP类型	C语言中的定义
8u	Ipp8u	unsigned char
8s	Ipp8s	char
16u	Ipp16u	unsigned short
16s	Ipp16s	short
32u	Ipp32u	unsigned int
32s	Ipp32s	int
32f	Ipp32f	float
64f	Ipp64f	double
1u	Ipp8u	unsigned char* bitstream, int offset
16sc	Ipp16sc	struct { Ipp16s re; Ipp16s im;} Ipp16sc
32sc	Ipp32sc	struct { Ipp32s re; Ipp32s im;} Ipp32sc
32fc	Ipp32fc	struct { Ipp re; Ipp im;} Ipp32fc
64fc	Ipp64fc	struct { Ipp re; Ipp im;} Ipp64fc

数据布局

- pixel格式中，每个像素所有位都被顺序存储
- 在planer格式中，每个像素的第一位被存储，接着每个像素的第二位被存储，等等



ipps函数描述符

代码	描述	例子
Axx	用于高级算法运算，详述精确的结果位数	ippsSqrt__A11 ippsSqrt_32fA24 ippsCos__A50 ippsCos__A53
I	运算是一体的。运算结果被写回源变量，自变量既是源变量又是目标变量	ippsFlip_16u（三个自变量src，dst，len） ippsFlip_16u_I（两个自变量，srcdst，len）
Sfs	此函数缩放了运算的结果，通过变量ScaleFactor转换。这通常在运算结果太大时，用于保持结果精度。	ippsAddC_16s_Sfs（src，val，dst，len，scaleFactor）用src和val相加得到的结果处以scaleFactor，结果存入dst

ippi函数的描述符编码

编码	描述	例子
AC4	图像具有4个通道，第四个通道是α通道。当用这种方法标识时，α通道将被排除在运算外。	ippiAddC_8u_AC4R ippiMean_8u_AC4R
C1,C3,C4	图像在内存中具有1，3或4个通道的交叉数据。大多数情况下用来表示一个通道的C1也可以用于多平面图像。	ippiDCT8x8Fwd_16s_C1T ippiRGBToYUV_8u_C3R
C2	图像在内存中具有交叉的通道。这是种特殊情况，在这种情况下第二个“通道”本身经常是两个交叉子通道组成的。它的大小和变量都是基于双通道图像的。	ippiYCbCr422ToRGB_8u_C3R ippiJoin422_9u_P2R
I	运算的结果被写回源程序，因此变量既是源代码又是目标代码。	ippiDCT8x8Fwd_16s_C1（用三个图像作为变量src1，src2和dst） ippiDCT8x8Fwd_16s_C1T（用两个图像作为变量src和srcdst）
M	运算使用模板来判断该对哪个像素进行运算。	ippiSet_8u_C1MR(val, dst, dstStep, size, mask, maskStep)
P3,P4	图像具有3个或4个通道的数据分布在独立的平面；运算采用指针序列对图像平面进行运算。	ippiRGBToYCbCr420_8u_C3P3R（用交叉的RGB图像作为输入并把结果写入三个分立的输出序列中）
R	函数在输入图像中定义的兴趣区域运算。大多数图像处理函数有此描述符并使用ROI。	
Sfs	此函数缩放运算的结果，通过变量ScaleFactor转换。这通常在运算结果太大时，用于保持结果精度。	ippiAddC_8u_AC4Sfs(src, values, dst, size, scalefactor) 用src和value相加得到的结果处以scaleFactor，结果存入dst

Selecting Between Linking Models

- dynamically using the run-time dynamic link libraries (DLLs)
- statically using e-merged and merged static libraries
- statically without automatic dispatching using merged static libraries
- dynamically building your own, custom, DLL.

Summary of Intel IPP Linkage Model Comparison

Feature	Dynamic Linkage	Static Linkage with Dispatching	Static Linkage without Dispatching	Custom Dynamic Linkage
Processor Updates	Automatic	Recompile & redistribute	Release new processor-specific application	Recompile & redistribute
Optimization	All processors	All processors	One processor	All processors
Build	Link to stub static libraries	Link to static libraries and static dispatchers	Link to merged libraries	Build separate DLL
Calling	Regular names	Regular names	Processor-specific names	Regular names
Total Binary Size	Large	Small	Smallest	Small
Executable Size	Smallest	Small	Small	Smallest
Kernel Mode	No	Yes	Yes	No

To dynamically link with Intel IPP

1. Include ipp.h which will include the include files of all IPP domains,

2. Use the normal IPP function names when calling IPP functions,

3. Link corresponding domain import libraries. For example, if you use the ippsCopy_8u function, link against ipps.lib,

4. Make sure that run-time libraries, for example ipps.dll are on the executable search path at run time. Run the ippenv.bat from directory \tools\env to ensure this application built with Intel IPP dynamic libraries will load the appropriate processor-specific DLL.

To dynamically link with Intel IPP

To use the dynamic linking libraries, you need to link to ipp*.lib files in the \stublib directory. Intel IPP domain-specific functions are included in this directory. For example, ippj.lib is the JPEG library and ippi.lib is the image processing library (see Building an Application for details). You need to link to all corresponding domain libraries used in your applications.

