NVIDIA’s Pascal Architecture

Gabriel Naghi
CIS565 Project 4

Outline

Outline

Nvidia GPGPUs: G80 through Pascal

“The 5 Technological Breakthroughs of Pascal”

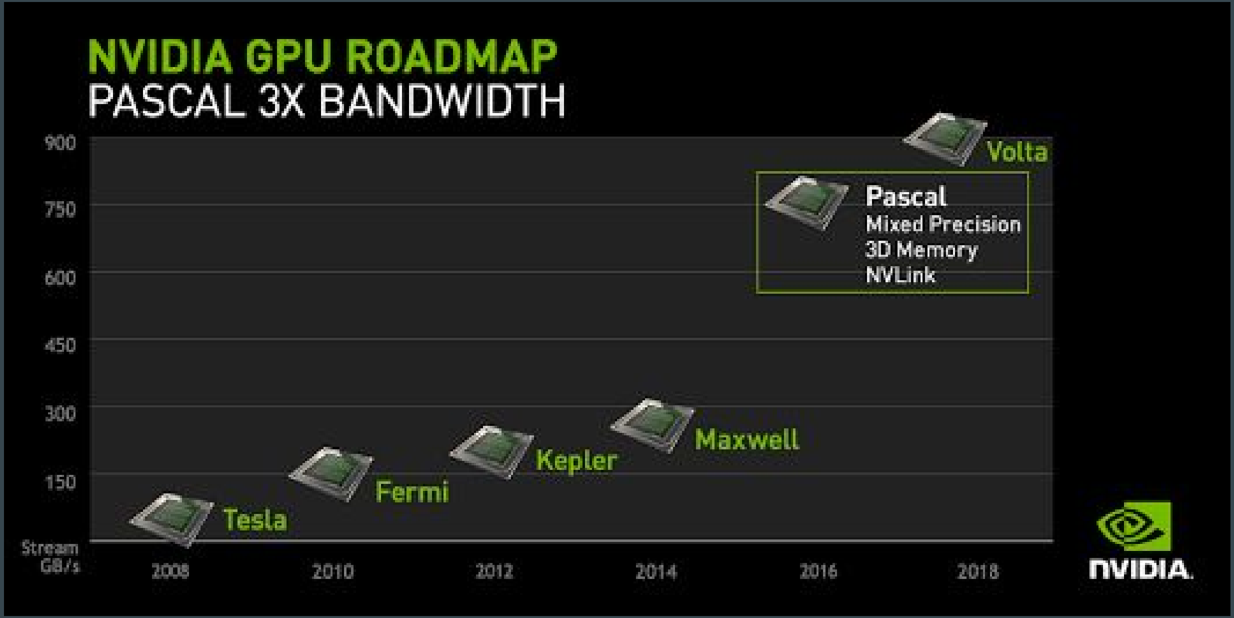
- Modules and Layout
  - Floating Point
- 16 nm FinFET
- NVIDIA NVLink
- CoWoS with HBM2
- Unified Memory and Compute Preemption
  - NUMA

GTX 1080 Summary

History of NVIDIA Architectures

Past NVIDIA Architectures

- Tesla
  - 576 GFLOPS
  - 681M transistors (90 nm)
  - 768 MB Memory
- Fermi
  - 1.5 TFLOPS
  - 3B transistors (40 nm)
  - 1.5 GB Memory
- Kepler
  - 3.1 TFLOPS
  - 3.5B transistors (28 nm)
  - 2 GB Memory
- Maxwell
  - 4.6 TFLOPS
  - 5.2B transistors (28 nm)
  - 4 GB Memory



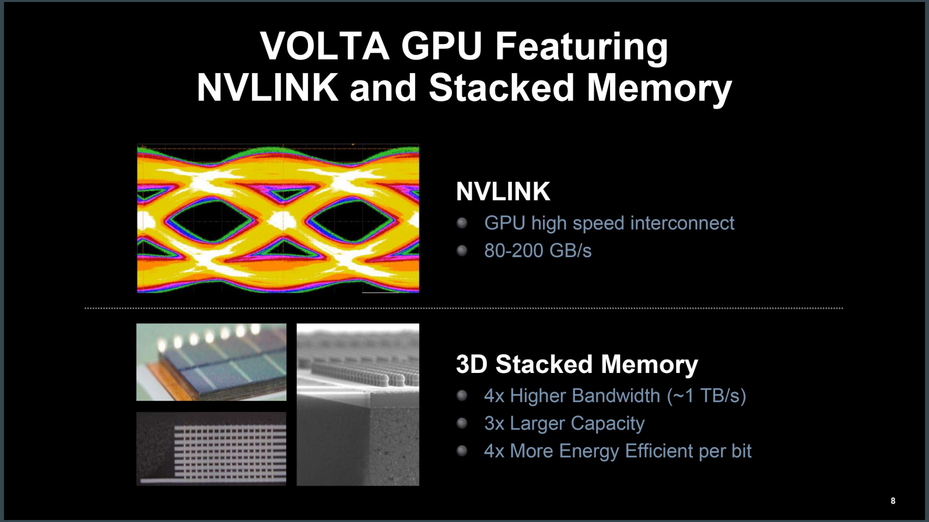
New Architectures

Pascal

- 15 TFLOPS
- 15B transistors (16 nm)
- 16 GB Memory

Volta

- ??? (Pascal stole all its roadmapped features)
- Focus on DX12
- Better Async execution
- Think: How is AMD still winning?



