RSSI 2008
Foundation of FPGA Acceleration

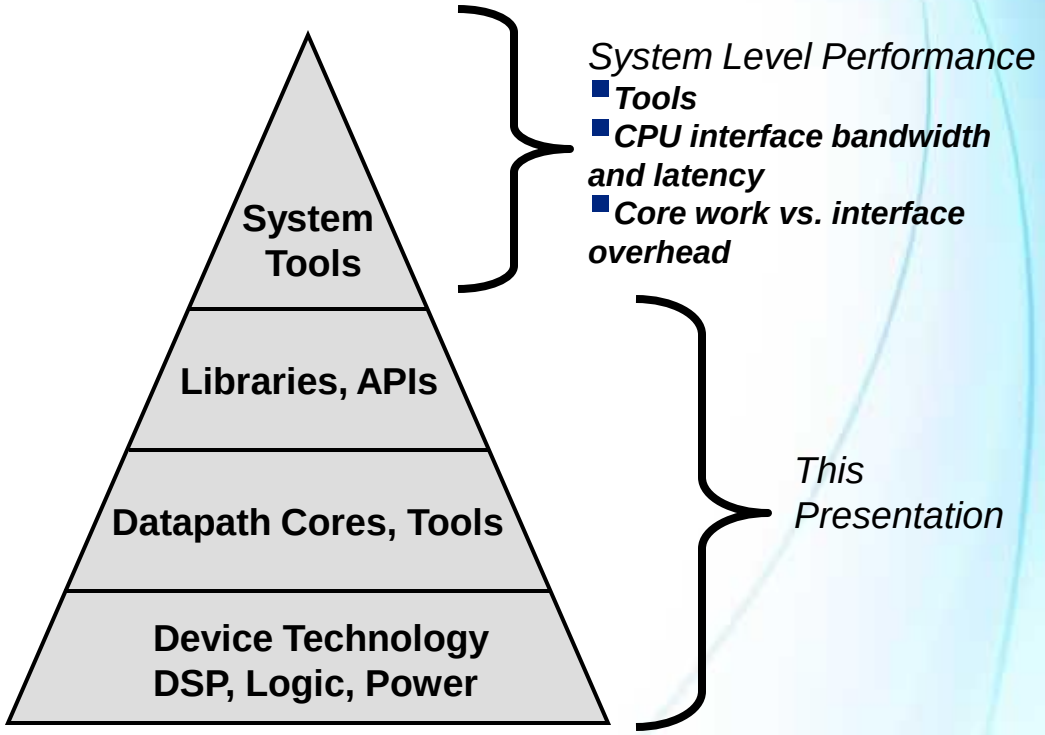
Martin Langhammer
Architect, Altera

Overview

- **Foundation of FPGA Acceleration**
  - FPGA Technology, Tools, and Libraries
- **FPGA Technology Advantage for Computing**
  - Bandwidth, Arithmetic Density, Power Density
- **The FPGA as a Computing Platform**
  - Fixed and Floating Point
  - Arithmetic Flexibility
- **Floating Point Datapath Compiler**
  - Turning the Fixed Point FPGA into a Floating Point Compute Engine
- **FPGA Computing Library Functions**
  - High Performance and High Density

Foundation for FPGA Acceleration

- **System Tool Chain**
  - Programmers can make daily changes
  - High Level Language
    - e.g. SRC CARTE
  - Functional simulation on x86
  - Hide FPGA synthesis complexity
  - Tool Chain optimizes for FPGA resources
    - e.g. no manual loop unrolling
- **Library Foundation**
  - Abstract the hardware
    - e.g. MKL for x86
  - Optimize for given device
    - cache, data reuse, etc.
- **Arithmetic Foundation**
  - Efficient operators and functions
    - Optimized for DSP blocks
  - Floating Point Compiler
- **Silicon Foundation**
  - DSP Density
  - Power Management



FPGA – Technology Advantage

Advantage - Bandwidth

External

- **Flexible, Multi-Standard I/O**
  - HSSI, QDR, SDRAM
- **Any external traffic can be routed internally**
- **Stratix ® EP2S180 – XDI Board Example**
  - **54 Single Precision Words/clock**
  - **27 Double Precision Words/clock**

Internal

- **Tri-Matrix ® Memory**
- **A signal can be routed to all places on the device**
  - **A signal can be routed to many multiple places on the device**
- **M144/M9K Bandwidth**
  - **11K Single Precision words/clock**
  - **5.5K Double Precision words/clock**