For example, your application uses three Intel IPP functions ippicopy_8uC1R, ippiCanny_16s8u_C1R and ippmMul_mc_32f. These three functions belong to the image processing domain, computer vision domain and matrix domain respectively. In order to include these functions into your application, you need to link to the following three Intel IPP libraries: ippi.lib、ippcv.lib、ippm.lib

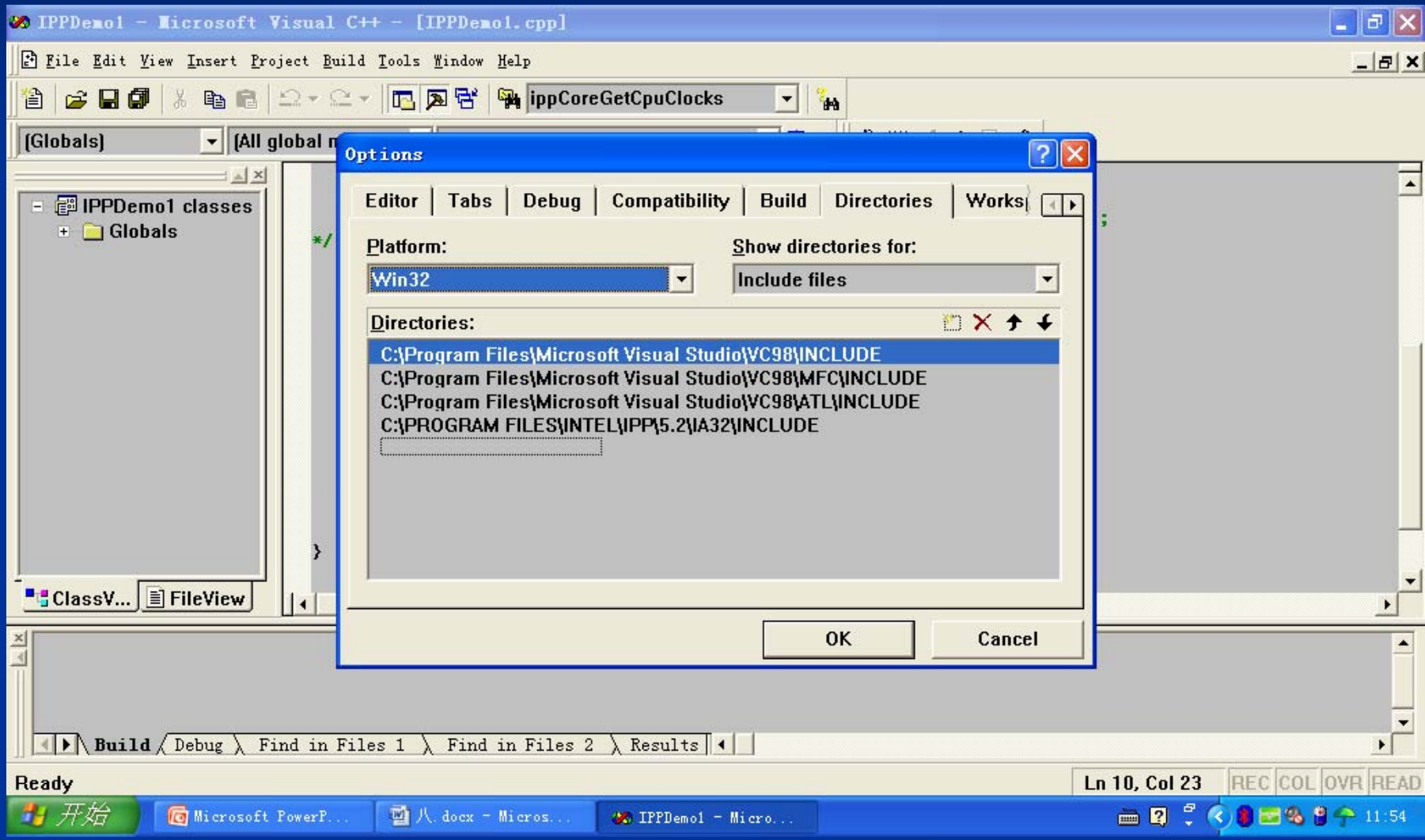
Libraries used for each linking model - 1

Domain Description	Header Files	Dynamic Linking	Static Linking with Dispatching & Custom Dynamic Linking	Static Linking without Dispatching
Audio Coding	ippac.h	ippac.lib	ippacemerged.lib	ippacmerged.lib
Color Conversion	ippcc.h	ippcc.lib	ippccemerged.lib	ippccmerged.lib
String Processing	ippch.h	ippch.lib	ippchemerged.lib	ippchmerged.lib
Common Functions	ippcore.h	ippcore.lib	ippcorel.lib	ippcorel.lib
Cryptography	ippcp.h	ippcp.lib	ippcpemerged.lib	ippcpmerged.lib
Computer Vision	ippcv.h	ippcv.lib	ippcvemerged.lib	ippcvmerged.lib
Data Compression	ippdc.h	ippdc.lib	ippdcemerged.lib	ippdcmerged.lib
Image Processing	ippi.h	ippi.lib	ippiemerged.lib	ippimerged.lib