VOLTA GPU Featuring
NVLINK and Stacked Memory

NVLINK
- GPU high speed interconnect
- 80-200 GB/s

3D Stacked Memory
- 4x Higher Bandwidth (~1 TB/s)
- 3x Larger Capacity
- 4x More Energy Efficient per bit

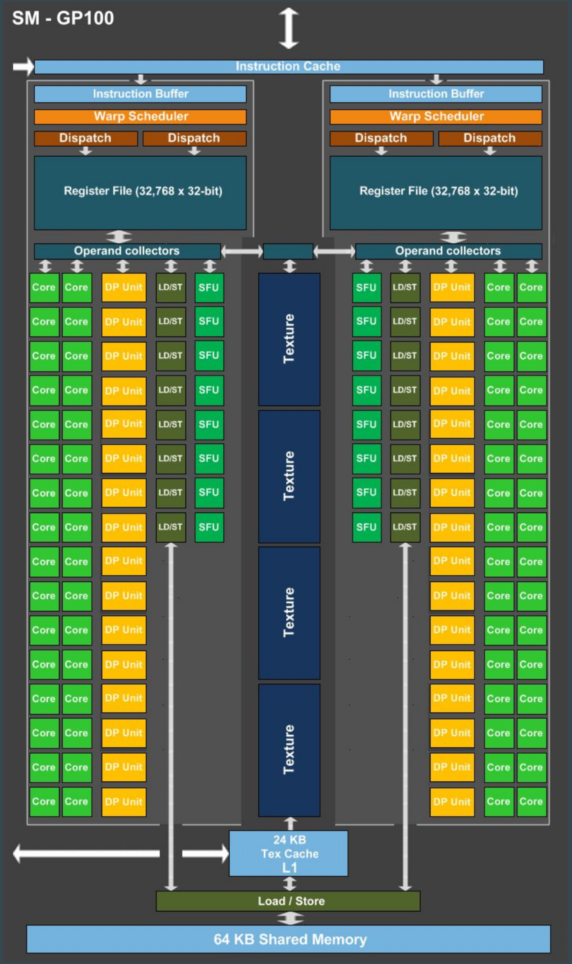
8

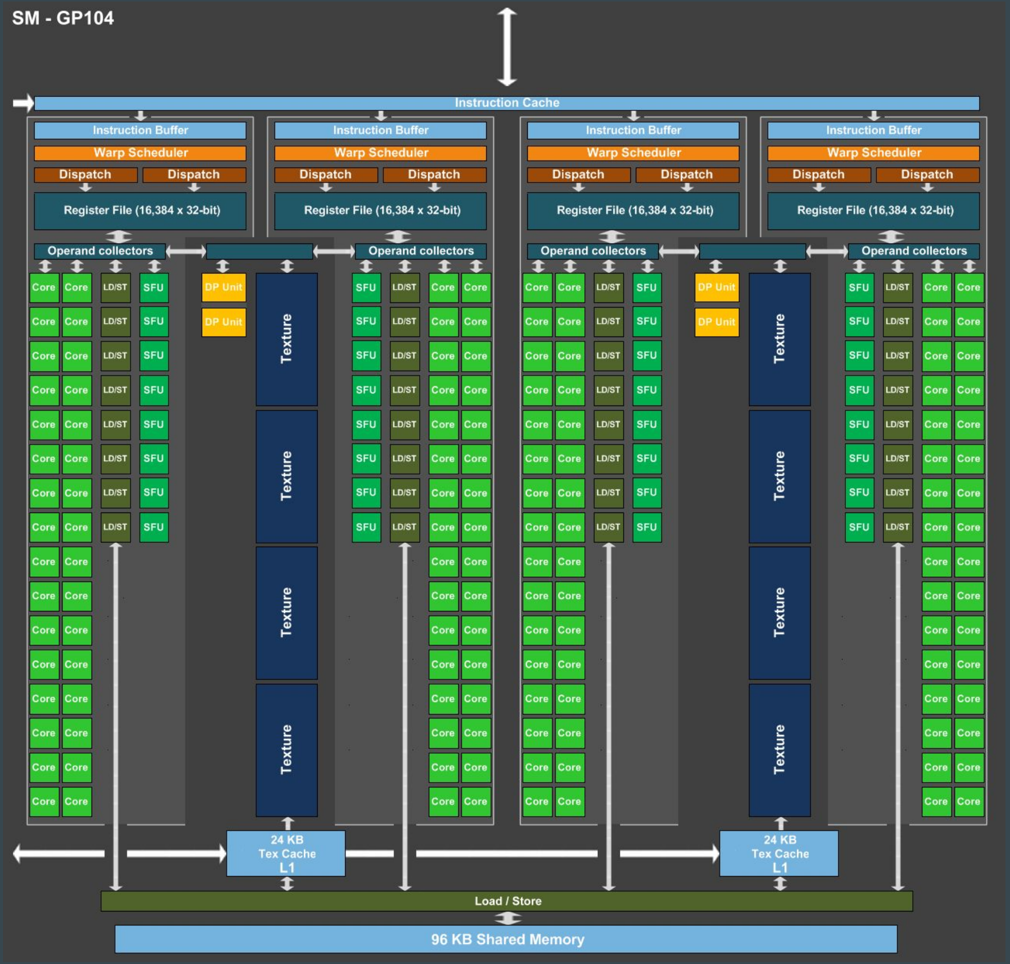
Source: NVIDIA

Pascal Architecture

Quick note on Title

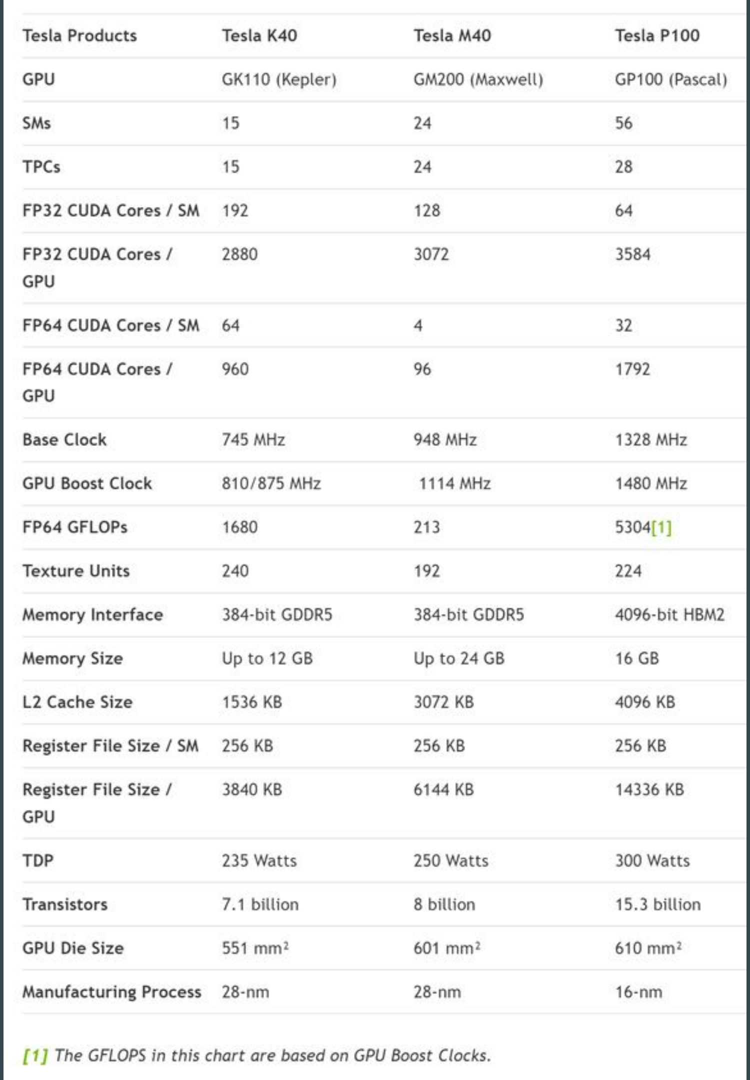
- NVIDIA touts these features as “Pascal Architecture”, but really will only be included in Tesla P100.
  - Uses GP100 GPU die (**Focus of this presentation**)