XtremeData XD2000F

22.4 GBps QDR
10.6 GBps DDR
10.6 GBps Host

43.6 GBps

Stratix ® EP3SL340 Memory Bandwidth

Memory	Number	Width	Ports	b/clock
M144	48	72	2	6912
M9K	1040	36	2	74880
MLAB	6750	20	2	270000

Advantage – DSP Performance

- **Floating Point density largely determined by hard multiplier density**
  - Multipliers must efficiently support floating point mantissa sizes
- **Fixed point DSP**
  - Stratix ® III : 896 18x18 Multipliers
  - Stratix ® IV : 1288 18x18 Multipliers
- **Single precision IEEE754**
  - Stratix ® III : 224 36x36 Multipliers
  - Stratix ® IV : 322 36x36 Multipliers
  - Extended Single Precision
- **Double precision IEEE754**
  - Stratix ® III : 112/89 54x54 Multipliers
  - Stratix ® IV : 161/128 54x54 Multipliers

Selected Devices Stratix ® III

Device	ALMs	18x18	36x36	54x54	Memory (Kbits)
EP3SE110	85,200	896	224	112/89	8055
EP3SL200	159,120	576	144	72/57	7668
EP3SE260	203,520	768	192	96/76	14688
EP3SL340	270,400	576	144	72/57	16272

Selected Devices Stratix ® IV

Device	ALMs	18x18	36x36	54x54	Memory (Kbits)
EP4SE230	182,400	1288	322	161/128	14283
EP4SE360	282,880	1040	260	130/104	18144

Advantage – Power Density

- **DGEMM and SGEMM examples (1:1 Multiplier:Adder Ratio)**
- **Naïve Functional Density and Power Analysis**
  - First order comparison
  - More complex operator and dataflow mix will change some results dramatically
  - FPGA numbers remain relatively constant

Technology	GFLOPS/ WATT	Sustained GFLOPS	Peak GFLOPS	Power	Precision
Altera 2S180	0.5	14.1†	25.2	25W	Double
Xeon 4 Cores	0.6	43.1*		70W	Double
Altera 3SE260 w/FPC	1.6	46.7††	48.6	30W	Double
(2) 3SE260 w/FPC **	1.6	93.4††	97.3	60W	Double
Altera 4SGX530	1.2	42.8†	67.3	35W	Double
Altera 4SE680	1.6	57.1†	90.0	35W	Double
(2) Altera 4SGX360 **	1.4	89.9†	137.3	65W	Double
nVidia C870	.5	70.0*	500	150W	Single
Xeon Quad E54XX	1.0	70.0*		70W	Single
Altera 3SE260 w/FPC	3.4	102.1††	136.0	30W	Single

Note *: HP published SGEMM and DGEMM benchmarks
Note **: In small 29x29mm package, two of these fit in current Xeon "keepout" e.g. XD2000i
Note †: 5% cpu i/f overhead, plus 33% FMAX and 15% area degradation for IEEE754; 1K x 1K matrices
Note ††: 5% cpu i/f overhead, plus 20% FMAX and 15% area degradation with Floating Point Compiler

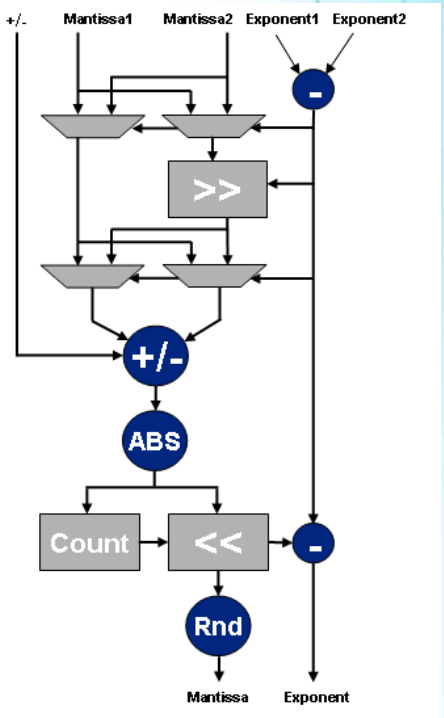
FPGA - Computing Platform

FPGA Computing

- **FPGAs are fixed point devices – with flexible internal extensions for floating point**
  - Floating point algorithm design and implementation is fundamentally different than using fixed point
- **FPGA Fixed Point system design**
  - Embedded monolithic multipliers
  - Direct soft logic support for adders
  - Dividers and square root are orders of magnitude more expensive
    - Both Area and Latency – therefore avoid whenever possible
  - MATH.H generally unavailable
- **FPGA Floating Point system design**
  - Write in C or other HLL
  - Specification and use of divides and square roots simple
    - Cost in performance hidden during modelling
    - System impact may be minimal if infrequently used
  - MATH.H easily available
- **FPGAs need to do more than just support floating point – must add value compared to other targets**
- **This does not mean embedded floating point units**
  - That would be an ASSP, not FPGA