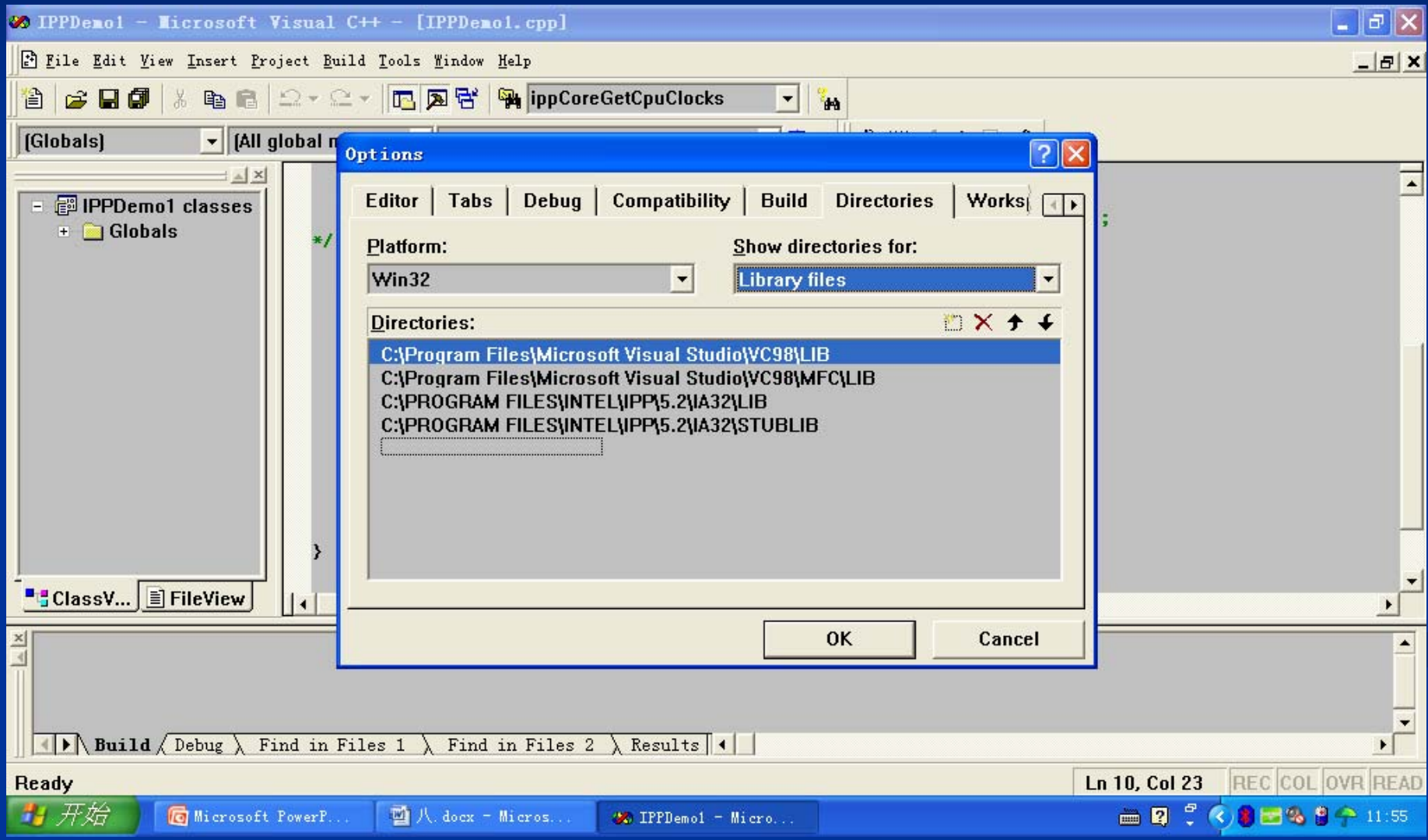
Libraries used for each linking model - 2

JPEG Primitives	ippj.h	ippj.lib	ippjemerged.lib	ippjmerged.lib
Realistic Rendering	ippr.h	ippr.lib	ippremerged.lib	ipprmerged.lib
Small Matrix	ippm.h	ippm.lib	ippmemerged.lib	ippmmerged.lib
Signal Processing	ipps.h	ipps.lib	ippsemerged.lib	ippsmerged.lib
Speech Coding	ippsc.h	ippsc.lib	ippscemerged.lib	ippscmerged.lib
Speech Recognition	ippsr.h	ippsr.lib	ippsremerged.lib	ippsrmerged.lib
Video Coding	ippvc.h	ippvc.lib	ippvcemerged.lib	ippvcmerged.lib
Vector Math	ippvm.h	ippvm.lib	ippvmemerged.lib	ippvmmerged.lib