  - P100 meant for compute acceleration (HPC), not gaming
  - CRUNCH DATA
- First Pascal GPUs (GTX 10xx) have very few of these features
  - Based on GP104 die
  - 16 nm FinFET process!
  - GDDR5X vs HBM2
  - Different module organization (next slide)
  - Includes some additional graphics modules
    - Huge focus on VR
    - Simultaneous multi projection, lossless compression engine
  - MOVE FRAMES





Source: hardware.fr

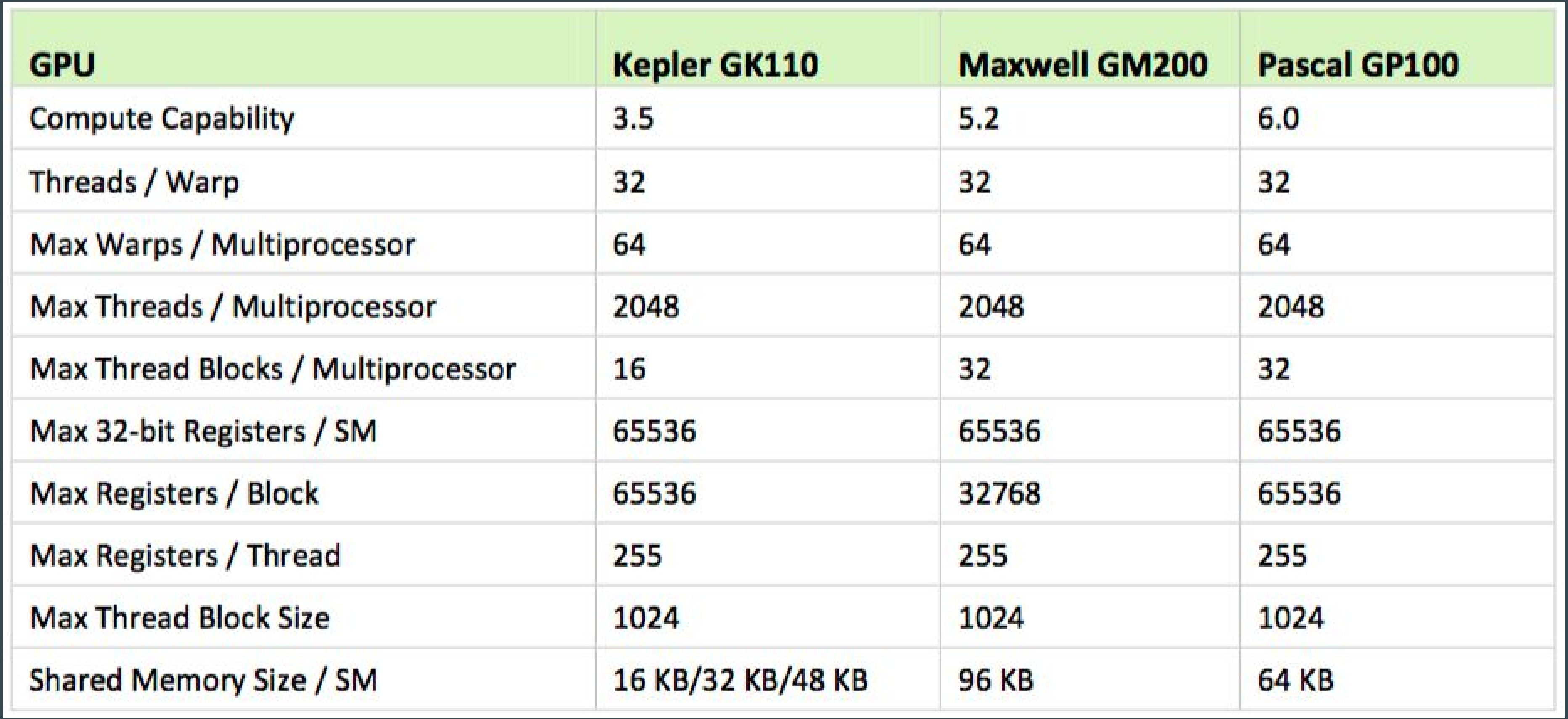
SPECS
(Not Architecture)



