Legal Disclaimer

INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS”. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance. Buyers should consult other sources of information to evaluate the performance of systems or components they are considering purchasing. For more information on performance tests and on the performance of Intel products, reference www.intel.com/software/products.

Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

*Other names and brands may be claimed as the property of others.

Copyright © 2010. Intel Corporation.
Overview
IPP Signal Processing Domain

- Time and Frequency Domain Filtering
- General-Purpose Signal Processing Functions
  - wide range of application uses:
    - audio processing, speech recognition, motion control, etc.
    - low-pass, high-pass and band-pass filters
- General Arithmetic Functions
  - power, root and logarithmic operations
- General Vector Manipulations
  - initialize, zero, set, copy and move vectors
- Statistical, Sampling and Windowing Functions
  - min, max, mean, standard deviation, etc.
  - up/down-sampling and windowing functions
Functionalities and Algorithms
IPP Signal Processing Domain

- **Time Domain Processing**
  - FIR, IIR, LMS (Least Means Square Adaptive FIR) filters
  - Convolution and correlation functions

- **Frequency Domain Transforms**
  - FFT, DFT, DCT filters
  - Hartley, Walsh-Hadamard, Hilbert and Wavelet variants

- **Signal Generation and Initialization Functions**
  - Tone, triangular and random sequence generators
  - Vector initialization (Jaehne, slope and ramps)

- **SPIRAL Generated [Signal Processing] Functions**
  - ippg* functions → adjunct to ipps* functions
  - Machine generated by SPIRAL tool ([www.spiral.net](http://www.spiral.net))
  - Generally fastest but at expense of larger code size
## Data Types and Data Structures

### IPP Signal Processing Domain

- **Full range of integer and float data types**
  - **Integer** → 8u/s, 16u/s, 32u/s and 64s
  - **Float** → 32f and 64f
  - **Complex** → 16sc, 32fc and 64fc
    - e.g., typedef struct { Ipp32f re ; Ipp32f im ; } Ipp32fc ;

<table>
<thead>
<tr>
<th>Type</th>
<th>Usual C Type</th>
<th>Intel IPP Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>8u</td>
<td>unsigned char</td>
<td>Ipp8u</td>
</tr>
<tr>
<td>8s</td>
<td>signed char</td>
<td>Ipp8s</td>
</tr>
<tr>
<td>16u</td>
<td>unsigned short</td>
<td>Ipp16u</td>
</tr>
<tr>
<td>16s</td>
<td>signed short</td>
<td>Ipp16s</td>
</tr>
<tr>
<td>16sc</td>
<td>complex short</td>
<td>Ipp16sc</td>
</tr>
<tr>
<td>32u</td>
<td>unsigned int</td>
<td>Ipp32u</td>
</tr>
<tr>
<td>32s</td>
<td>signed int</td>
<td>Ipp32s</td>
</tr>
<tr>
<td>32f</td>
<td>float</td>
<td>Ipp32f</td>
</tr>
<tr>
<td>32fc</td>
<td>complex float</td>
<td>Ipp32fc</td>
</tr>
<tr>
<td>64s</td>
<td>__int64 (Windows*) or long (Linux*)</td>
<td>Ipp64s</td>
</tr>
<tr>
<td>64f</td>
<td>double</td>
<td>Ipp64f</td>
</tr>
<tr>
<td>64fc</td>
<td>complex double</td>
<td>Ipp64fc</td>
</tr>
</tbody>
</table>
Data Types and Data Structures
IPP Signal Processing Domain

- 24u, 24s and 16f not directly supported
  - can convert to and from 16s and 32f using ippsConvert
- 24u and 24s (24-bit integers) defined:
  - three consecutive Ipp8u bytes in little-endian byte order
  - sign is ms bit of highest order byte byte (24s only)
- 16f (16-bit floating point) defined:
  - pos/neg numbers between ~6.1e-5 and 6.5e+4
  - see image for illustration of the 16f layout
  - s → sign-bit
  - e → exponent
  - m → significand
### Signal Generation Functions

#### IPP Signal Processing Domain

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippsTone_Direct</td>
<td>Generate a sinusoid</td>
</tr>
<tr>
<td>ippsTriangle_Direct</td>
<td>Generate a triangle wave</td>
</tr>
<tr>
<td>ippsRandGaussInitAlloc, ippsRandUniformInitAlloc</td>
<td>Initialize State for random number generation</td>
</tr>
<tr>
<td>ippsRandGauss, ippsRandUniform, ippsRandGauss_Direct, ippsRandUniform_Direct</td>
<td>Generate a sequence of uniformly-distributed or Gaussian random numbers</td>
</tr>
<tr>
<td>ippsVectorJaehne</td>
<td>Generate a Jaehne signal</td>
</tr>
<tr>
<td>ippsVectorRamp</td>
<td>Generate a linearly increasing or decreasing ramp</td>
</tr>
</tbody>
</table>
Signal Generation Functions
IPP Signal Processing Domain