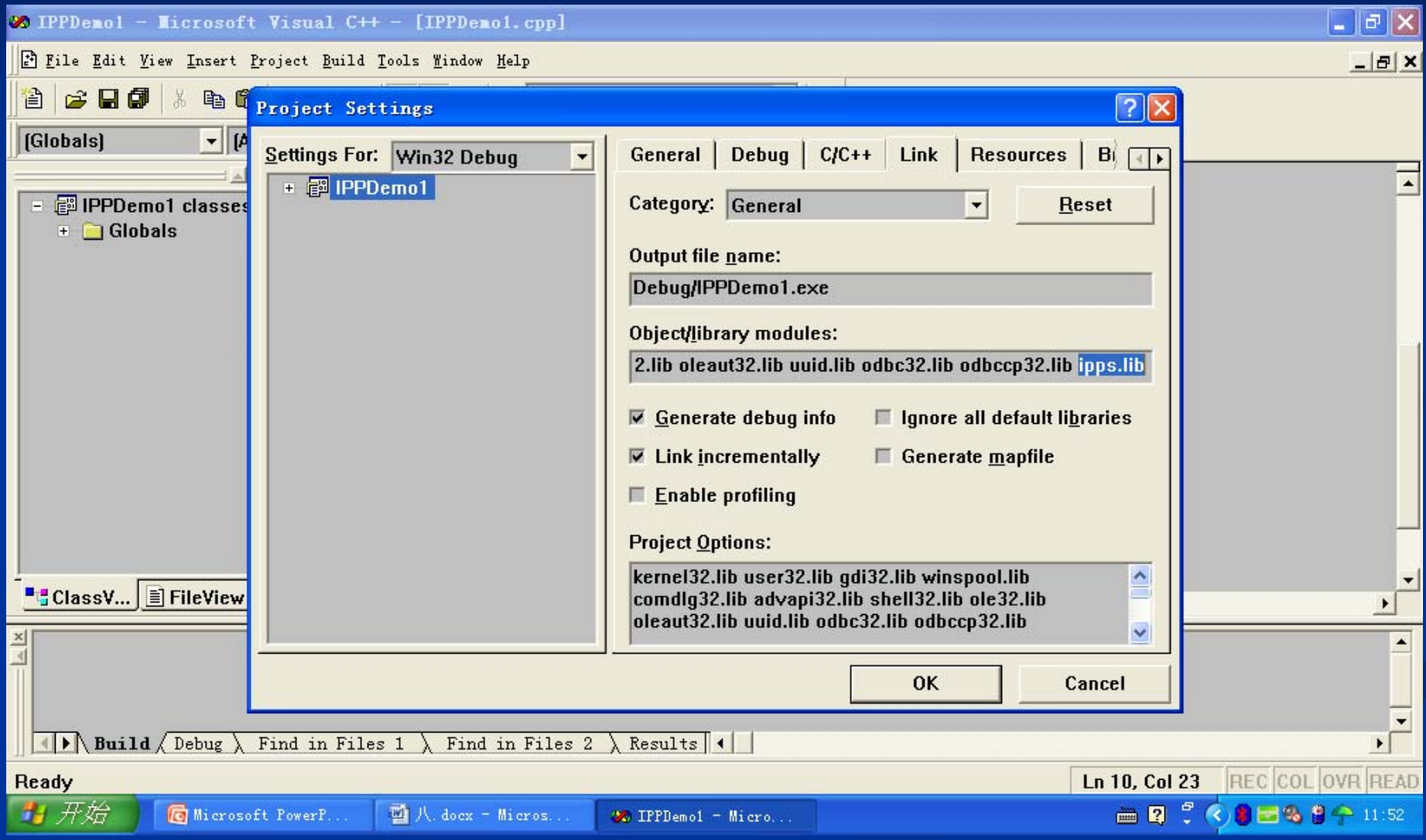
VC6.0编程头文件与库文件路径设置



VC6.0编程头文件与库文件路径设置

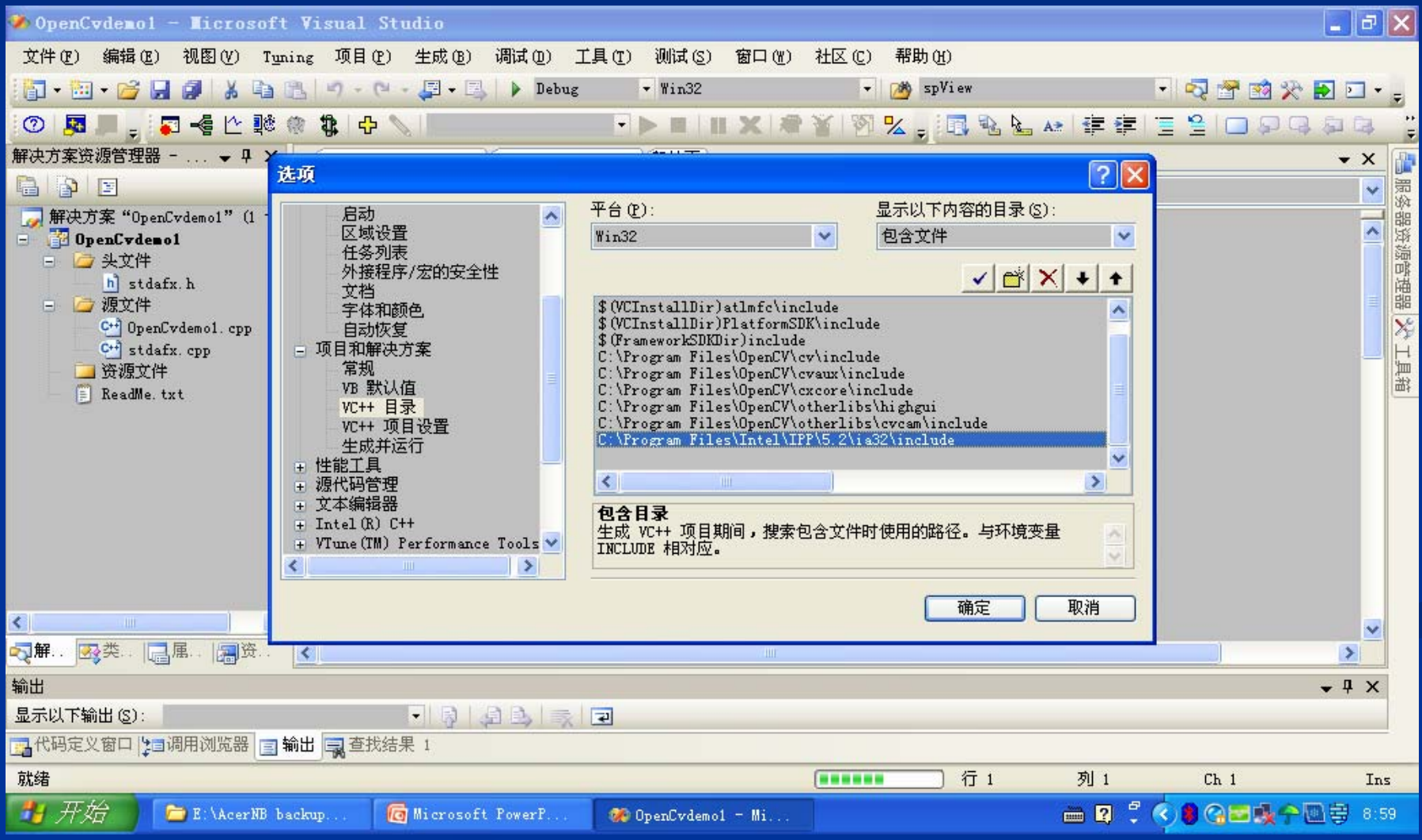


VC6.0编程链接时的库文件设置

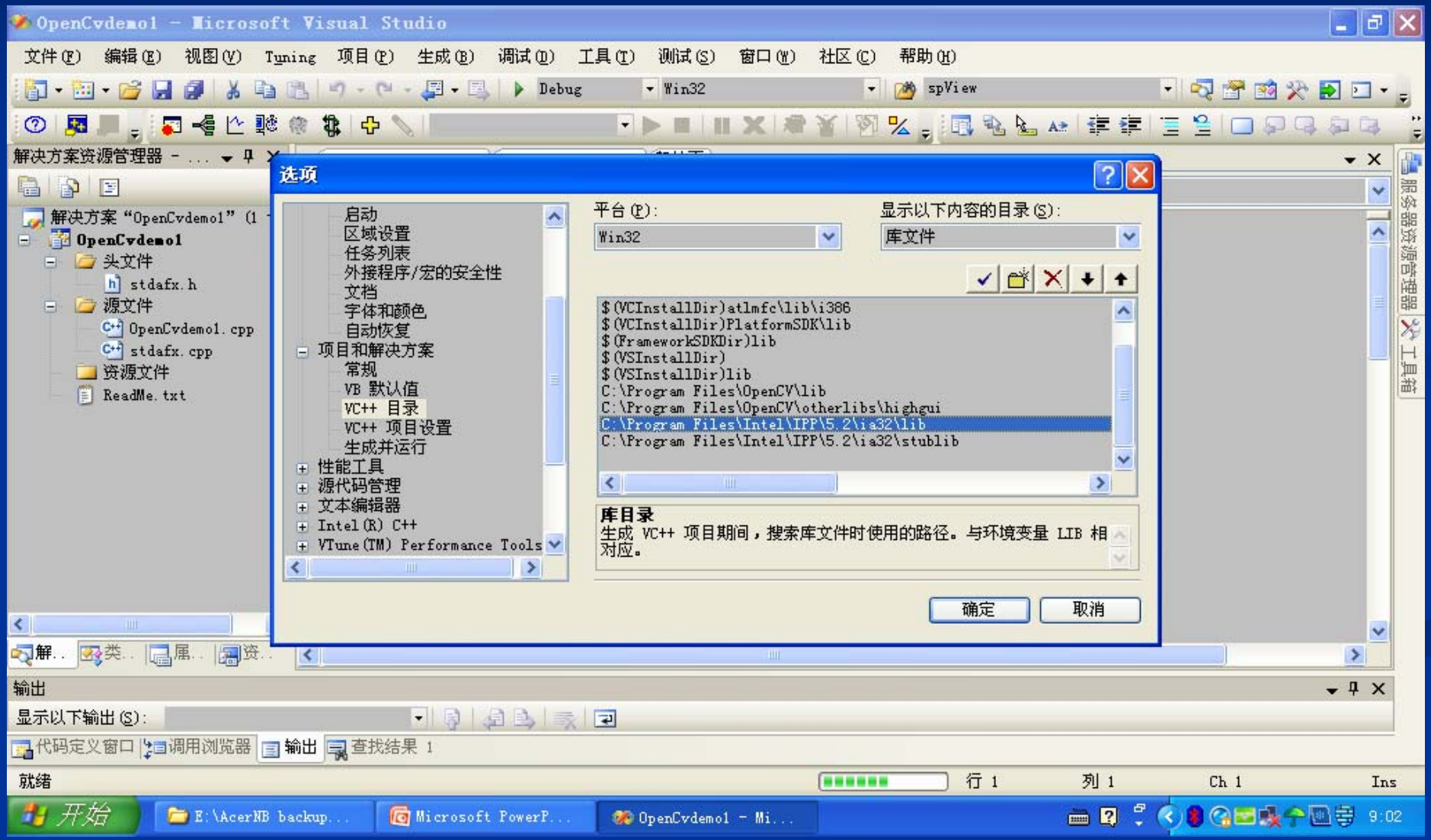


VC.NET编程头文件路径设置

在主菜单中选择“工具\选项”一项，弹出以下界面

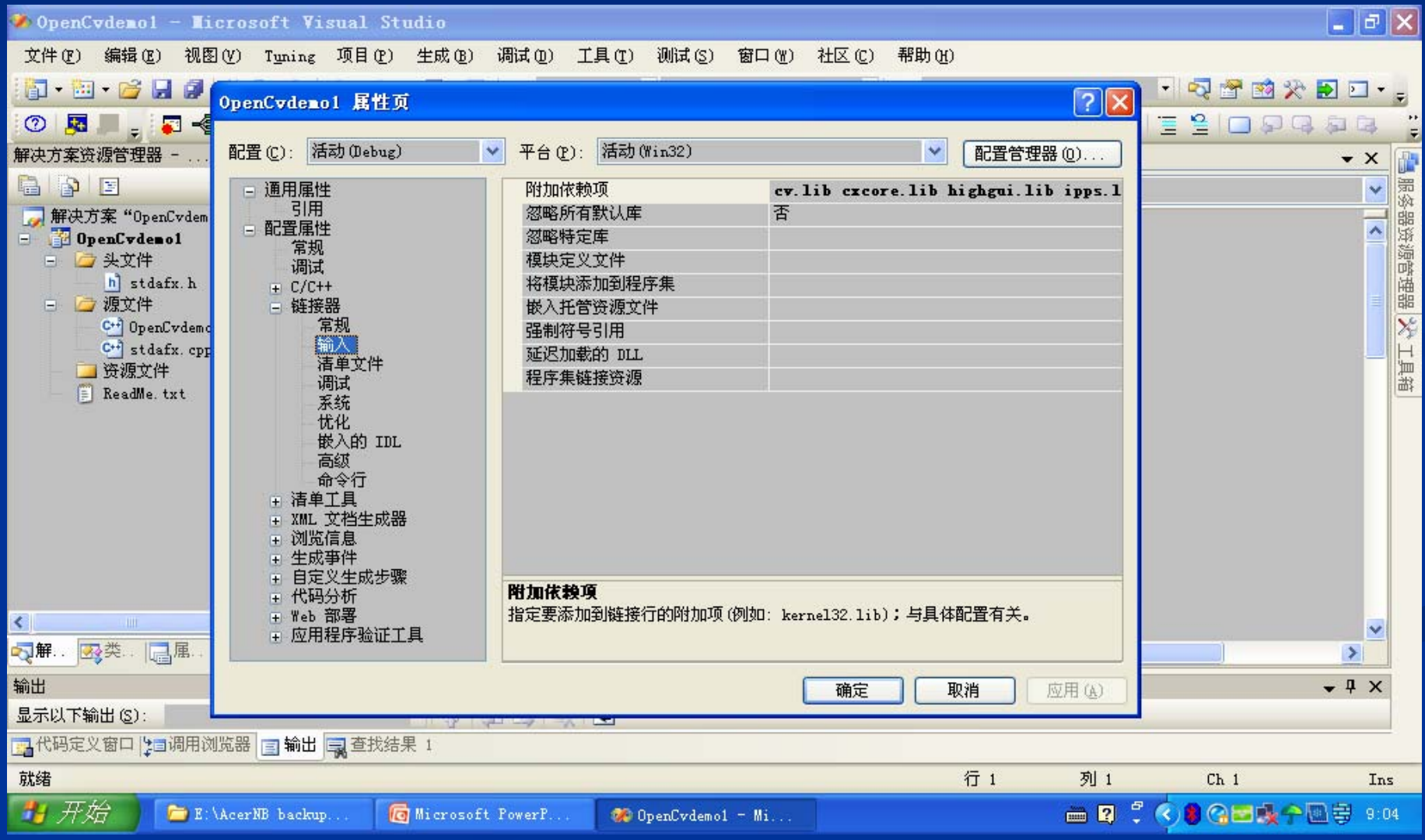


VC.NET编程库文件路径设置



VC.NET编程链接时的库文件设置

在主菜单中选择“项目\xx属性”一项，弹出以下界面



用IppGetCpuClocks测试性能

- 处理器利用时钟来组织其运算。每次处理器时钟改变状态时，处理器就执行一些计算。在几乎所有情况下，始终以固定的频率振动，所以计算始终振动的次数是一个很好的计时方式。为了允许使用该时钟做性能测试，许多Intel处理器的核包含了一个可以计算时钟振动次数的记录器。
- Intel IPP函数IppGetCpuClocks轮询这个64位记录器。两个记录器相减就能得到一个非常精确的处理器消逝时间的结果。这个结果是最细颗粒消逝时间测量方法，尽管，这要取决于程序，它并不一定是最为精确的运算性能测量方法。

运算计时编程

```
#include "stdafx.h"
#include “ipp.h”      //引入头文件
int main(int argc, char* argv[]){