Tesla Products	Tesla K40	Tesla M40	Tesla P100
GPU	GK110 (Kepler)	GM200 (Maxwell)	GP100 (Pascal)
SMs	15	24	56
TPCs	15	24	28
FP32 CUDA Cores / SM	192	128	64
FP32 CUDA Cores / GPU	2880	3072	3584
FP64 CUDA Cores / SM	64	4	32
FP64 CUDA Cores / GPU	960	96	1792
Base Clock	745 MHz	948 MHz	1328 MHz
GPU Boost Clock	810/875 MHz	1114 MHz	1480 MHz
FP64 GFLOPs	1680	213	5304[1]
Texture Units	240	192	224
Memory Interface	384-bit GDDR5	384-bit GDDR5	4096-bit HBM2
Memory Size	Up to 12 GB	Up to 24 GB	16 GB
L2 Cache Size	1536 KB	3072 KB	4096 KB
Register File Size / SM	256 KB	256 KB	256 KB
Register File Size / GPU	3840 KB	6144 KB	14336 KB
TDP	235 Watts	250 Watts	300 Watts
Transistors	7.1 billion	8 billion	15.3 billion
GPU Die Size	551 mm²	601 mm²	610 mm²
Manufacturing Process	28-nm	28-nm	16-nm

[1] The GFLOPS in this chart are based on GPU Boost Clocks.

Source: NVIDIA



GPU	Kepler GK110	Maxwell GM200	Pascal GP100
Compute Capability	3.5	5.2	6.0
Threads / Warp	32	32	32
Max Warps / Multiprocessor	64	64	64
Max Threads / Multiprocessor	2048	2048	2048
Max Thread Blocks / Multiprocessor	16	32	32
Max 32-bit Registers / SM	65536	65536	65536
Max Registers / Block	65536	32768	65536
Max Registers / Thread	255	255	255
Max Thread Block Size	1024	1024	1024
Shared Memory Size / SM	16 KB/32 KB/48 KB	96 KB	64 KB

Source: NVIDIA

Modules and Layout

Die Hierarchy

GPU

⇩

Graphics Processing Cluster (GPC)

⇩

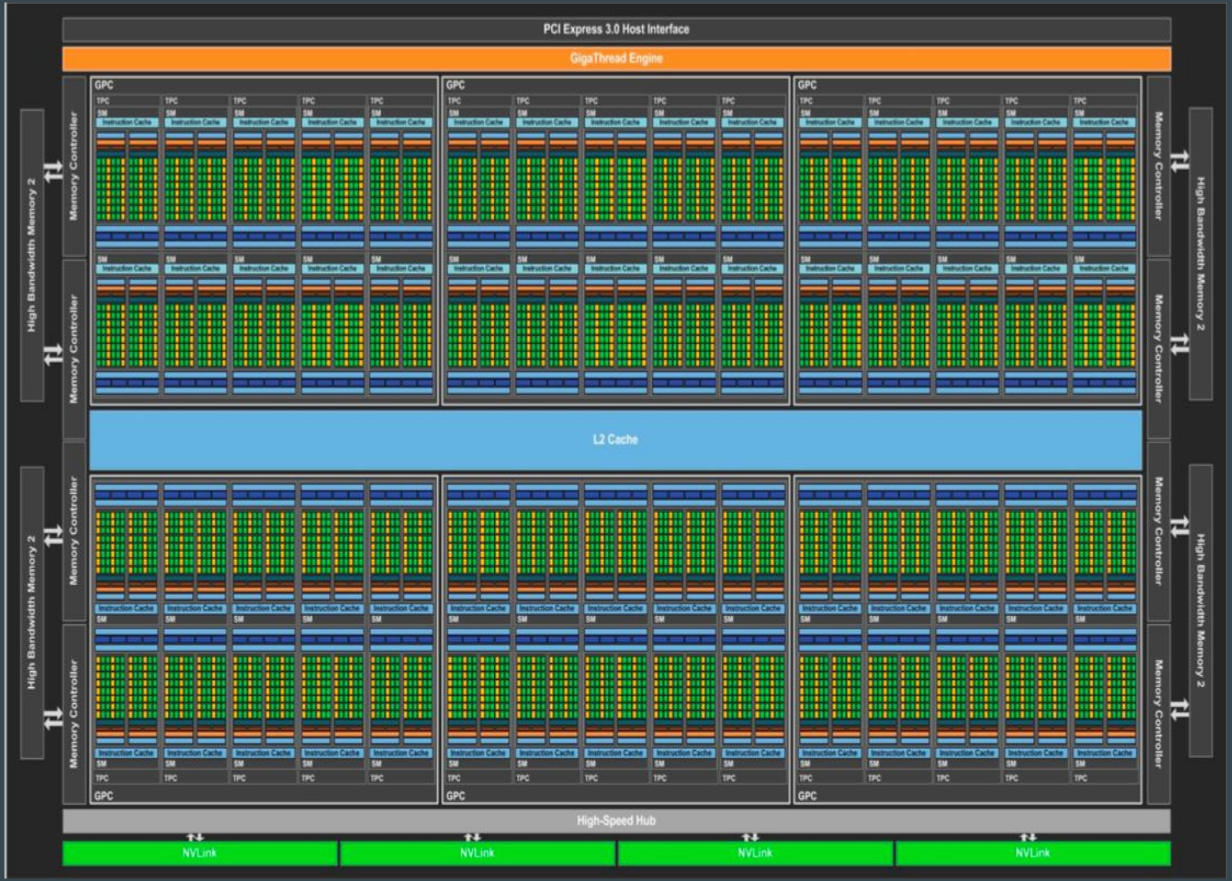
Texture Processing Cluster (TPC)