- **Tone (Sine) Wave**
  - ippsTone_Direct_
  - \((t*v, int, t mg, t rf, t*ph, hint)\)

- **Triangular Wave**
  - ippsTriangle_Direct_
  - \((t*v, int, t mg, t rf, t as, t*ph)\)

- **Uniform or Gaussian Random Sequence**
  - ippsRandUniform_Direct_
  - \((t*v, int, t lo, t hi, uint*seed)\)

- **Special Purpose**
  - VectorRamp:
    - \(y(n) = offset + slope*n\)
  - VectorJaehne:
    - \(y(n) = mag \times \sin(\pi n^2/(2\text{len}))\)
## FFT Memory Layout

### IPP Signal Processing Domain

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
<th>Layout</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>Real</td>
<td>$R_0 R_1 R_2 \ldots R_{N-1}$</td>
<td>N</td>
</tr>
<tr>
<td>C</td>
<td>Complex</td>
<td>$R_0 I_0 R_1 I_1 \ldots R_{N-1} I_{N-1}$</td>
<td>2N</td>
</tr>
<tr>
<td>CCS</td>
<td>Complex Conjugate-Symmetric</td>
<td>$R_0 0 R_1 I_1 \ldots R_{N/2} 0$</td>
<td>N+2</td>
</tr>
<tr>
<td>Pack</td>
<td>Packed Real-Complex</td>
<td>$R_0 R_1 I_1 \ldots R_{N/2}$</td>
<td>N</td>
</tr>
<tr>
<td>Perm</td>
<td>Permuted Real-Complex</td>
<td>$R_0 R_{N/2} R_1 I_1 \ldots R_{N/2-1} I_{N/2-1}$</td>
<td>N</td>
</tr>
</tbody>
</table>

The layout code “$R_n$” represents the real component of element $n$; “$I_n$” represents the imaginary component of element $n$. Size is the number of values required to represent an $N$-element sequence.
## FFT and Other Transform Functions

### IPP Signal Processing Domain

<table>
<thead>
<tr>
<th>Function Group</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Support Functions</td>
<td>data conversion and packed format multiplication</td>
</tr>
<tr>
<td>Fourier Transform</td>
<td>functions for fast and discrete Fourier transforms</td>
</tr>
<tr>
<td>Hartley Transform</td>
<td>a real-number alternative to the Fourier transform</td>
</tr>
<tr>
<td>Walsh-Hadamard</td>
<td>a transform that requires no multiplications</td>
</tr>
<tr>
<td>Discrete Cosine</td>
<td>commonly applied to lossy compression algorithms</td>
</tr>
<tr>
<td>Hilbert Transform</td>
<td>commonly used for fast spectral analysis</td>
</tr>
<tr>
<td>Wavelet Transform</td>
<td>used by image edge detection and data compression</td>
</tr>
</tbody>
</table>
Discrete Fourier Transform
IPP Signal Processing Domain

▪ Most Widely Used Transformation

\[ X(k) = \sum_{n=0}^{N-1} x(n)W_{N}^{nk}, \quad W_{N} = e^{-\frac{j2\pi}{N}} \]
void myFFT_RToC_32f32fc(Ipp32f* pSrc, Ipp32fc*pDst, int order)
{
    IppsFFTSpec_R_32f *pFFTSpec;
    ippsFFTInitAlloc_R_32f( &pFFTSpec, order,
        IPP_FFT_DIV_INV_BY_N, ippAlgHintFast );
    ippsFFTFwd_RToCCS_32f(pSrc, (Ipp32f*)pDst, pFFTSpec, 0 );
    ippsConjCcs_32fc_I(pDst, 1<<order);
    ippsFFTFree_R_32f(pFFTSpec);
}

...  
Ipp32f* pSrc = ippsMalloc_32f(len);
Ipp32f* pDstMag = ippsMalloc_32f(len);
Ipp32fc* pDst = ippsMalloc_32fc(len);
ippsVectorJaehne_32f( pSrc, 1<<order, 1 );
myFFT_RToC_32f32fc(pSrc, pDst, order);
ippsMagnitude_32fc(pDst, pDstMag, 1<<order);
... 