Fixed Point

Floating Point



Multiplier Advantage = Core Advantage

- **New devices – 1000+ multipliers**
- **Embedded multipliers give consistent performance**
  - Routing and Speed
- **Multiplier/LUT ratio determines device capability**
- **Assume 1:1 Multiplier:ALU Ratio for Linear Algebra**
  - 20-40 LUTs for Fixed Point
  - 2000 LUTs for Double Precision
  - 750 LUTs for Single Precision
  - Floating Point is soft logic limited
- **New divider and MATH.H multiplier based**
  - System complexity pushed into embedded multipliers

Single Precision

Core	ALUT	DSP (18x18)	Latency
ALU	500	0	10
Multiplier	250	4	8
Divider	375	13	15
Inverse Root	325	11	19
EXP	525	11	16
LOG	1200	7	21

Double Precision

Core	ALUT	DSP (18x18)	Latency
ALU	1000	0	10-14
Multiplier	600	10	7
Divider	900	27	21
Inverse Root	900	27	32
EXP	1650	28	22
LOG	2050	12	26

Towards the C Programmable FPGA

- **This slide is not about HLL-FPGA compilers**
  - This technology already exists, and is being supported by FPGA vendors and third party tools providers
- **To take the lead in computing, FPGAs must support compromise-free algorithm to datapath mapping**
  - Linear trade-off between all operators and functions
  - Linear trade-off between precision
    - Single and Double Precision
  - Complete flexibility in inserting any function into any point of the datapath pipeline
- **Everything else has one or more compromises**
  - Non-linear function vs. operator cost
  - Fixed number of functions
  - Fixed number representation

Black-Scholes

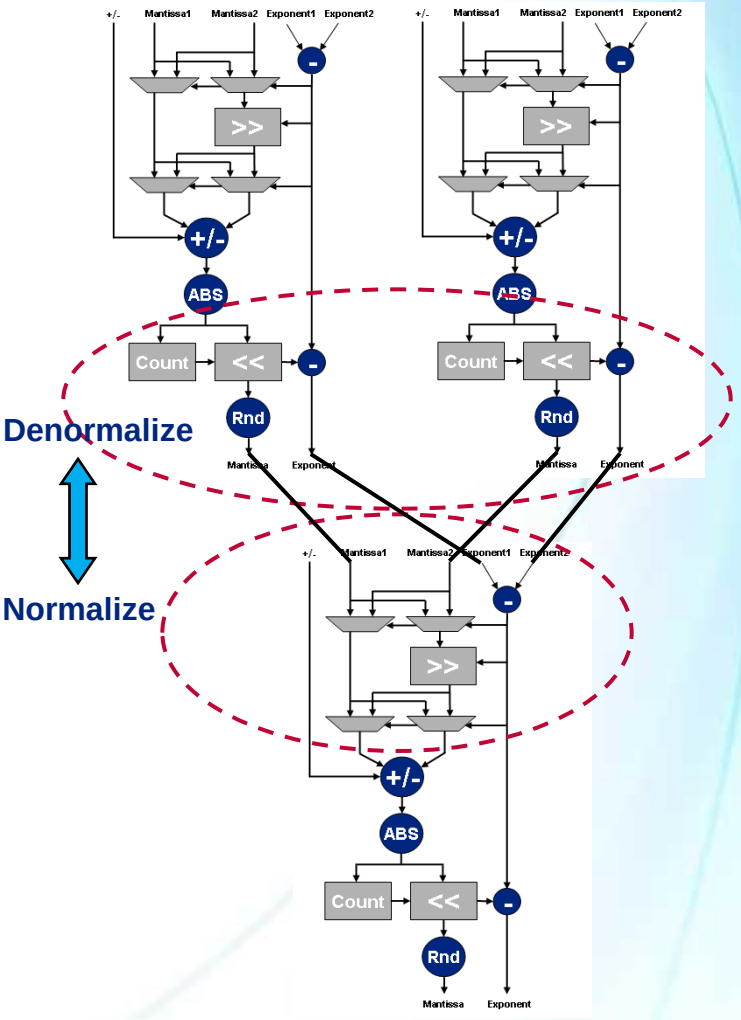
```
double rr, tt, vv;
double ss, xx;
double result;

void main()
{
  double one, two;

  one = rr * tt;
  two = vv * vv * tt;
  result = (log(ss/xx) + one +
  ldexp(two,1))*invsqr(two);
}
```

