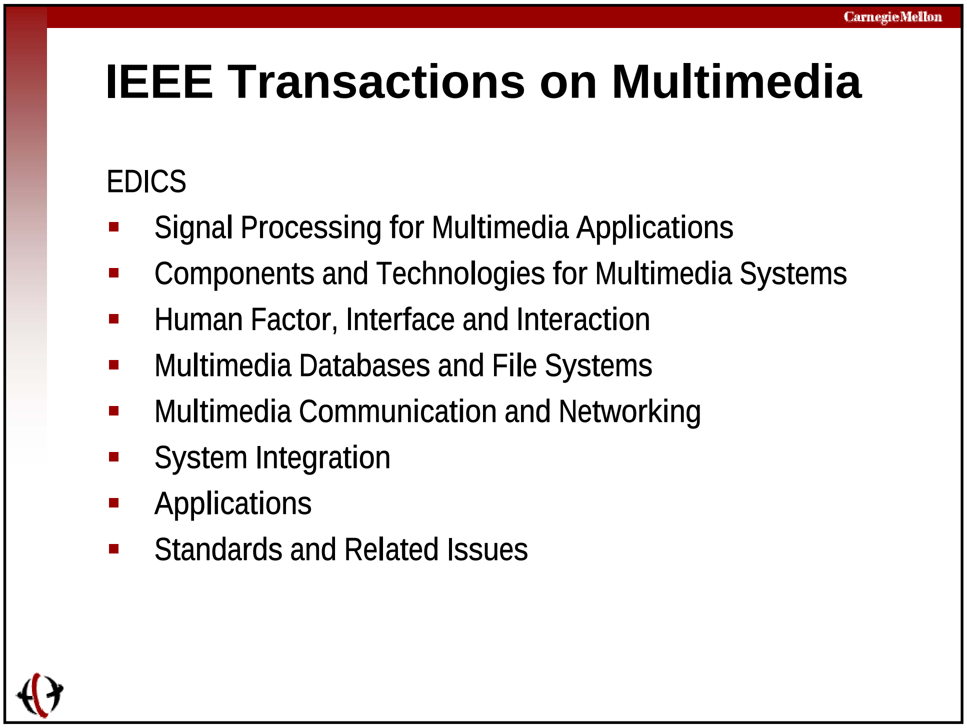
Lightweight Arithmetic for Mobile Multimedia Devices

Tsuhan Chen
Carnegie Mellon University
tsuhan@cmu.edu

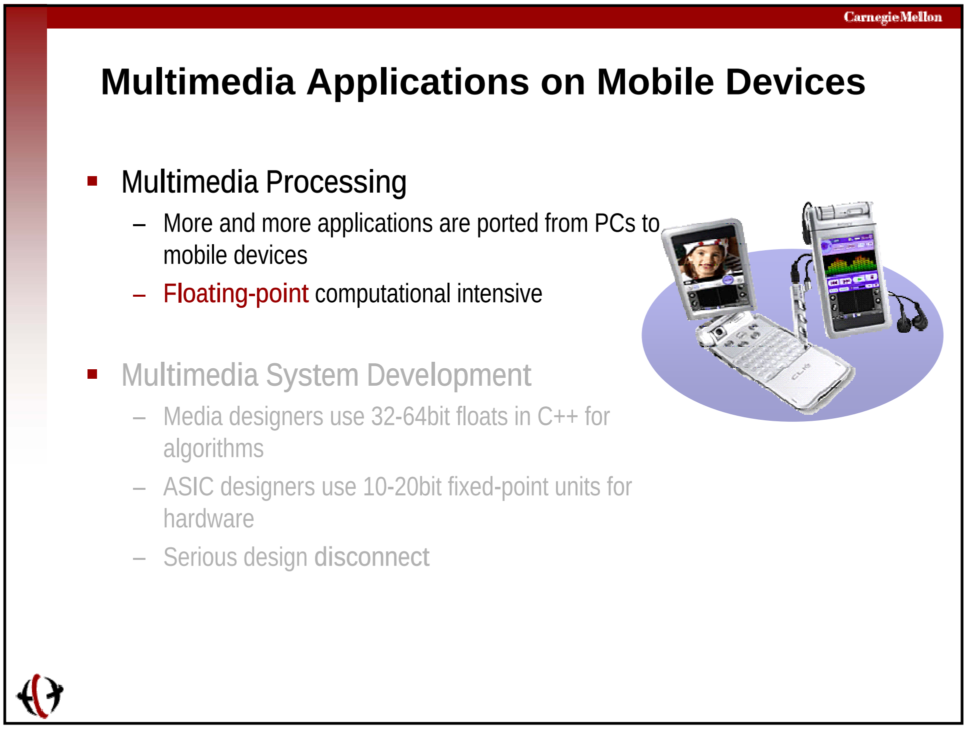
Thanks to Fang Fang and Rob Rutenbar



IEEE Transactions on Multimedia

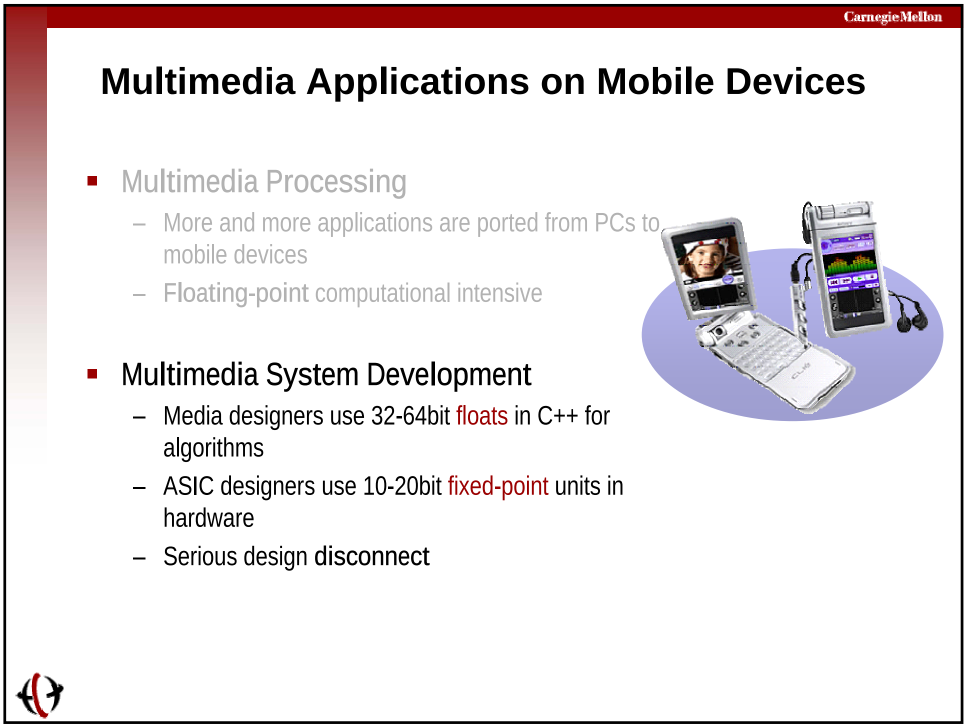
EDICS

- Signal Processing for Multimedia Applications
- Components and Technologies for Multimedia Systems
- Human Factor, Interface and Interaction
- Multimedia Databases and File Systems
- Multimedia Communication and Networking
- System Integration
- Applications
- Standards and Related Issues



Multimedia Applications on Mobile Devices

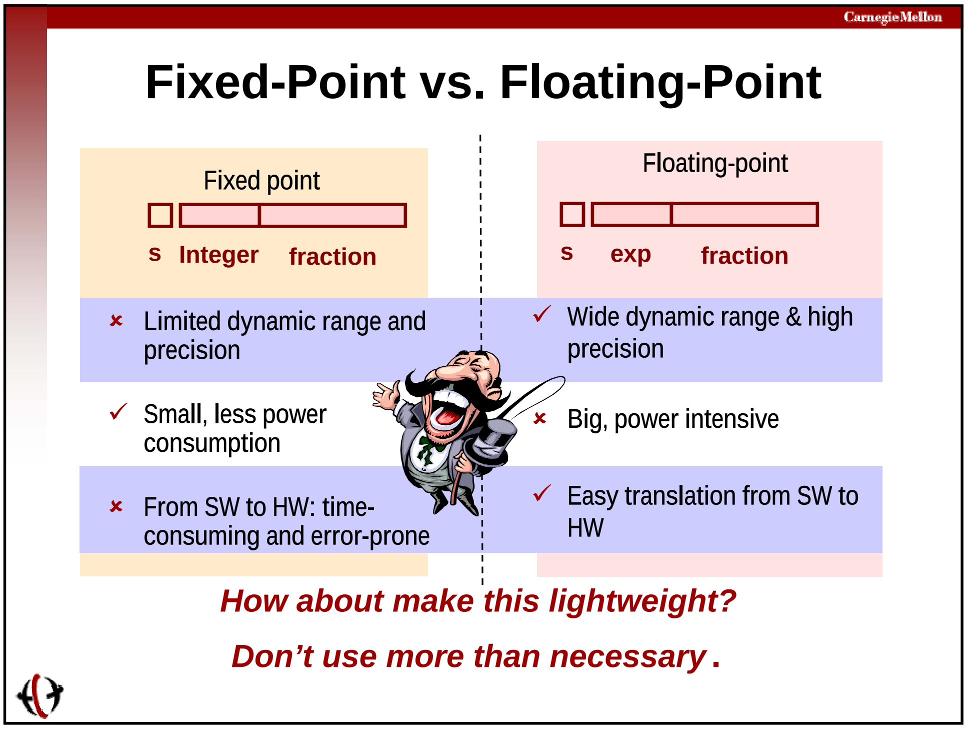
- Multimedia Processing
  - More and more applications are ported from PCs to mobile devices
  - Floating-point computational intensive
- Multimedia System Development
  - Media designers use 32-64bit floats in C++ for algorithms
  - ASIC designers use 10-20bit fixed-point units for hardware
  - Serious design disconnect