        Ipp64u start, end;
        start=ippGetCpuClocks();  //得到开始CPU时钟
        end=ippGetCpuClocks();   //得到终止CPU时钟
           printf("Clocks to do nothing: %d\n",(Ipp32s)(end - start));
           start = ippGetCpuClocks();
           printf("Hello World\n");
           end = ippGetCpuClocks();
           printf("Clocks to print 'hello world': %d\n",(Ipp32s)(end - start));
          return 0;
}
```

数据拷贝示例

```
#include "stdafx.h"
#include “ipp.h”      //引入头文件
int main(int argc, char* argv[]){
      const  int  SIZE = 256;
            Ipp8u  pSrc[SIZE], pDst[SIZE];  //定义数组
            int i;
            for (i=0; i<SIZE; i++)
                  pSrc[i] = (Ipp8u)i;    //初始化数组
            ippsCopy_8u(pSrc, pDst, SIZE);    //数组拷贝 pSrc---> pDst
            printf("pDst[%d] = %d\n", SIZE-1, pDst[SIZE-1]);
            return 0;
}
```

IPP的应用

- 信号处理中的优化应用
- 图像处理中的优化应用

IPP 对信号处理的优化

IPP 为信号处理应用提供了优化的性能

- 信号的生成
- 信号的转换
- 时域频域的转换(FFTs)
- 滤波器(FFT, FIR)

Lab

IPP 对信号处理的优化

- Support
  - conj, copy, imag, real, zero, set
- Convert
  - polar/cart, complex/real
  - integer/float
  - up/down sample
- Windowing
  - Bartlett, Blackman, Hamming, Hann, Kaiser
- Signal Generation
  - random, wave patterns

- Filters
  - FIR, IIR
  - median
- Transforms
  - FFT, DFT, Goertzel, DCT, wavelet
- Statistics
  - norms, threshold, min / max / std.dev., mean, powerspectr
- Audio
  - A-law / mu-law, preemphasize