⇩

Streaming Multiprocessor (SM)

⇩

Execution Cores (Floating Point, Special Function)



Source: NVIDIA

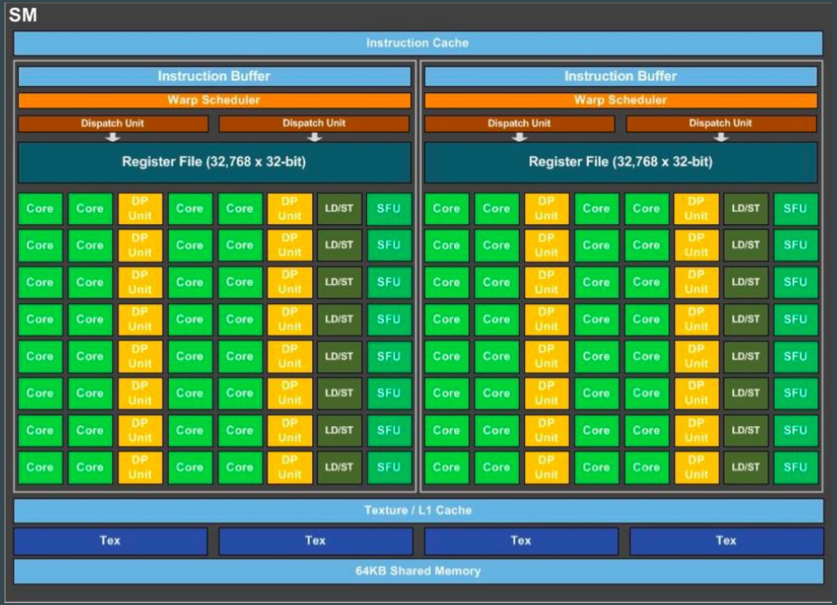
Pascal Streaming Multiprocessor (SM)

- Each SM partitioned into 2 execution blocks. Each has:
  - 32 32-bit FP units (64 total)
  - 16 64- bit FP units (32 total)
  - Instruction Buffer
  - Warp scheduler (2 instructions per clock)
  - Two dispatch units
  - 128 KB Register File, or ~32,000 4 byte words

2 SMs per TPC

5 TPCs per GPC

6 GPCs per GPU



Source: NVIDIA

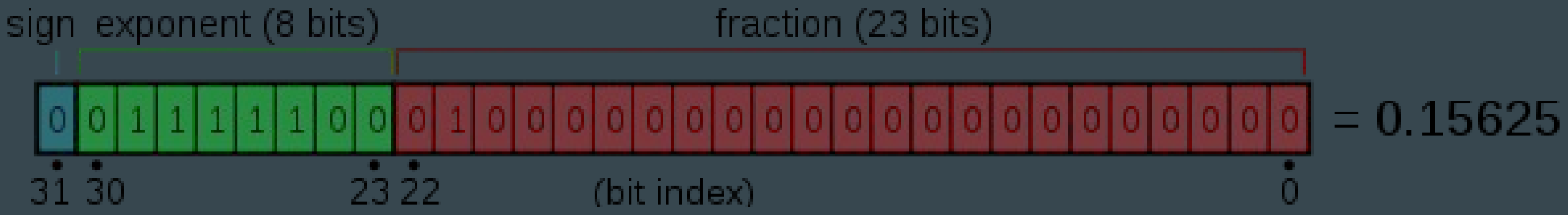
Strategic Optimizations

- More than double SM count over Maxwell
  - More threads, warps, and blocks in flight
- Higher ratio of shared memory, registers, and warps per SM
  - More warps for scheduler to choose from
- Lower ratio of FP32 units to FP64 units
  - Better workload balancing
  - Better performance for FP64 workloads
- Simpler datapath
  - Power, area efficiency
  - At the expense of bandwidth efficiency, probably
    - “Aggregate bandwidth effectively more than doubled” - despite >> 2x SMs
- optimized scheduling and overlapped load/store instructions
  - Better floating point utilization

32-bit Floating Point

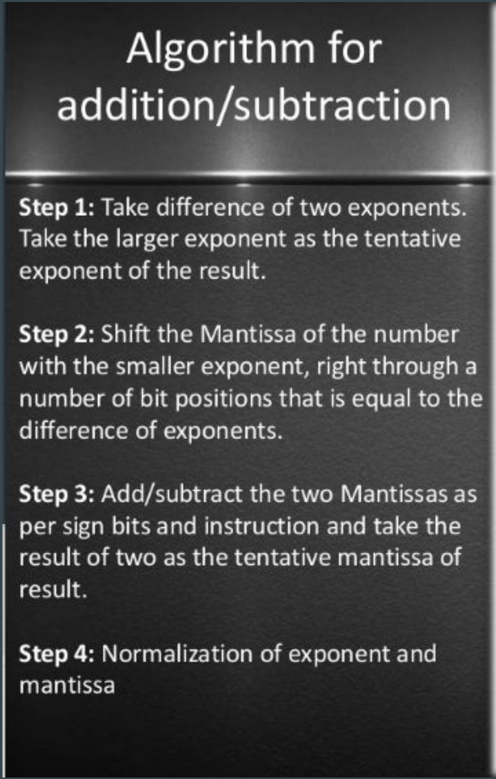
- IEEE 754 Single Precision Binary Floating Point Format
  - A.k.a. “Float” in c/c++
  - As opposed to Double-precision (FP64) or half-precision (FP16)
- 32 bit wide float B
  - B[31] (MSB) - 1 bit *sign*
  - B[30 : 23] - 8 bit *exponent*
  - B[22 : 0] - 23 bit *fraction*