Carnegie Mellon

Multimedia Applications on Mobile Devices

- Multimedia Processing
  - More and more applications are ported from PCs to mobile devices
  - Floating-point computational intensive
- Multimedia System Development
  - Media designers use 32-64bit floats in C++ for algorithms
  - ASIC designers use 10-20bit fixed-point units in hardware
  - Serious design disconnect



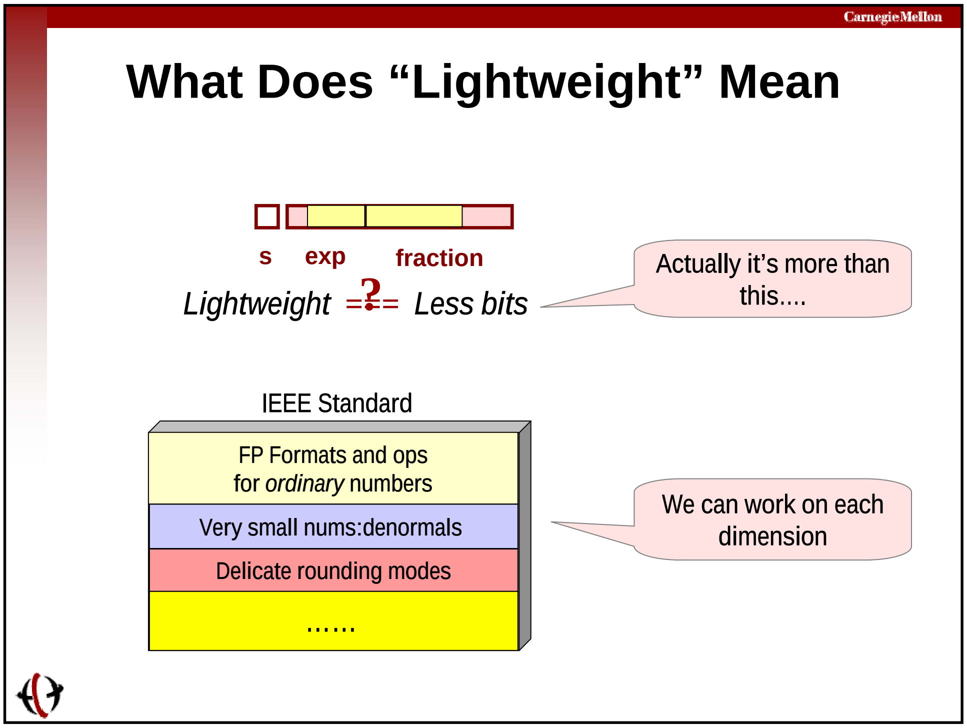
CarnegieMellon

Fixed-Point vs. Floating-Point

Fixed point	Floating-point
s Integer fraction	s exp fraction
× Limited dynamic range and precision	✓ Wide dynamic range & high precision
✓ Small, less power consumption	× Big, power intensive
× From SW to HW: time-consuming and error-prone	✓ Easy translation from SW to HW

How about make this lightweight?

Don't use more than necessary.



CarnegieMellon

What Does "Lightweight" Mean

s exp fraction

Lightweight =?= *Less bits*

Actually it's more than this....

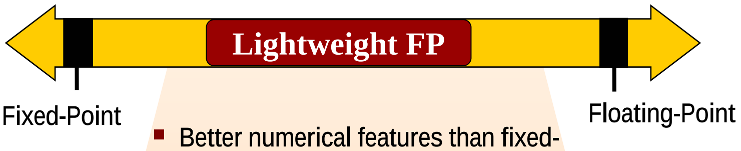
IEEE Standard

- FP Formats and ops for *ordinary* numbers
- Very small nums:denormals
- Delicate rounding modes
- ……

We can work on each dimension

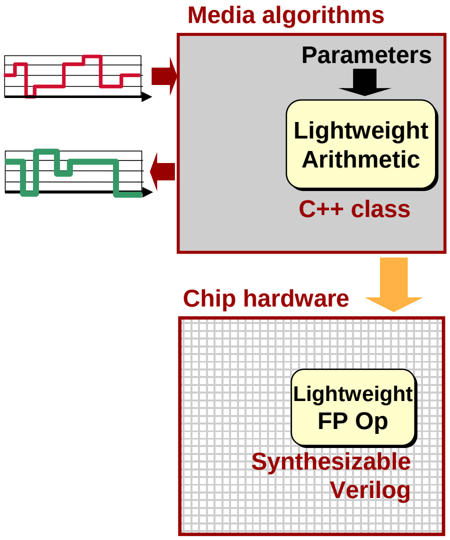
Lightweight Floating-Point Arithmetic

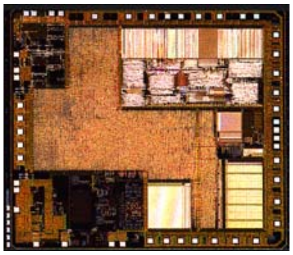
- Lightweight FP arithmetic is a *middle-ground* solution

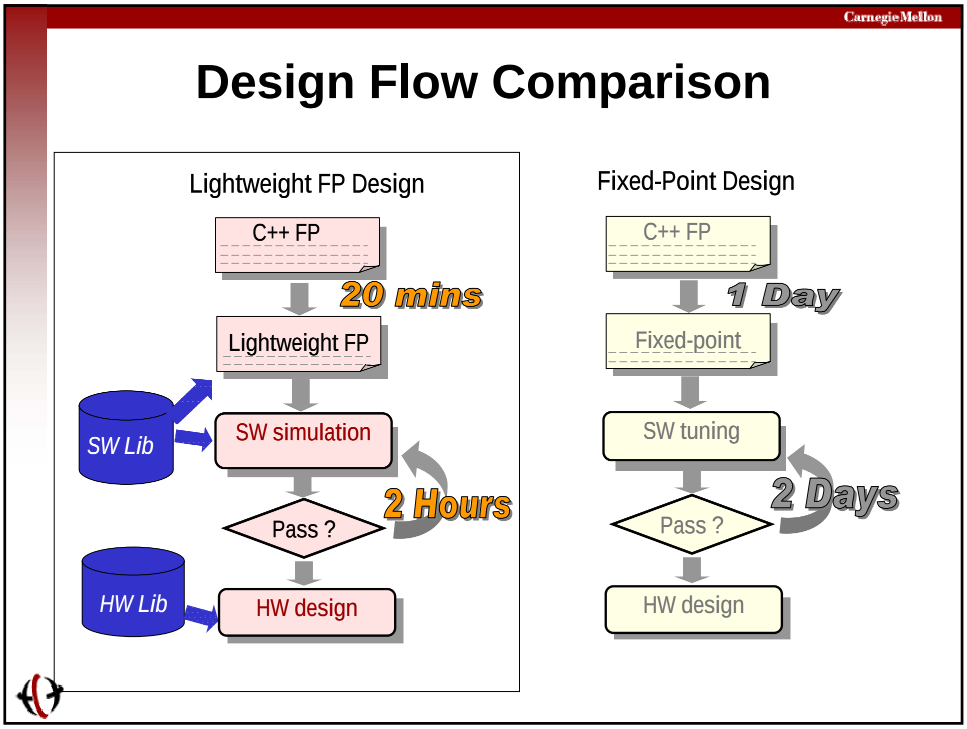


- Better numerical features than fixed-point
- Less complicated than IEEE FP
- Acceptable energy consumption
- Easy to prototype algorithms with
- Easy to implement into hardware

Software to Hardware Cycles







Design Flow Comparison

Lightweight FP Design

C++ FP

20 mins

Lightweight FP

SW Lib

SW simulation

2 Hours

Pass ?