Multiple instances on an FPGA
100% multiplier usage
1 result/clock/core
No pipeline stall

Floating Point Compiler

Floating Point Datapath Compiler Need

- **Conventional Wisdom : IEEE754 system level design too complex for FPGAs**
  - Floating Point core based design requires more soft logic than a fixed point FPGA supports
- **But IEEE754 datapath inefficient - functional redundancy between operators**
  - Required for processors – unlimited operation combinations
    - Arithmetic unit requires all inputs and outputs to be of a known format
  - Not required for datapath – *a priori* knowledge of inter-operator relationships
    - Datapath unit requires all inputs and outputs to be IEEE754 format
    - Internal format only has to guarantee that casting to and from IEEE754 is correct
  - Investigate possibility of a fused datapath
- **Attempt to use fixed point FPGA strengths for equivalent floating point implementation**



Floating Point Compiler

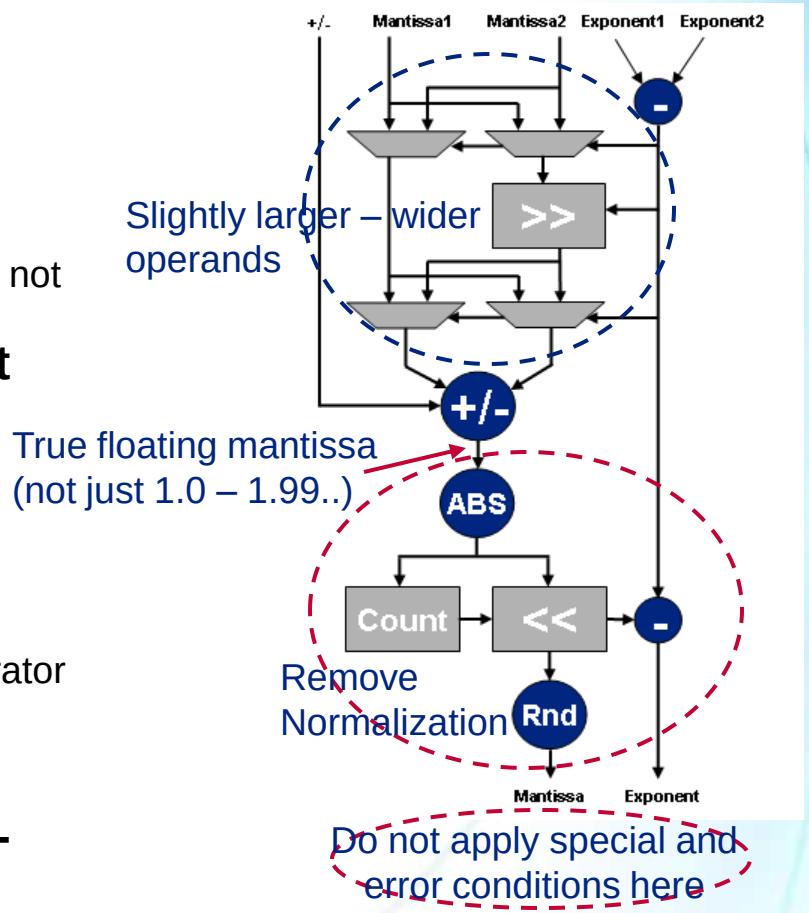
- **C to Datapath**
  - Abstracts implementation
  - Datapath optimizations would be difficult to direct in HDL environment – higher level of abstraction needed
  - Limited to simple expressions – many other tool flows for control and memory management
- **Fuses together entire datapath**
  - Typical 50% soft logic reduction
  - Typical 50% latency reduction
  - Power reduction : most of the dynamic power consumption in the datapath will be soft logic, not multipliers
- **Area and latency reduction allows 100% of device floating point capability to be used**
  - Large, multiplier-rich Stratix devices can be filled with 100's of floating point operators
  - Depending on algorithm, only 30-60% soft logic is used
    - Application wrapper, interface, routing headroom and other functions easily accommodated