$$\text{Value} = (-1)^{sign} * 1.fraction * 2^{exponent - 127}$$



Source: wikipedia

Floating Point Arithmetic



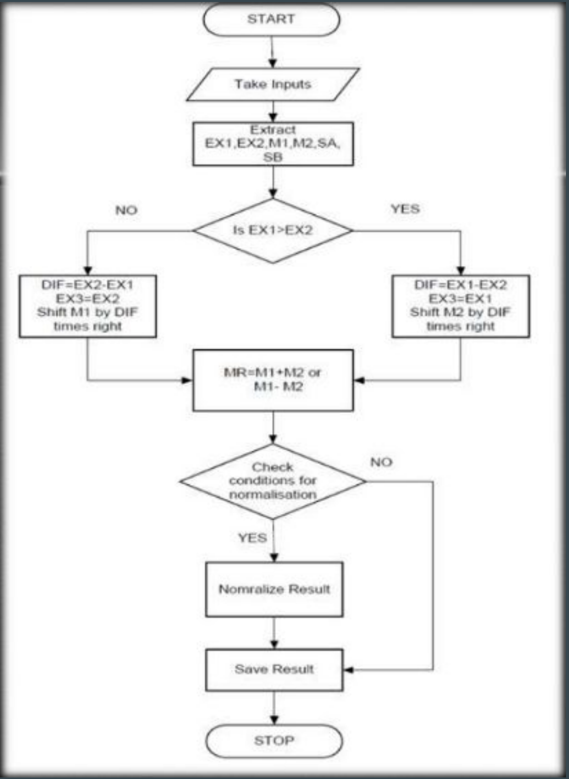
Algorithm for addition/subtraction

Step 1: Take difference of two exponents. Take the larger exponent as the tentative exponent of the result.

Step 2: Shift the Mantissa of the number with the smaller exponent, right through a number of bit positions that is equal to the difference of exponents.

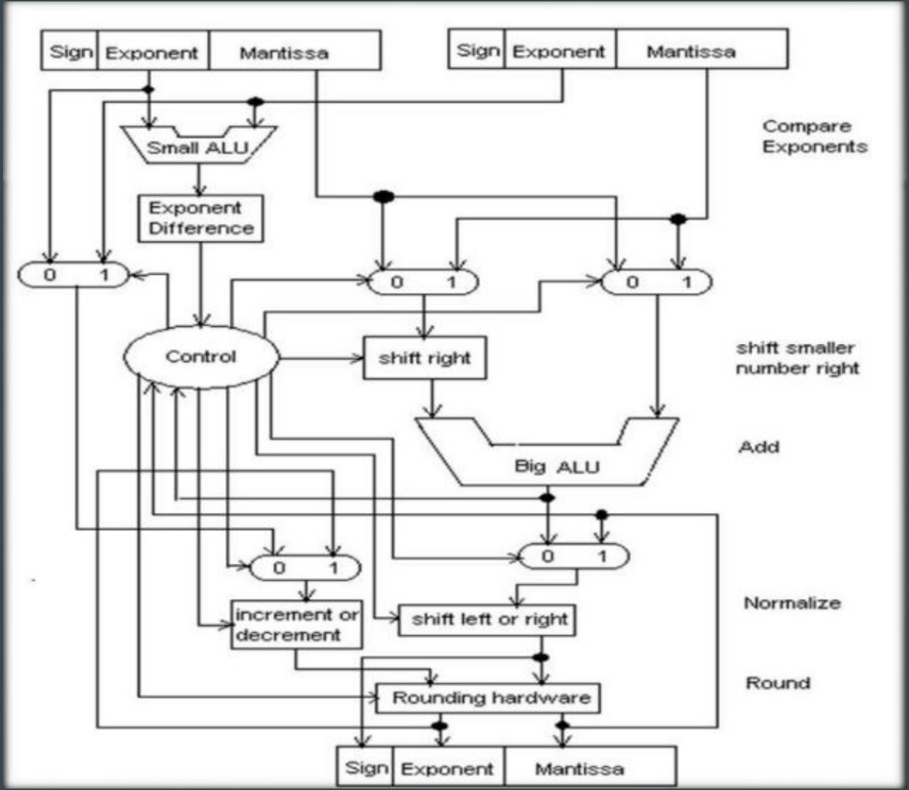
Step 3: Add/subtract the two Mantissas as per sign bits and instruction and take the result of two as the tentative mantissa of result.

Step 4: Normalization of exponent and mantissa


START
Take Inputs
Extract EX1,EX2,M1,M2,SA, SB
NO
Is EX1>EX2
YES
DIF=EX2-EX1 EX3=EX2 Shift M1 by DIF times right
DIF=EX1-EX2 EX3=EX1 Shift M2 by DIF times right
MR=M1+M2 or M1- M2
Check conditions for normalisation
NO
YES
Nomralize Result
Save Result
STOP

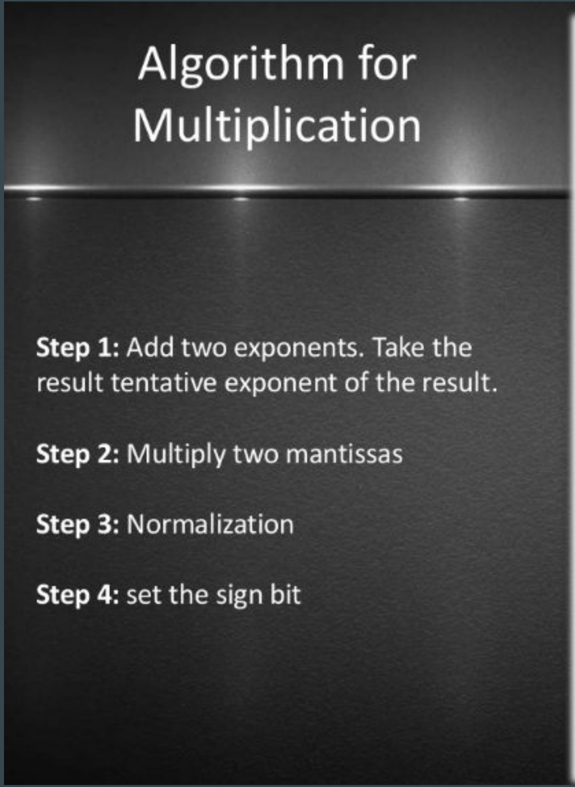
source : http://www.slideshare.net/azharsyed898/floating-point-alu-using-vhdl