DFT Example*
IPP Signal Processing Domain

void myDFT_RToC_32f32fc(Ipp32f* pSrc, Ipp32fc*pDst, int len)
{
    IppsDFTSpec_R_32f *pDFTSpec;
    ippsDFTInitAlloc_R_32f( &pDFTSpec, len,
        IPP_FFT_DIV_INV_BY_N, ippAlgHintFast );
    ippsDFTFwd_RToCCS_32f(pSrc, (Ipp32f*)pDst, pDFTSpec, 0 );
    ippsConjCcs_32fc_I(pDst, len);
    ippsDFTFree_R_32f(pDFTSpec);
}

## Time Domain Filter Functions
### IPP Signal Processing Domain

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Convolution</td>
<td>Convolve two signals of any length; resulting length is ( \text{len1} + \text{len2} - 1 )</td>
</tr>
<tr>
<td>Correlation</td>
<td>Convolution without reversing either signal</td>
</tr>
<tr>
<td>FIR</td>
<td>Apply a finite impulse response filter to a signal; different from convolution mostly in setup, how edges are handled, and output length (which is equal to input length)</td>
</tr>
<tr>
<td>LMS</td>
<td>Adaptive FIR: Apply an FIR, but adapt the filter taps each filter cycle</td>
</tr>
<tr>
<td>IIR</td>
<td>Apply an infinite impulse response filter; like an FIR applied twice each cycle – once on the input history and once on the output history. IIR filters tend to be shorter but more complicated and less stable than FIR filters</td>
</tr>
</tbody>
</table>
## Time-Domain Filter Functions

### IPP Signal Processing Domain

<table>
<thead>
<tr>
<th>Function Group</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IIR Filters</td>
<td>infinite impulse response filters</td>
</tr>
<tr>
<td>FIR Filters</td>
<td>finite impulse response filters</td>
</tr>
<tr>
<td>IIR Generator</td>
<td>compute coefficients (taps) for IIR filters</td>
</tr>
<tr>
<td>FIR Generator</td>
<td>compute coefficients (taps) for FIR filters</td>
</tr>
<tr>
<td>Adaptive FIR</td>
<td>adaptive FIR using least mean squared (LMS) criteria</td>
</tr>
<tr>
<td>Adaptive MR FIR</td>
<td>multi-rate adaptive FIR using LMS criteria</td>
</tr>
<tr>
<td>Sampling</td>
<td>up-sampling and down-sampling functions</td>
</tr>
<tr>
<td>Convolution</td>
<td>convolution and correlation vector functions</td>
</tr>
</tbody>
</table>
Convolution
IPP Signal Processing Domain

\[ y(n) = h(n) \circ x(n) = \sum_{k=0}^{\infty} h(k)x(n-k) \]

- Fundamental DSP Operation

\[
y(3) = h(0) \times x(3) + h(1) \times x(2) + h(2) \times x(1) + h(3) \times x(0)
\]
\[
y(3) = 0 \times 1 + 1 \times 1 + 1 \times 1 + 1 \times 0
\]
\[
y(3) = 2
\]

- SIMD Favored (Optimal) Operation
  - MAC (multiply and accumulate) on an array or vector
Time-Domain Filters
IPP Signal Processing Domain

- More SIMD Favored (Optimal) Operations
  - MAC (multiply and accumulate) on an array or vector

- FIR (Finite Impulse Response) Filter
  - ippsFIR_\langle t\rangle ( t*x, t*y, int n, t*M, ... )

\[
y(n) = \sum_{k=0}^{M-1} h(k)x(n-k)
\]

- IIR (Infinite Impulse Response) Filter
  - ippsIIR_\langle t\rangle ( t*x, t*y, int n, t*order, ... )

\[
y(n) = \sum_{k=0}^{order} b(k)x(n-k) - \sum_{1}^{order} a(k)y(n-k)
\]
Adaptive FIR Filters
IPP Signal Processing Domain

- Adapt FIR Coefficients to Converge on Reference
  - ippsFIRLMSOne_\(<t>\) (t \(*x\), t \(*d\), t \(*y\), int M, float mu, t \(*h\))

\[
y(n) = \sum_{i=0}^{M-1} h(i)x(n-1)
\]

\[
e(n) = y(n) - d(n)
\]

\[
h_{n+1}(i) = h_{n}(i) + 2\mu \cdot e(n)x(n-i)
\]
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);
}

Performance
IPP Signal Processing Domain

single-precision real-data, 1d transforms

powers of two

higher is better (faster)

www.fftw.org/speed/CoreDuo-3.0GHz-icc64/

*Other brands and names are the property of their respective owners.
Usage & Applications
IPP Signal Processing Domain

- Noise Filtering – Before and After
Where to Go for More Information
IPP Signal Processing Domain

- Intel IPP Product Page
  www.intel.com/software/products/ipp/

- Intel IPP Free Code Samples

- Intel IPP Forum

- Intel IPP Knowledge Base
  software.intel.com/en-us/articles/intel-ipp-kb/all/1/

- Intel IPP Documentation

- Intel IPP Books by Stewart Taylor
  www.intel.com/intelpress/sum_ipp.htm
  www.intel.com/intelpress/sum_ipp2.htm