HW Lib

HW design

Fixed-Point Design

C++ FP

1 Day

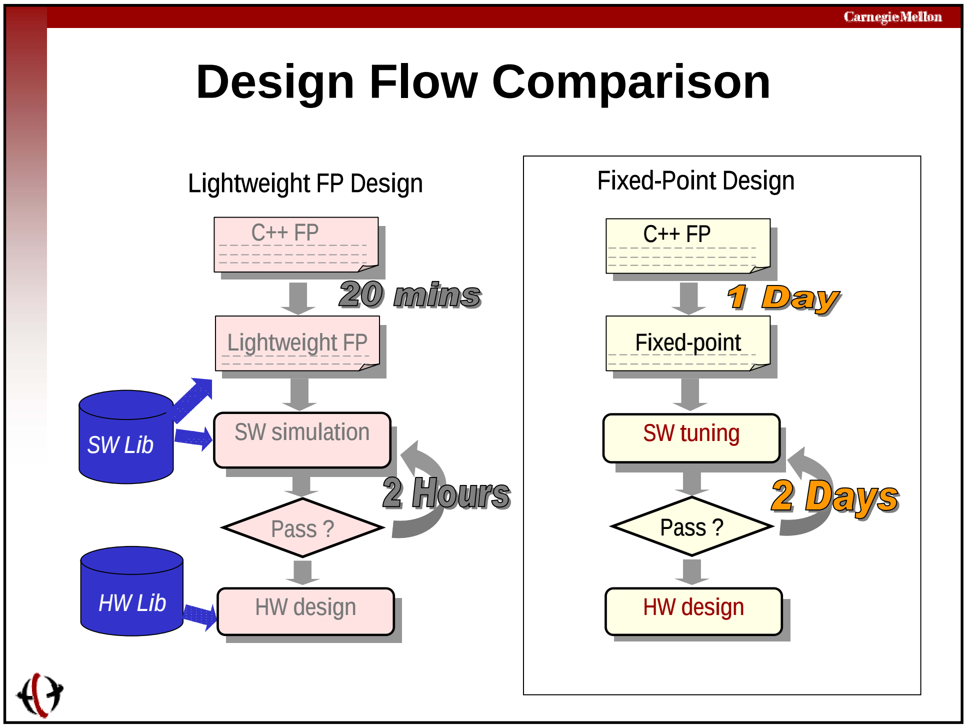
Fixed-point

SW tuning

2 Days

Pass ?

HW design



Design Flow Comparison

Lightweight FP Design

C++ FP

20 mins

Lightweight FP

SW Lib

SW simulation

2 Hours

Pass ?

HW Lib

HW design

Fixed-Point Design

C++ FP

1 Day

Fixed-point

SW tuning

2 Days

Pass ?

HW design

IEEE Standard vs. Lightweight IP

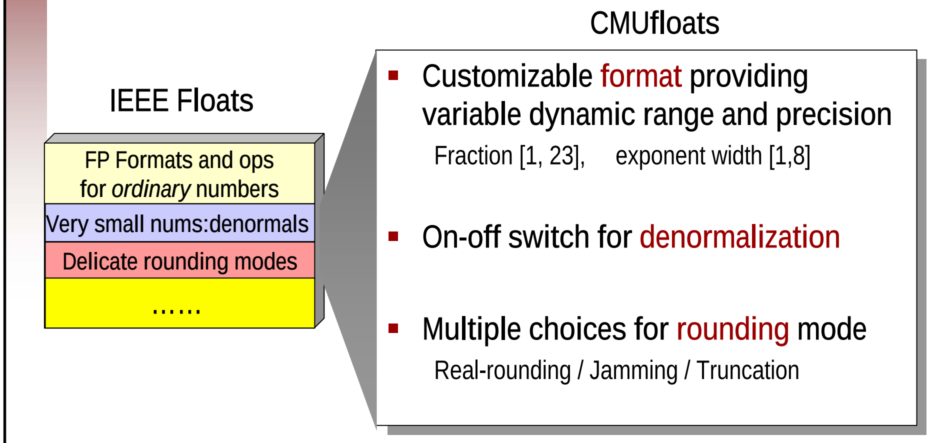
IEEE FP Standard

- 32 / 64 bits
  - 8 / 11 bits exponent
  - 23 / 52 bits mantissa
  - 1 sign bit
- Specs
  normal numbers *as well as*
  special values (infinity),
  edge cases (INF - INF), etc.

Lightweight Arithmetic IP

- *Fewer bits*
  - Fewer bits of fraction
    → less numerical precision
  - Fewer bits of exponent
    → less dynamic range
- Which of the special cases/numbers should be supported?

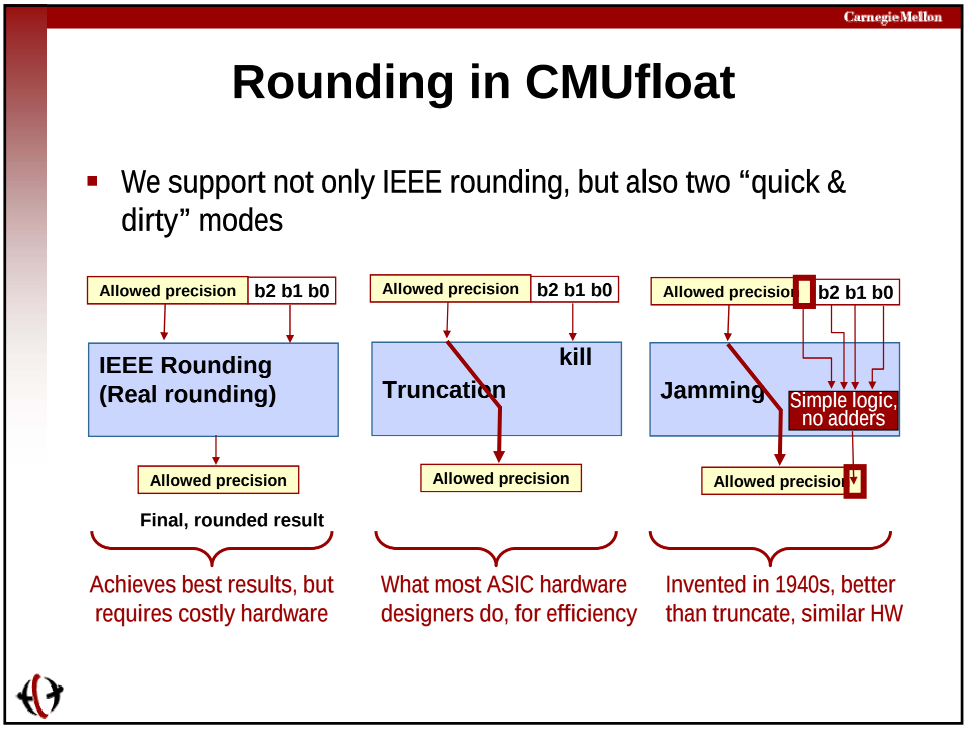
IEEE Floats vs. CMUfloats



IEEE Floats

CMUfloats

- Customizable format providing variable dynamic range and precision
  Fraction [1, 23],   exponent width [1,8]
- On-off switch for denormalization
- Multiple choices for rounding mode
  Real-rounding / Jamming / Truncation



Rounding in CMUfloat

- We support not only IEEE rounding, but also two “quick & dirty” modes

Allowed precision | b2 b1 b0

IEEE Rounding (Real rounding)