IPP中的信号发生函数

函数名	功能描述
ippsTone_Direct	产生一个正弦波
ippsTriangle_Direct	产生一个三角波
ippsRandGaussInitAlloc, ippsRandUniformInitAlloc	随机序列发生器状态初始化
ippsRandFauss, ippsRandUniform, ippsRandGauss_Direct, ippsRandUniform_Direct	产生高斯随即序列或同一分类的随机序列
ippsVectorJaehne	产生Jaehne信号
ippsVectorRamp	产生一个线性增加或线性衰减的斜坡信号

示例

- 产生长度为len、大小为0.1的Jaehne信号pSrc：ippsVectorJaehne_32f(pSrc,len,0.1)

运用ippsFIR的简单时域滤波

- 产生一个滤波系数为{0.25，0.5，0.25}的滤波器。
- void myFilterA_32f(Ipp32f* pSrc, Ipp32f*pDst, int len)
- {
- // Low-pass filter:
- Ipp32f taps[] = { 0.25f, 0.5f, 0.25f };
- Ipp32f delayLine[] = { 0.0f, 0.0f, 0.0f};
- IppsFIRState_32f *pFIRState;
- ippsFIRInitAlloc_32f(&pFIRState, taps, 3, delayLine);
- ippsFIR_32f(pSrc, pDst, len, pFIRState);
- ippsFIRFree_32f(pFIRState);
- }

IPP中的相关函数

函数名	功能描述
ippsCrossCorr_*	将两信号非标准化相关
ippsCrossCorr_NormA_*	将两信号标准化有偏相关，标准化因子为1/srcLen
ippsCrossCorr_NormB_*	将两信号标准化无偏相关，相关因子为1/(srcLen-n)
ippsAutoCorr_*	非标准化自相关
ippsAutoCorr_NormA_*	标准化有偏自相关
ippsAutoCorr-NormB_*	标准化无偏自相关

IPP中的FIR函数

函数名	功能描述
ippsFIR_*	FIR，需要初始化
ippsFIROne_*	FIR，一次采样输入，需要初始化
ippsFIR_Direct_*	FIR，不需要初始化
ippsFIROne_Direct_*	FIR，一次采样输入，不需要初始化
IppsFIRMR_Direct_*	FIR，可重复采样，不需要初始化

IPP 对图像处理的优化

IPP 为图像处理应用提供了优化的性能

- 图像生成
- 颜色转换
- 变形和过滤
- 算术和逻辑操作
- 几何变形
- 编解码支持(MPEG-1, -2, -4, H.263)