Radix 4 DFT C Description

```
double realina, imagina;
double realinb, imaginb;
double realinc, imaginc;
double realind, imagind;

double realtwid, imagtwid;

double realout, imagout;

void main()
{

  double realdft, imagdft;

  realdft = (realina + realinb) +
            (realinc + realind);
  imagdft = (imagina + imaginb) +
            (imaginc + imagind);

  realout = (realdft * realtwid) –
            (imagdft * imagtwid);
  imagout = (realdft * imagtwid) +
            (imagdft * realtwid);

}
```

Compiler - Operation

- **Define new floating point format**
  - More precision in mantissa, exponent
  - More dynamic range in mantissa, exponent
  - Floating mantissa – no implied range
- **Denormalize – minor impact**
  - Fixed point conversion relative to exponents, not mantissa
- **Fixed point operation - minor impact**
  - Larger adder, few ALUTs
  - Larger multiplier, 36x36 native DSP Block support
    - Double precision case more complex
- **Normalize – remove**
  - Relative value denormalization creates valid floating point mantissa out of fixed point operator
  - Insert normalization operator where needed
    - Overflow or underflow (loss of precision case) possible
- **Error and special condition section - remove**
  - Detect only – forward condition to next node



FPGA - Library Functions

Library Functions

Partner Libraries

SRC, XDI, Mitrionics, Impulse-C

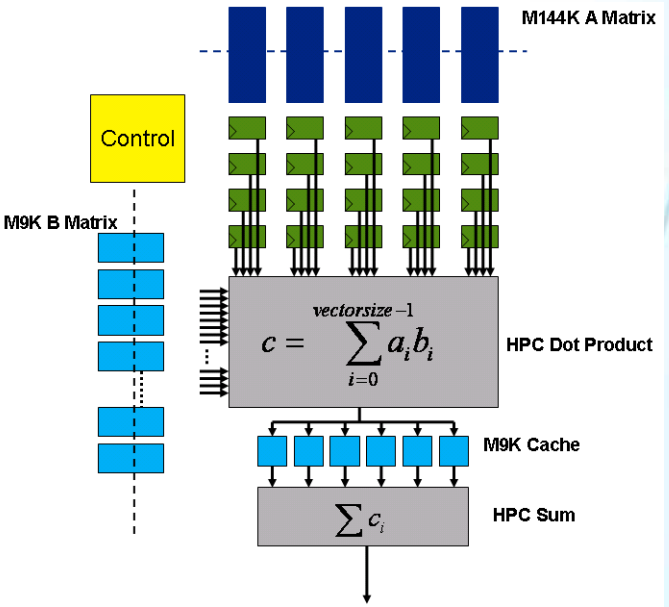
- **Image and Signal Processing**
  - 1D and 2D FFT
  - DCT, IDCT
  - Image filtering, Image matching, Image fusion, Image enhancement
  - Histogram, Global image analysis,
  - Cross-correlation, Color space transformations
  - Filtered backprojection
  - Wavelet compression
  - Sobel convolution
- **Encryption**
- **Application/Algorithm Examples**
  - Monte Carlo Black-Scholes
  - BLASTn, BLASTp

Altera Libraries

- **xGEMM**
  - SGEMM
  - ZGEMM
  - DGEMM
- **FFT (Double Precision)**
  - Streaming (Up to 16K)
  - Block (256K-1M pt)
  - Single Precision FFT
    - Altera Megacore (designed with Floating Point Compiler)
- **LU Decomposition**
- **Cholesky Decomposition**
- **Black-Scholes**