Allowed precision

Final, rounded result

Achieves best results, but requires costly hardware

Allowed precision | b2 b1 b0

Truncation

kill

Allowed precision

What most ASIC hardware designers do, for efficiency

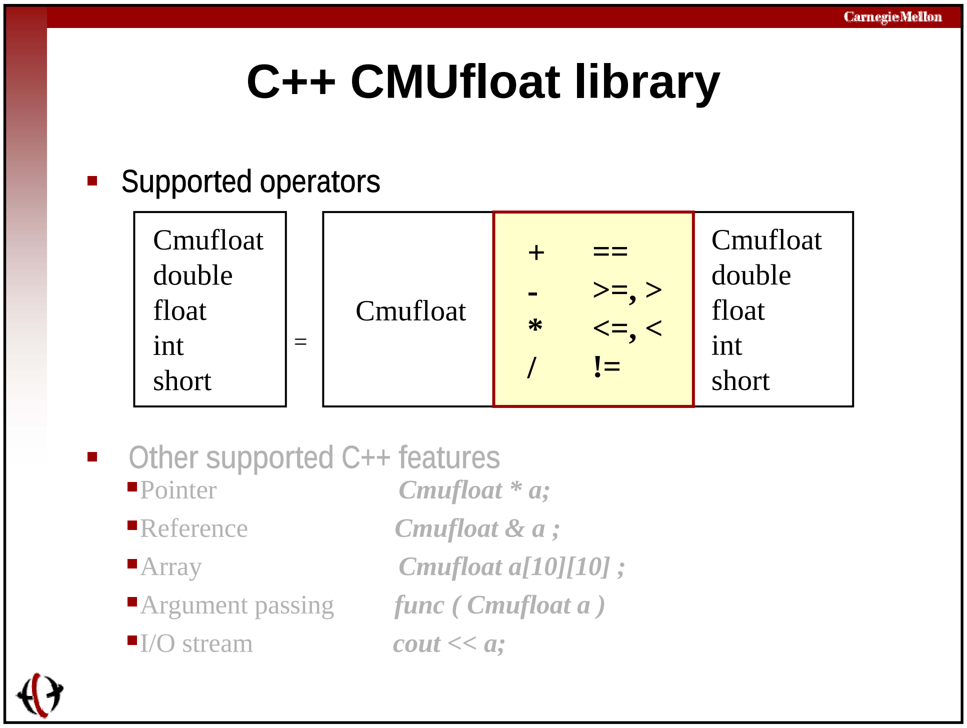
Allowed precisio | b2 b1 b0

Jamming

Simple logic, no adders

Allowed precisio

Invented in 1940s, better than truncate, similar HW

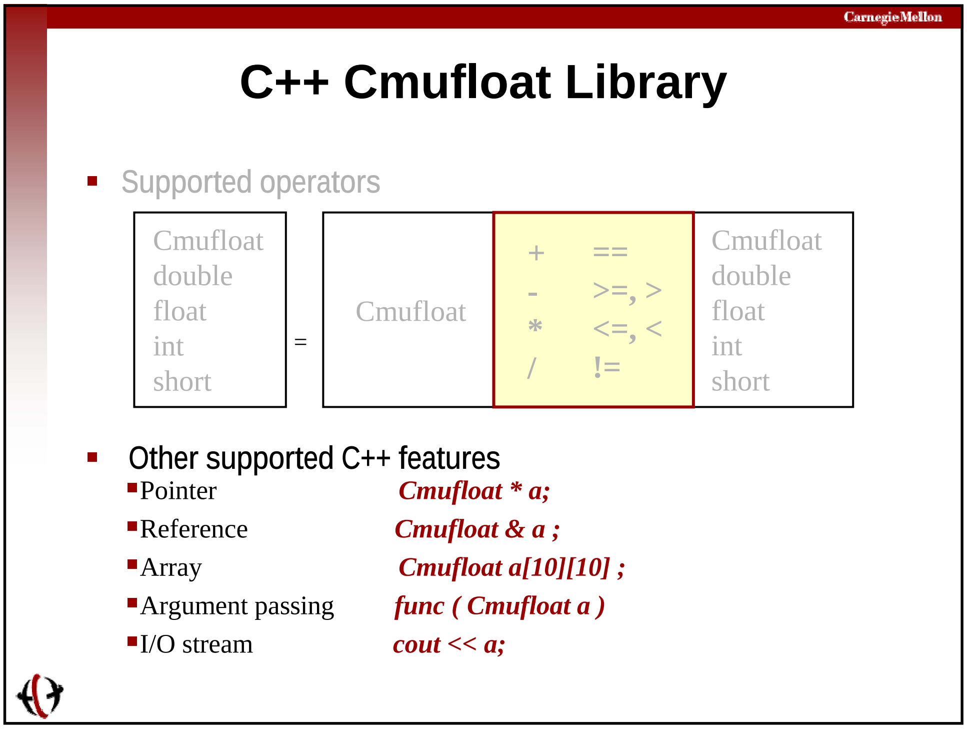


C++ CMUfloat library

- Supported operators

Cmufloat double float int short	=	Cmufloat	+ == - >=, > * <=, < / !=	Cmufloat double float int short

- Other supported C++ features
  - Pointer *Cmufloat * a;*
  - Reference *Cmufloat & a ;*
  - Array *Cmufloat a[10][10] ;*
  - Argument passing *func (Cmufloat a)*
  - I/O stream *cout << a;*



CarnegieMellon

C++ Cmufloat Library

- Supported operators

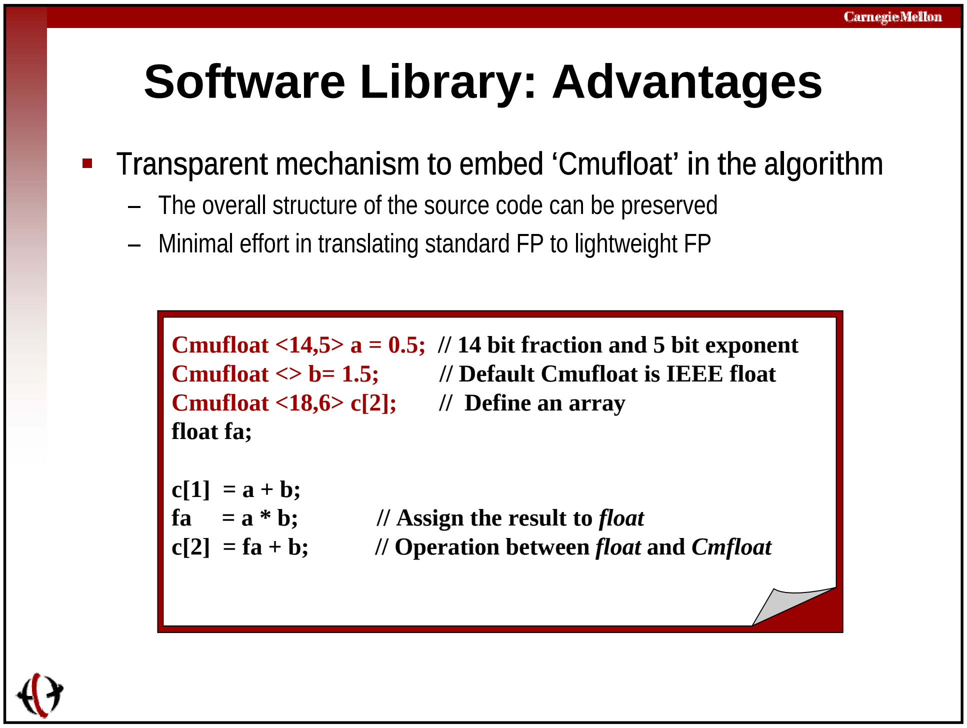
Cmufloat
double
float
int
short = Cmufloat

Cmufloat double float int short	=	Cmufloat

+ - * /	== >=, > <=, < !=	Cmufloat double float int short

- Other supported C++ features

Pointer	*Cmufloat * a;*
Reference	*Cmufloat & a ;*
Array	*Cmufloat a[10][10] ;*
Argument passing	*func (Cmufloat a)*
I/O stream	*cout << a;*



CarnegieMellon

Software Library: Advantages

- Transparent mechanism to embed 'Cmufloat' in the algorithm
  - The overall structure of the source code can be preserved
  - Minimal effort in translating standard FP to lightweight FP

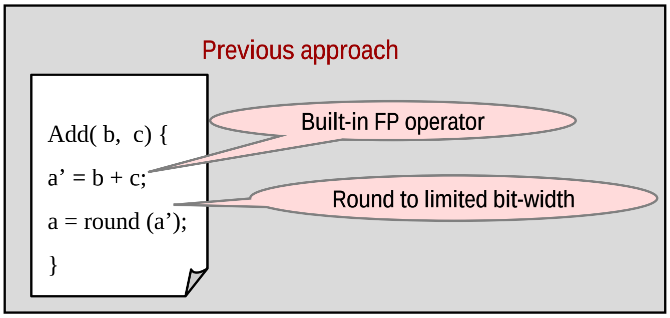
```
Cmufloat <14,5> a = 0.5;  // 14 bit fraction and 5 bit exponent
Cmufloat <> b= 1.5;       // Default Cmufloat is IEEE float
Cmufloat <18,6> c[2];     //  Define an array
float fa;

c[1]  = a + b;
fa    = a * b;          // Assign the result to float
c[2]  = fa + b;         // Operation between float and Cmfloat
```

Software Library: Advantages (Cont.)