Floating Point Arithmetic



source : http://www.slideshare.net/azharsyed898/floating-point-alu-using-vhdl

Floating Point Arithmetic



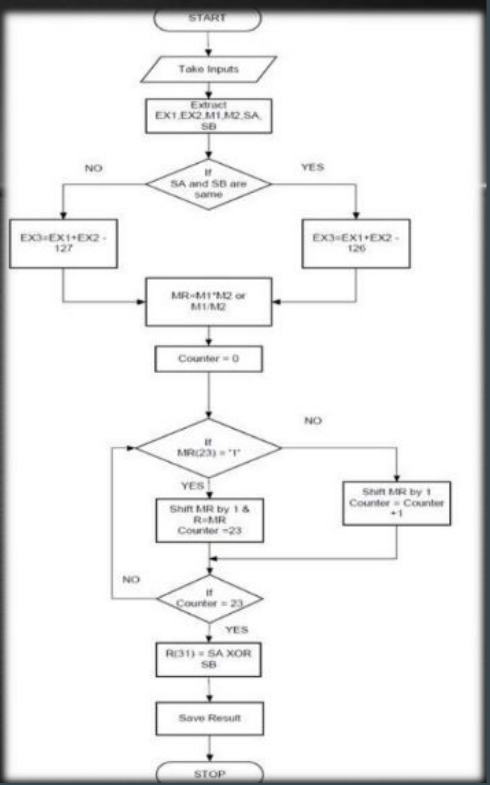
Algorithm for Multiplication

Step 1: Add two exponents. Take the result tentative exponent of the result.

Step 2: Multiply two mantissas

Step 3: Normalization

Step 4: set the sign bit

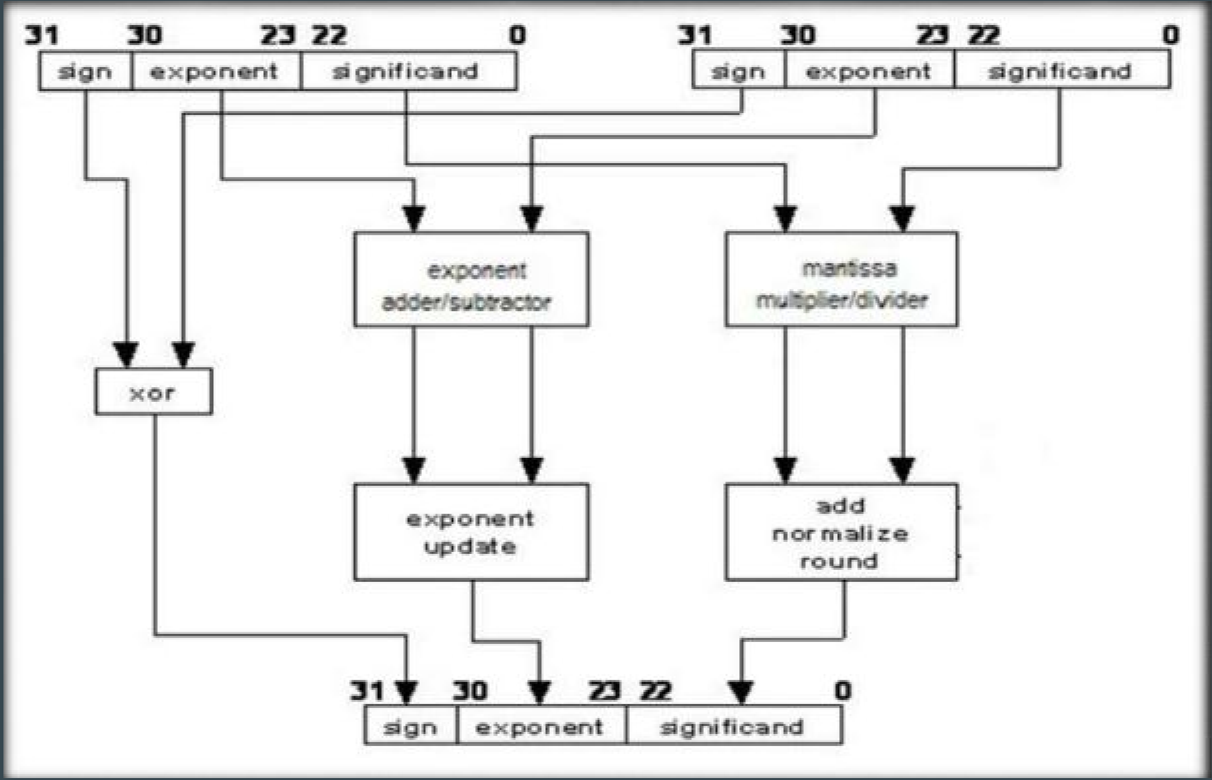


source : http://www.slideshare.net/azharsyed898/floating-point-alu-using-vhdl

Algorithm for Floating Point Division

1. Sign bit = S1 xor S2
2. Fraction = Fraction1 / Fraction2
3. Exponent = Exponent1 - Exponent2 + bias (in FP32, -127)
4. Normalize if required by left-shifting Fraction and decrementing Exponent