Altera xGEMM

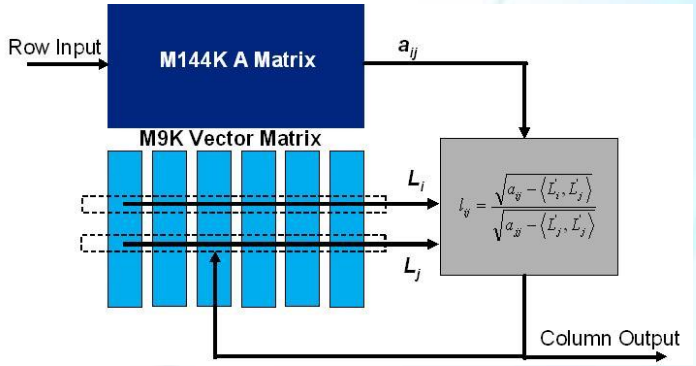
- Compiler technology gives a linear tradeoff between cost and specification
  - Real vs. Complex
  - Single vs. Double
- Linear algebra good example of compiler capability
  - 100% fixed point multiplier usage in a single datapath – but for floating point application
  - 35% device soft logic use
  - Interface, application wrapper, other functions easily accommodated
- High performance
  - 200+ MHz double precision
    - ~50 GFLOPs – 3SE260
  - 250+ MHz single precision
    - ~100 GFLOPs – 3SE260



Altera Cholesky Matrix Decomposition

- **Single architecture for both real and complex numbers**
- **Parameterized VHDL top level file supports wide range of parameters**
  - Example: Stratix ® EP4SE230
  - 1288 Multipliers = 300 real, 75 complex element vectors
    - Some multipliers needed for square root operators
  - To change matrix sizes,
    - Change vector operator description in C
    - Use Floating Point Compiler
    - Create new Component, add to core
- **Fused datapath design more accurate than IEEE754 model**
  - Divides, Square Roots supported in 32+ bits
  - Cluster normalization keeps information during wordgrowth

```
float L00, L01, L02, L03, L04, L05, L06, L07;
float L08, L09, L0a, L0b, L0c, L0d, L0e, L0f;
float L10, L11, L12, L13, L14, L15, L16, L17;
float L18, L19, L1a, L1b, L1c, L1d, L1e, L1f;

float M00, M01, M02, M03, M04, M05, M06, M07;
float M08, M09, M0a, M0b, M0c, M0d, M0e, M0f;
float M10, M11, M12, M13, M14, M15, M16, M17;
float M18, M19, M1a, M1b, M1c, M1d, M1e, M1f;

float aa;
float result;

void main()
{

 float midone, midtwo, midthr, midfor;
 float vector;

 midone = (((L00*M00) + (L01*M01)) + ((L02*M02) + (L03*M03))) +
        (((L04*M04) + (L05*M05)) + ((L06*M06) + (L07*M07)));
 midtwo = (((L08*M08) + (L09*M09)) + ((L0a*M0a) + (L0b*M0b))) +
        (((L0c*M0c) + (L0d*M0d)) + ((L0e*M0e) + (L0f*M0f)));
 midthr = (((L10*M10) + (L11*M11)) + ((L12*M12) + (L13*M13))) +
        (((L14*M14) + (L15*M15)) + ((L16*M16) + (L17*M17)));
 midfor = (((L18*M18) + (L19*M19)) + ((L1a*M1a) + (L1b*M1b))) +
        (((L1c*M1c) + (L1d*M1d)) + ((L1e*M1e) + (L1f*M1f)));

 vector = (midone + midtwo) + (midthr + midfor);

 result = aa - vector;

}
```



Summary and Conclusions

FPGA Acceleration Ramping Up
Built Upon a Solid Foundation

Phase:	Appliance*	Function Acceleration	Algorithm Acceleration
Applications:	Database, Financial Market, Bioscience	Financial RNG, Military	Financial Options, Scientific
Tool Chain: Standards	N/A	API Library	Common Front-End Compiler Definition

Summary

More than naïve GFLOP comparison

- xGEMM used for baseline performance number
- Same performance with different operator mix

Linear trade-off for traditionally non-linear complexity

- Functions typically 3x cost of multiplier
- Same system performance and effect as multiplier
- Latency can usually be hidden by FP Compiler

Peak performance = sustained performance

- Compromise free algorithm to datapath implementation
- Changing operator mix will not change performance
  - 1, 5, 10 MATH.H functions will not break the datapath
- Changing dataflow mix will not change performance
  - xGEMM, FFT, Black-Scholes : unlimited flexibility in routing will ensure every operator busy every clock

Conclusions

- **Review: the foundations of FPGA acceleration**
  - Device Technology
  - Arithmetic Capability
  - Library Development
- **Flexibility is the key to FPGA Accelerator Performance**
  - All functional units guaranteed data on every clock
  - Linear trade-off on functional complexity
  - Linear trade-off on precision and representation
- **Datapath tools make fixed point FPGA a floating point FPGA**
  - Fixed Point architecture offers FPGA economies of scale – as well as power density, DSP density and memory density advantages

Backup