- Arithmetic operators are implemented by bit-level manipulation: more precise

Our approach:
Emulates the hardware implementation exactly

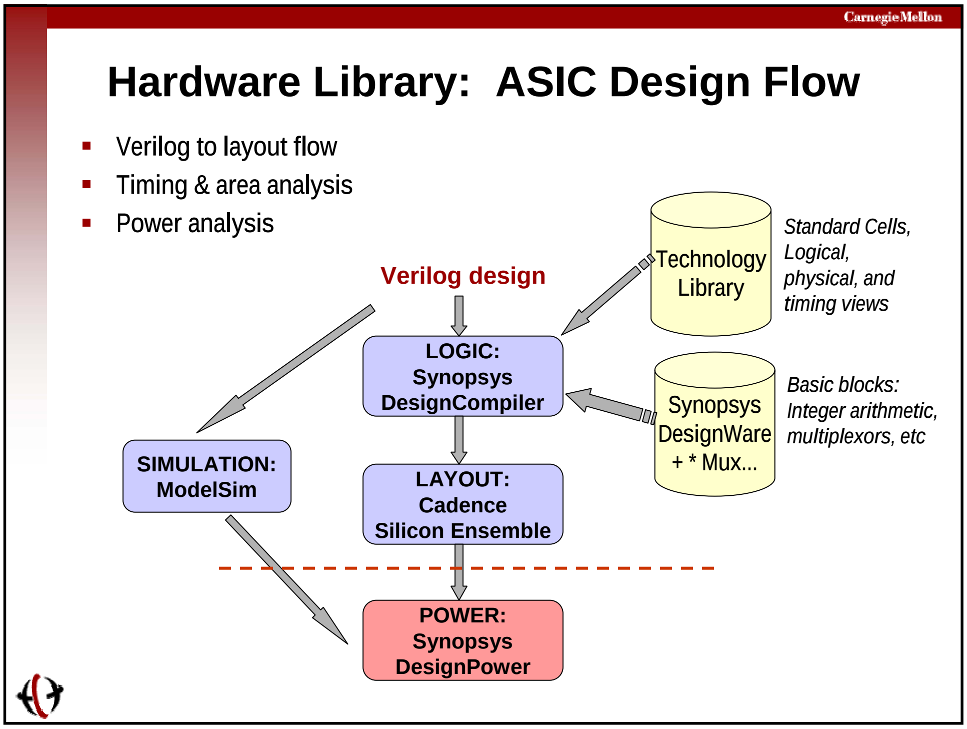


Previous approach

```
Add( b,  c) {
a' = b + c;
a = round (a');
}
```

Summary: Features Supported

- Bit widths
  - Variable from 2 bits (1 sign + 1 exp + 0 man) to 32 bits (IEEE std)
- Rounding
  - Use jamming (1.00011 rounds to 1.01)
  - Experiments show jamming is nearly as good as full IEEE rounding, always superior to truncation, yet same complexity as truncation
- Denormalized numbers
  - Not supported--our experiments on video/audio codecs suggest that denormal numbers do not improve the performance
- Exceptions
  - Support only the exceptional values for infinity, zero and NAN
  - Helps make the smaller FP sizes more robust

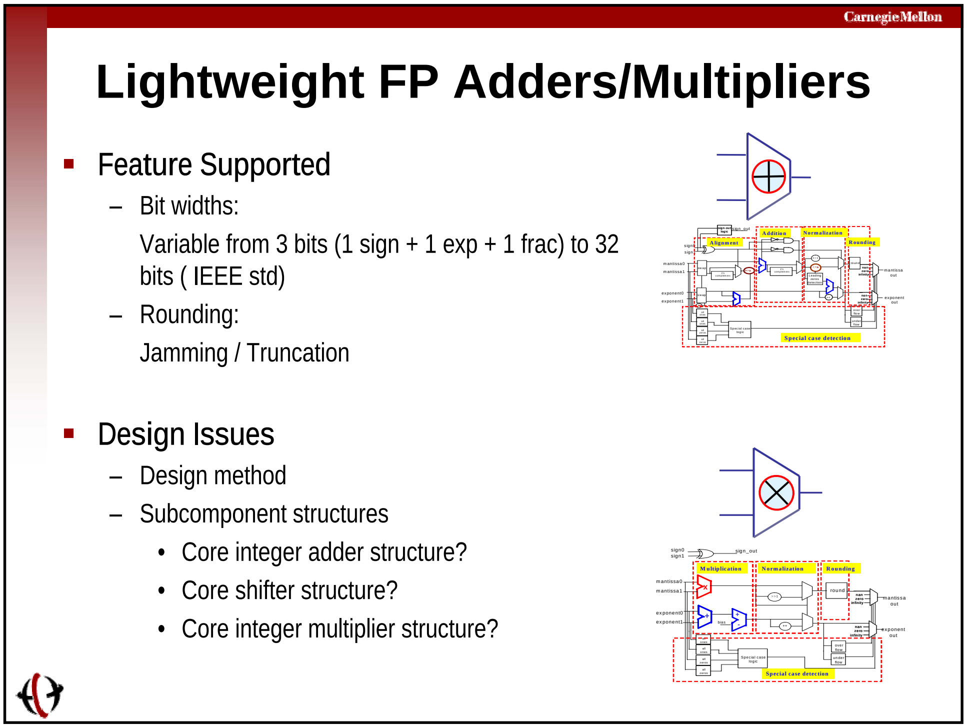


CarnegieMellon

Hardware Library: ASIC Design Flow

- Verilog to layout flow
- Timing & area analysis
- Power analysis

Verilog design
LOGIC: Synopsys DesignCompiler
LAYOUT: Cadence Silicon Ensemble
POWER: Synopsys DesignPower
SIMULATION: ModelSim
Technology Library
Standard Cells, Logical, physical, and timing views
Synopsys DesignWare + * Mux...
Basic blocks: Integer arithmetic, multiplexors, etc



CarnegieMellon

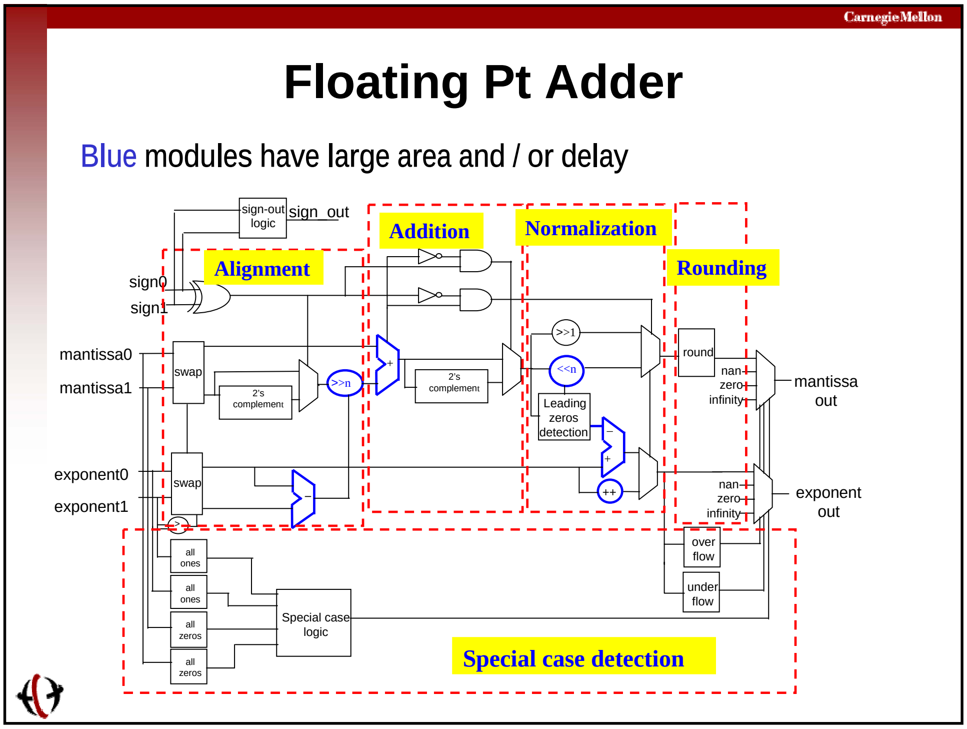
Lightweight FP Adders/Multipliers

- Feature Supported
  - Bit widths:

    Variable from 3 bits (1 sign + 1 exp + 1 frac) to 32 bits (IEEE std)
  - Rounding:

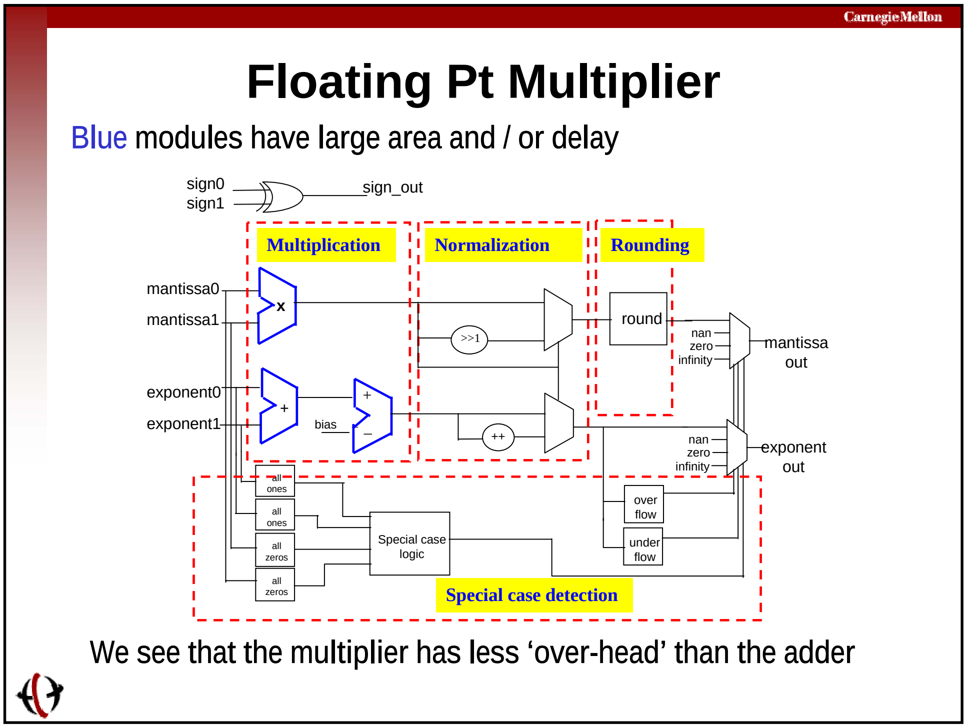
    Jamming / Truncation
- Design Issues
  - Design method
  - Subcomponent structures
    - Core integer adder structure?
    - Core shifter structure?
    - Core integer multiplier structure?