Lab

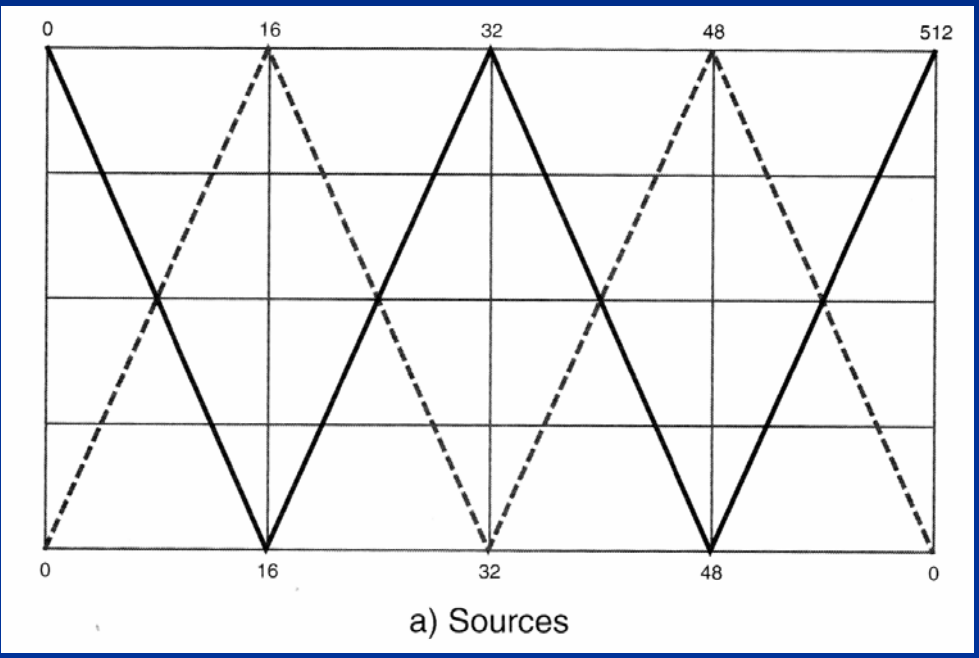
IPP 对图像处理的优化

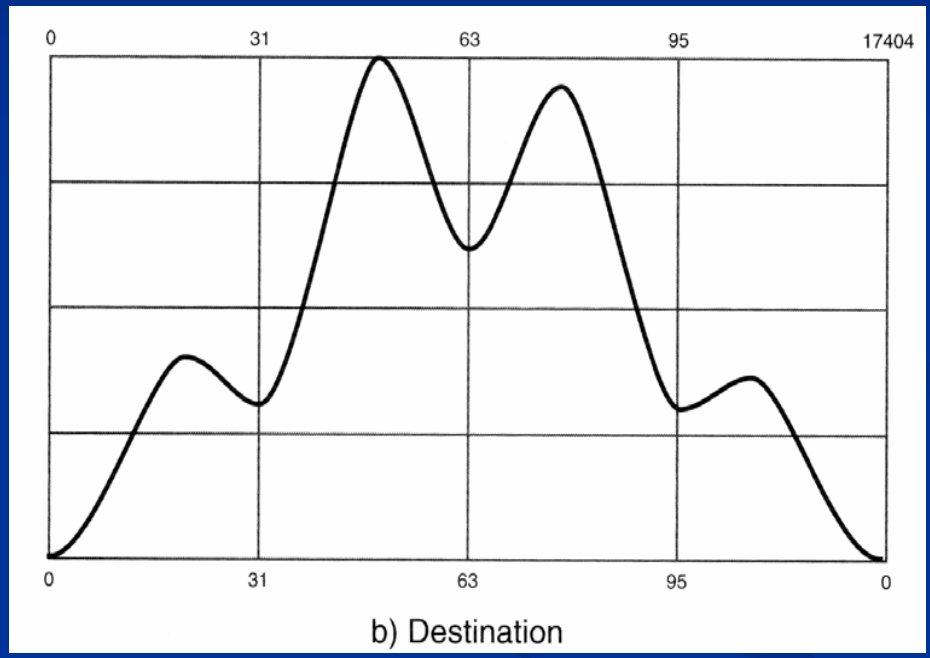
- Arithmetic & Logical
  - Abs, Add, convolve, cross-correlation, div, exp, ln, LShift, normalize, mul, RShift, sqr, sqrt, sub, threshold
  - And, not, or
  - Compare
  - Phase, magnitude
- Convert
  - Pixel/planar
  - Color conversions
- Filters
  - User-defined and built-in
- Transforms
  - FFT, DFT, DCT, wavelet
- Statistics
  - norms, threshold, min / max / std.dev., mean, powerspectr, moments
- Geometric
  - Mirror, rotate, resize, remap
- Alpha Composite
- Gamma correction
- Image Generation

图像阈值操作实例

```
void Posterize(unsigned char* pPixelData, int width,
 int height)
{
 IppiSize roi = { width, height}; //定义兴趣区域
 Ipp8u thresholds[] = { 128, 128, 128}; //定义三个通道的阈值
 Ipp8u valuesLT[] = { 0, 0, 0 }; //设定三个通道的最低值
 Ipp8u valuesGT[] = { 255, 255, 255}; //设定三个通道的最高值
 ippiThreshold_LTValGTVal_8u_C3IR(pPixelData,width*3, roi,
 Thresholds,valuesLT,thresholds,valuesGT);//调用阈值化函数
 }
```

三角波信号卷积波形图


0
16
32
48
512
0
16
32
48
0
a) Sources


0
31
63
95
17404
0
31
63
95
0
b) Destination

IPP编程完整实例

```
#include "stdafx.h"
#include "ipp.h"
#include "tools.h"
int main(int argc, char* argv[])
{
        const int len1 = 64, len2 = 64;
        Ipp16s* pSrc1 = ippsMalloc_16s(len1);    //存放源信号的数组
        Ipp16s* pSrc2 = ippsMalloc_16s(len2);  //存放源信号的数组
        Ipp16s* pDst = ippsMalloc_16s(len1+len2-1); //存放输出信号的数组
        // Generate two triangle-wave signals, with different phases  产生二个三角波信号
        float phase=3.141592654f; //相位=π
        ippsTriangle_Direct_16s(pSrc1, len1, 256.0f, 2.0/len1, 0.0f, &phase);
        ippsAddC_16s_I(256, pSrc1, len1);  //信号输出由【-256，256】归一化到【0，512】
        phase=0.0; //相位=0
        ippsTriangle_Direct_16s(pSrc2, len2, 256.0f, 2.0/len2, 0.0f, &phase);
        ippsAddC_16s_I(256, pSrc2, len2);
        // Convolve    求卷积,结果数据存放在pDst
        ippsConv_16s_Sfs(pSrc1, len1, pSrc2, len2, pDst, 8);
        ippsFree(pSrc1);   //释放内存资源
        ippsFree(pSrc2);
        ippsFree(pDst);
        return 0;
}
```

图像处理完整示例

- 利用IPP创建一幅图像并用OpenCV显示它.
- #include "cv.h"
- #include "highgui.h"
- **#include "ipp.h"**
- #include <stdio.h>
- int main()
- {
- **Ipp8u *gray = NULL;** // 定义一幅图像，类型为Ipp8u
- **IppiSize size;** // 定义存储图像大小的变量
- IplImage* img = NULL; // 定义一幅IplImage类型的图像
- CvSize sizeImg;
- int i = 0, j = 0;
- size.width = 640; size.height = 480;
- **gray = (Ipp8u *) ippsMalloc_8u(size.width * size.height);** //为图像申请内存
- for (i = 0; i < size.height; i++)
- for (j = 0; j < size.width; j++)
- *(gray + i * size.width + j) =(Ipp8u) abs(255 * cos((Ipp32f) (i * j)));// 给gray赋值

图像处理完整示例(续)

- sizeImg.width = size.width;
- sizeImg.height = size.height;
- img = cvCreateImage(sizeImg, 8, 1);
- cvSetImageData(img, gray, sizeImg.width); //将gray中数据传给img
- cvNamedWindow(“image”, 0); //创建一个新的窗口，并命名为”Image”
- cvShowImage(“image”, img); // 在”image”窗口中显示img图像
- cvWaitKey(0); // 等待关闭窗口的命令
- cvDestroyWindow(“image”); //销毁”image”窗口
- **ippsFree(gray);** //调用IPP函数释放gray所占内存
- cvReleaseImage(&img); //调用OpenCV函数释放img所占内存
- return(0);
- }

实验内容

- 参照PPT课件示例编写调试基于IPP的时钟计时、信号处理和图像处理程序。
- 利用IPP的时钟计时函数记录二个三角波卷积处理过程的时钟差和数据组拷贝过程的时钟差。