sign-out logic
sign_out
sign0
sign1
Alignment
Addition
Normalization
Rounding
mantissa0
mantissa1
exponent0
exponent1
swap
2's complement
>>n
<<n
>>1
Leading zeros detection
++
round
nan
zero
infinity
mantissa out
exponent out
all ones
all zeros
Special case logic
over flow
under flow
Special case detection
sign0
sign1
sign_out
Multiplication
Normalization
Rounding
mantissa0
mantissa1
exponent0
exponent1
bias
>>1
++
round
nan
zero
infinity
mantissa out
exponent out
all ones
all zeros
Special case logic
over flow
under flow
Special case detection



Floating Pt Adder

Blue modules have large area and / or delay



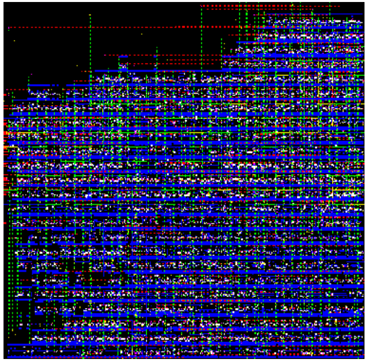
Floating Pt Multiplier

Blue modules have large area and / or delay

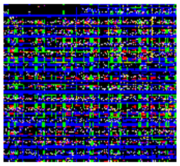
We see that the multiplier has less ‘over-head’ than the adder

Design Examples: Adders

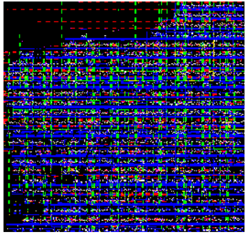
32-bit Floating-point



20-bit Fixed-pt



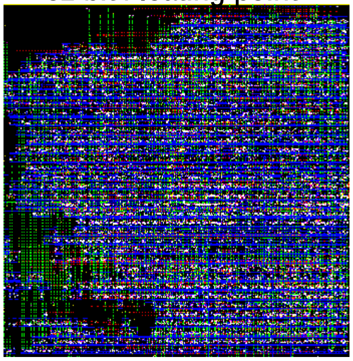
14-bit Floating-pt



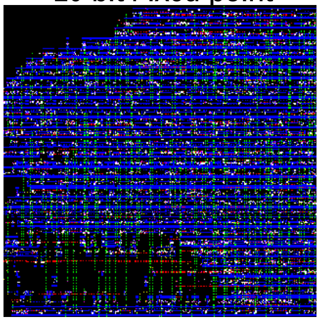
	32-bit FP	20-bit FIX	14-bit FP
Area(um^2) - post layout	26634	4866	10096
Delay(ns) - post synthesys	48.95	2.44	25.77

Design Examples: Multipliers

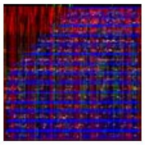
32-bit Floating-point



20-bit Fixed-point



14-bit Floating-pt



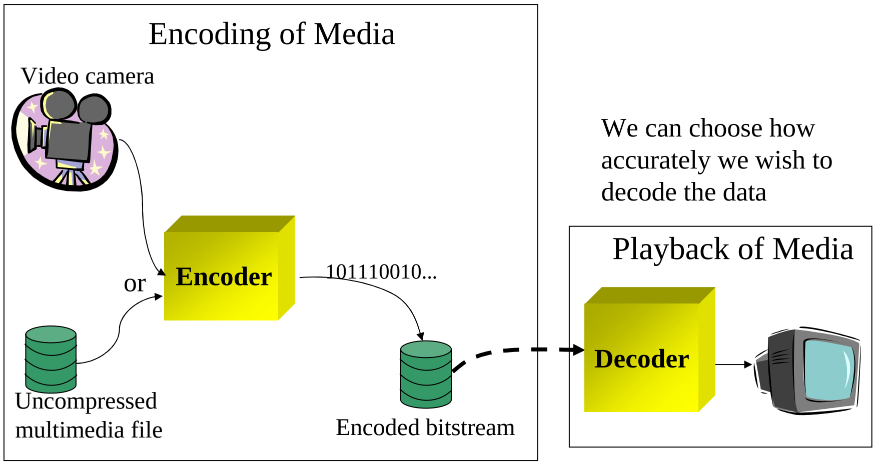
	32-bit FP	20-bit FIX	14-bit FP
Area(um^2) - post layout	60713	40738	8851
Delay(ns) - post synthesis	24.14	22.82	15.89

Power Analysis

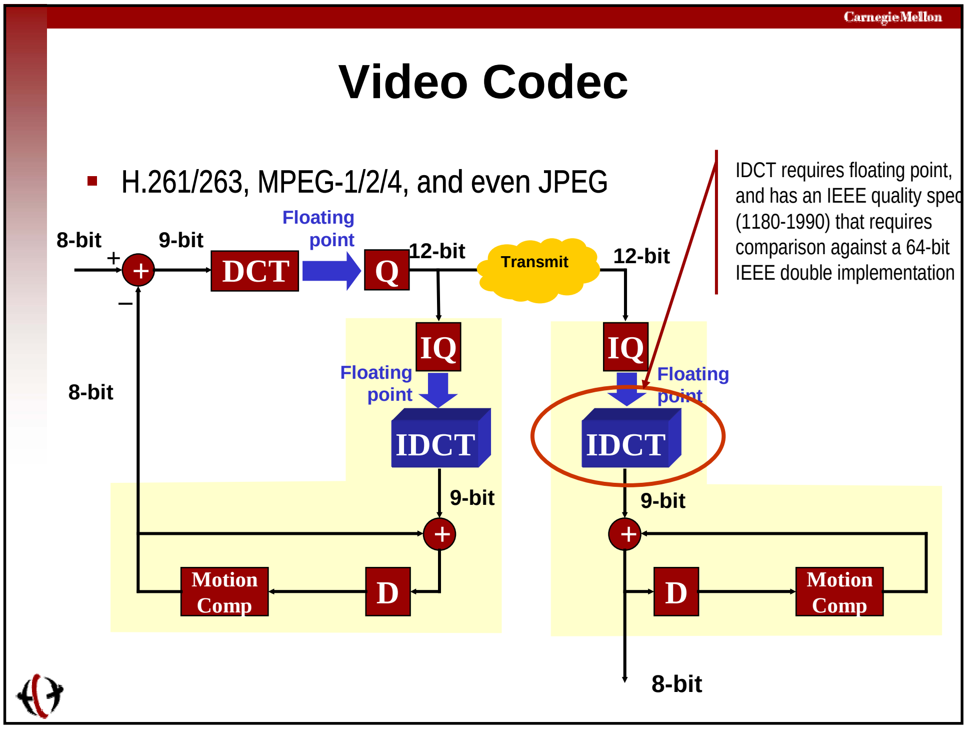
- IDCT in
  - 32-bit IEEE FP
  - 15-bit radix-16 lightweight FP
  - Fixed-point implementation
    - 12-bit accuracy for constants
    - Widest bit-width is 24 in the whole algorithm (not fine tuned)

Implementation	Area(um^2)	Delay(ns)	Power(mw)
IEEE FP	926810	111	1360
Lightweight FP	216236	46.75	143
Fixed-point	106598	36.11	110

Multimedia Encoding/Decoding



We can choose how accurately we wish to decode the data

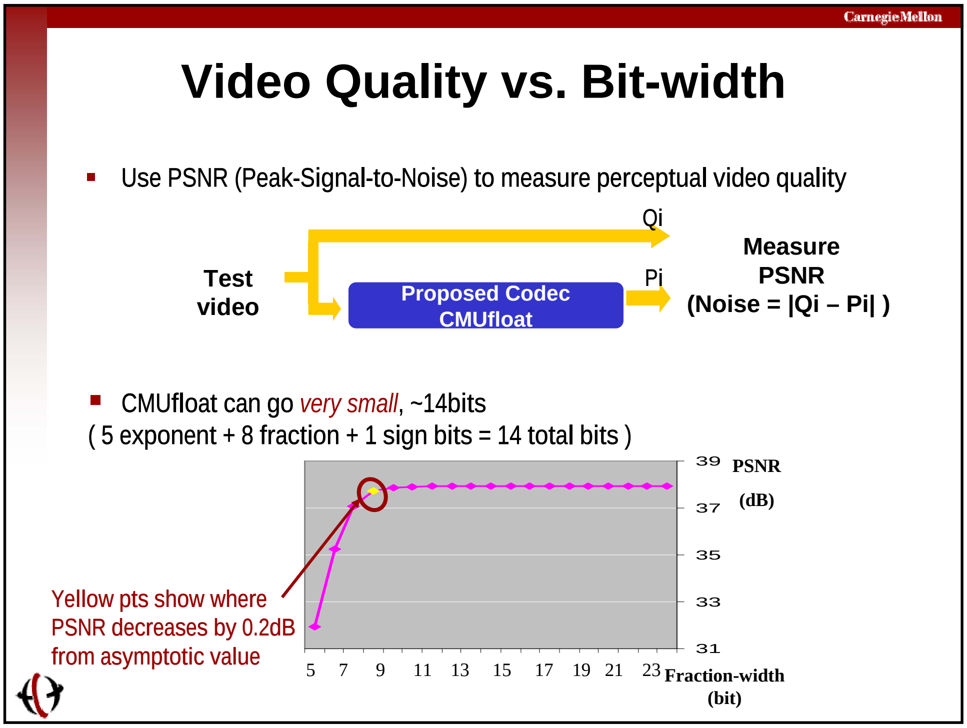


CarnegieMellon

Video Codec

- H.261/263, MPEG-1/2/4, and even JPEG

IDCT requires floating point, and has an IEEE quality spec (1180-1990) that requires comparison against a 64-bit IEEE double implementation



CarnegieMellon

Video Quality vs. Bit-width

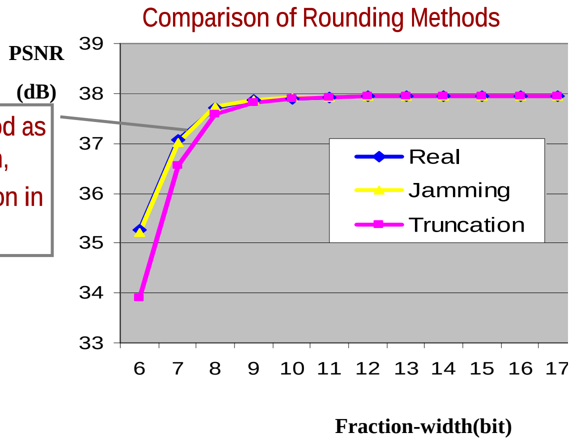
- Use PSNR (Peak-Signal-to-Noise) to measure perceptual video quality

- CMUfloat can go *very small*, ~14bits
(5 exponent + 8 fraction + 1 sign bits = 14 total bits)

Yellow pts show where PSNR decreases by 0.2dB from asymptotic value

Rounding Modes

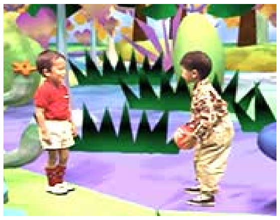
- Compare 3 rounding modes using IDCT video streams



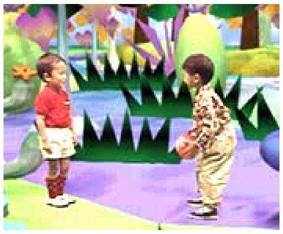
Jamming is nearly as good as real rounding in precision, but as simple as truncation in hardware.

Video Demo

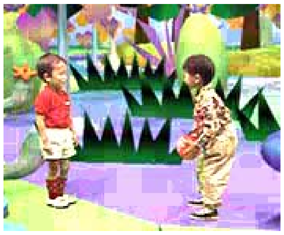
– IEEE double vs. variable-precision CMUfloats



Decoded with 14-bit “lightweight” IDCT



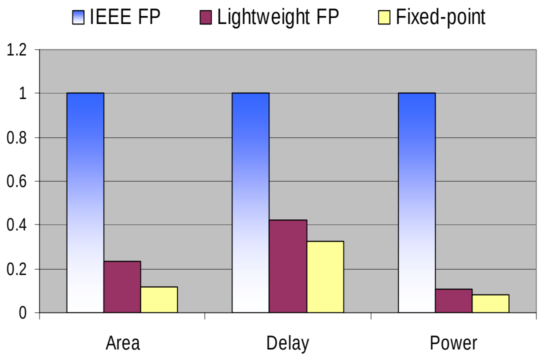
Decoded with 64-bit “double” IDCT



Decoded with 11-bit “lightweight” IDCT

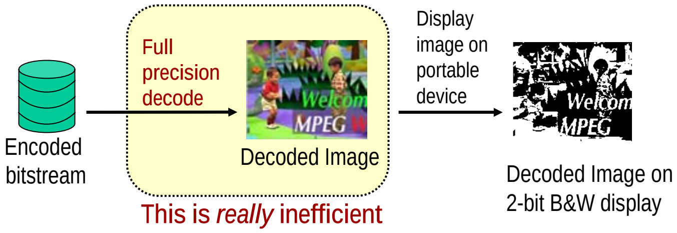
Hardware Reduction Using Lightweight FP

- Comparison in Area/Delay/Power
  - 32-bit IEEE FP IDCT / 14-bit lightweight FP IDCT with Jamming rounding / 20-bit fixed point IDCT



Low-Resolution Display

- Media software commonly done in full precision (32–64 bits)
  - *Why do this if the display cannot handle it?*
  - On a portable video player:



Full precision decode

Display image on portable device

Encoded bitstream

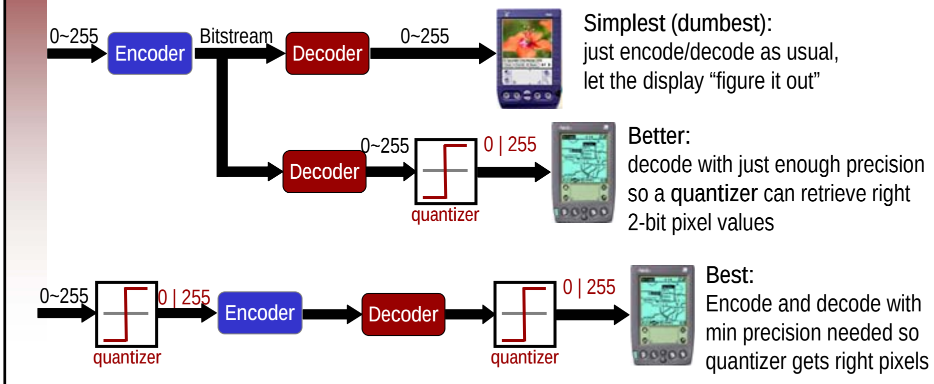
Decoded Image

Decoded Image on 2-bit B&W display

This is *really* inefficient

- Can't we do better than this, with smarter operators?

Low-Resolution Display (cont.)



Simplest (dumbest):
just encode/decode as usual,
let the display "figure it out"

Better:
decode with just enough precision so a **quantizer** can retrieve right 2-bit pixel values

Best:
Encode and decode with min precision needed so quantizer gets right pixels

- Results
  - Simplest: needs ~20-**bit** lightweight floats to work
  - Better: needs 16-**bit** lightweight floats; even just 11-**bits** looks decent
  - Best: needs just 9-**bit** floats (4 fraction bits) to work just fine.

Video Demo

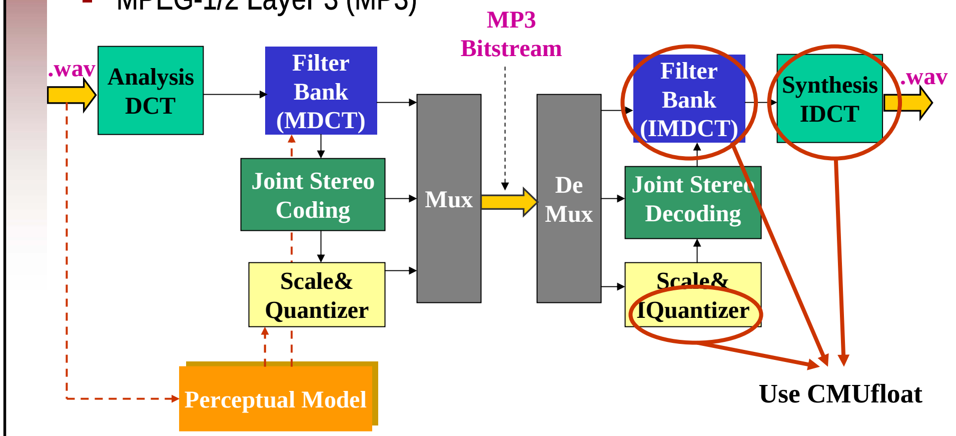
- Full Precision (64 bit)

- Using 23 bits (IEEE 1180 passed)

- Using 11 bits (IEEE 1180 failed)

How About Audio?

- MPEG-1/2 Layer 3 (MP3)



MP3 Bitstream

.wav

Analysis DCT

Filter Bank (MDCT)

Joint Stereo Coding

Scale& Quantizer

Perceptual Model

Mux

De Mux

Filter Bank (IMDCT)

Joint Stereo Decoding

Scale& IQuantizer

Synthesis IDCT

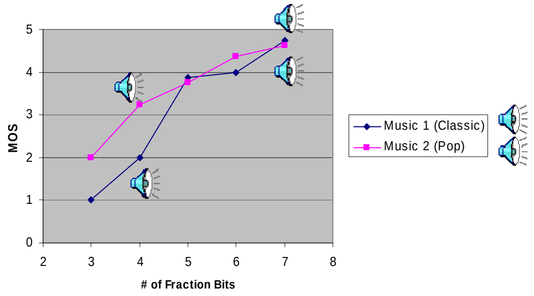
.wav

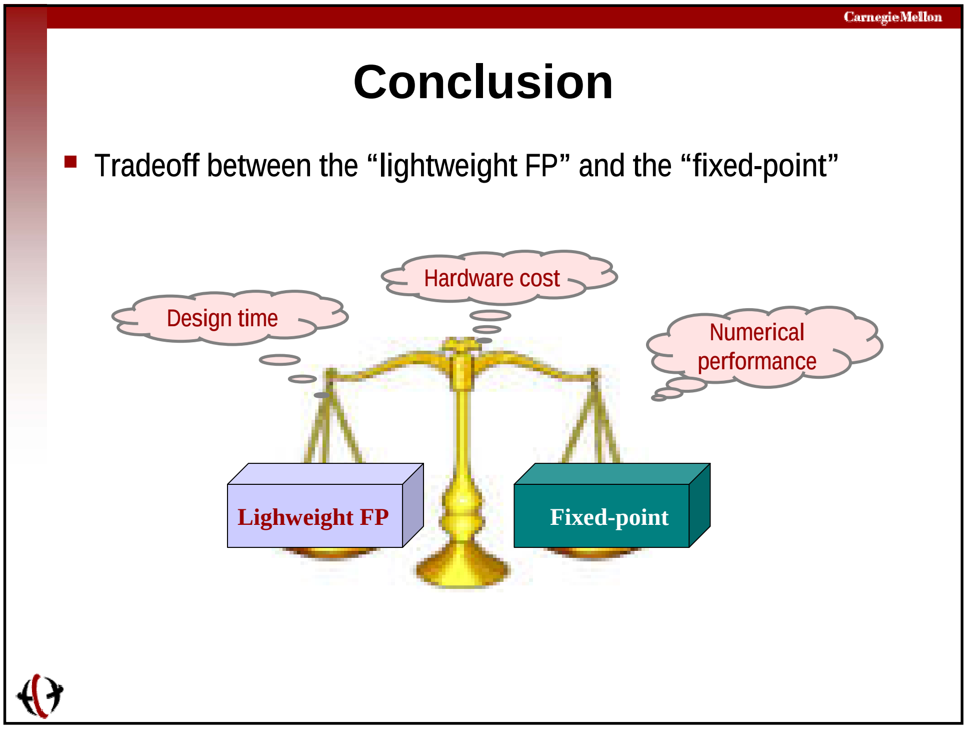
Use CMUfloat

- No standard tests for quality

Audio Quality

- Need to rely on subjective testing on perceptual quality
  - Mean Opinion Score (MOS)
    - From 5 “imperceptible difference” to 1 “really annoying”
- Results
  - 8 subjects. 6-bit exponent and 3~7 bit fraction





Conclusion

- Tradeoff between the “lightweight FP” and the “fixed-point”

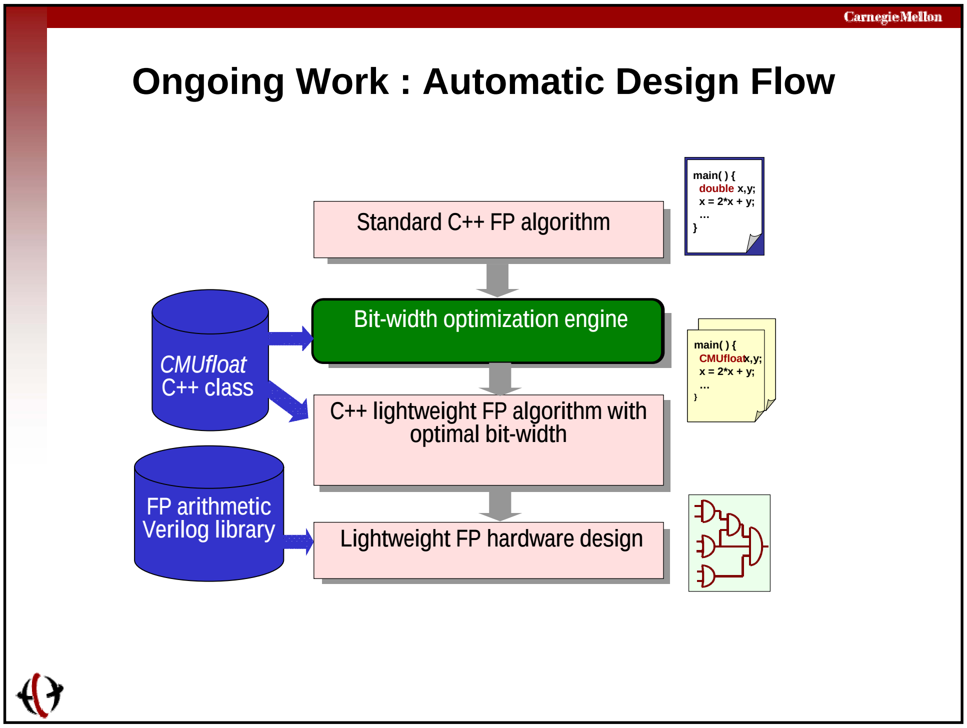
Hardware cost

Design time

Numerical performance

Lighweight FP

Fixed-point



Ongoing Work : Automatic Design Flow

Standard C++ FP algorithm

```
main( ) {
  double x,y;
  x = 2*x + y;
  ...
}
```

Bit-width optimization engine

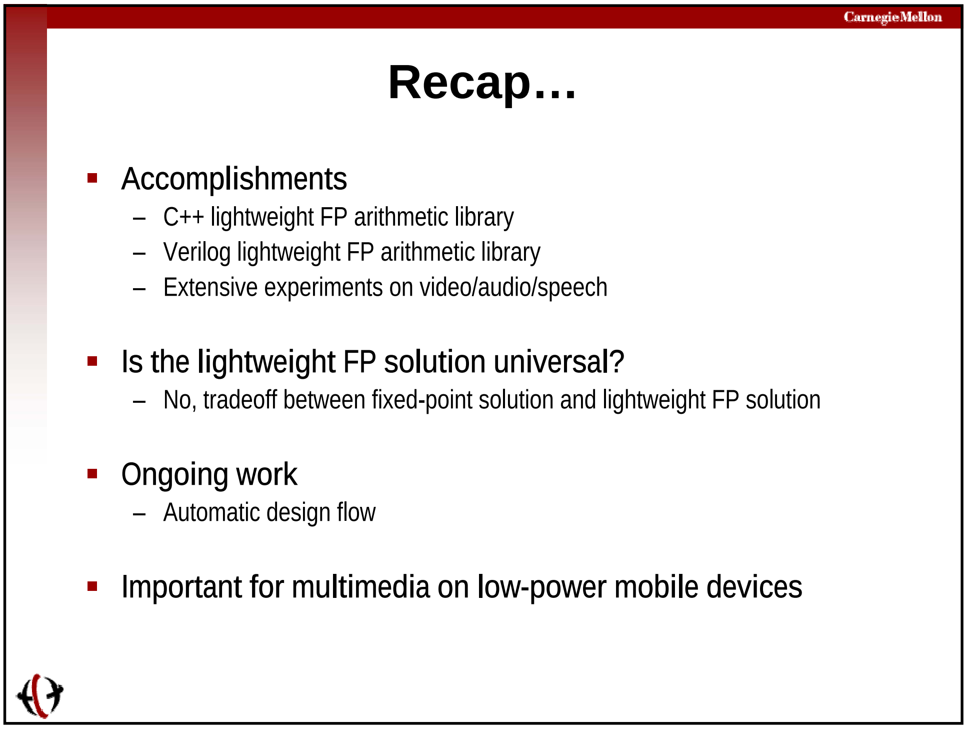
CMUfloat C++ class

C++ lightweight FP algorithm with optimal bit-width

```
main( ) {
  CMUfloatx,y;
  x = 2*x + y;
  ...
}
```

FP arithmetic Verilog library

Lightweight FP hardware design



Recap…

- Accomplishments
  - C++ lightweight FP arithmetic library
  - Verilog lightweight FP arithmetic library
  - Extensive experiments on video/audio/speech
- Is the lightweight FP solution universal?
  - No, tradeoff between fixed-point solution and lightweight FP solution
- Ongoing work
  - Automatic design flow
- Important for multimedia on low-power mobile devices

Advanced Multimedia Processing Lab

Please visit us at:

http://amp.ece.cmu.edu