Floating Point Arithmetic

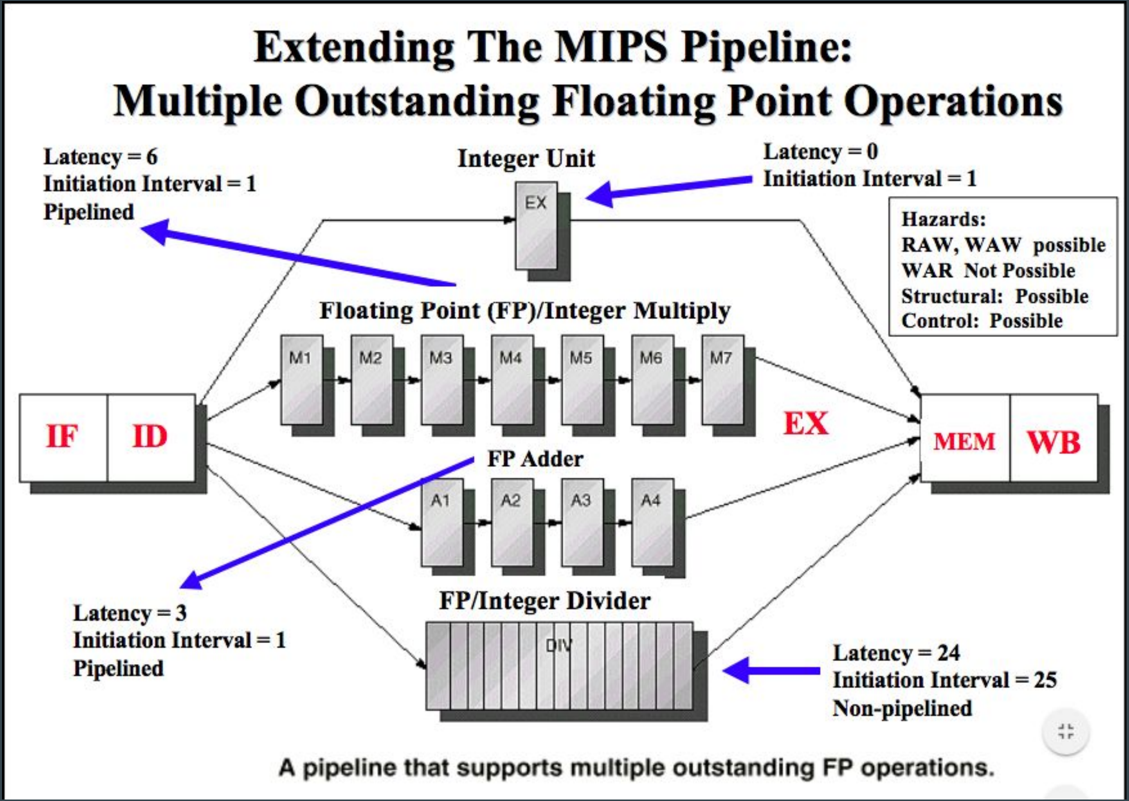


source : http://www.slideshare.net/azharsyed898/floating-point-alu-using-vhdl

Floating Point Hardware

- Involves *complicated* and *computationally expensive* operations
  - Especially divides!
- Often Pipelined to allow higher throughput
  - Latency same (or higher), but trade off computation per *fast* cycle rather
- Computation theoretically linear in operand length
  - In practice, super-linear
    - Caching
    - Fewer limitations imposed by shared memory, registers and fast memory
    - Don’t hit the bandwidth ceiling nearly as fast...

Pipelined Arithmetic Architecture Example (MIPS)



A pipeline that supports multiple outstanding FP operations.

source : http://meseec.ce.rit.edu/eecc551-fall2002/551-9-12-2002.pdf

Notice that there is only one divider for both floating point and integer, and it is not pipelined. This is likely because CPU division is considered uncommon and very area-expensive.

Pipelined hardware division algorithms do exist, typically based on Taylor Series Expansion.

Operational Optimizations

FP16 Support

- Each FP32 unit is capable of a single FP32 operation or 2 FP16 operations
- ½ sized data -> half space & bandwidth required (or 2x bandwidth)
- Useful for machine learning applications, truncate precision to avoid overfitting

Expanded Hardware-supported Atomics

- Maxwell implemented hardware 32 bit int atomicAdd() in shared memory
  - Otherwise, synchronization only possible via locking and compare-and-swap instructions
- Pascal implements 32 & 64 bit int and float atomicAdd.

More Operational Optimizations

Caching

- 64 KB shared L1 & Texture Cache per SM
    - Allocated depending on workload
- 4096 KB dedicated L2 cache shared across GPU

2x GPUDirect RDMA Bandwidth

- Available since Kepler
- More Multipurpose than NVLink
    - Access GPU memory directly from other PCIe devices. Eliminate extraneous memory copies..

16nm FinFET Process

FinFET Transistors

Why is smaller better?

What was wrong with traditional MOSFET?

FinFET Transistors

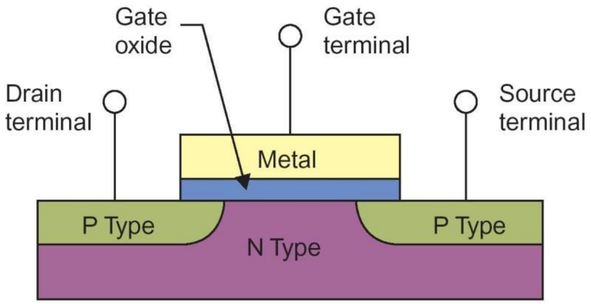
Bigger isn’t always better

Smaller Transistors means more per transistors per unit area

Less power required to activate transistors

- Better performance for same energy cost
  - Useful for modern GPUs
- Same performance for lower energy cost
  - Useful for modern CPUs, MOBILE
- Most likely a balance between the two
  - Vs Maxell, ~ 2x transistors count, 20% energy boost

Why ditch MOSFETs?



Source: greenoptimistic.com

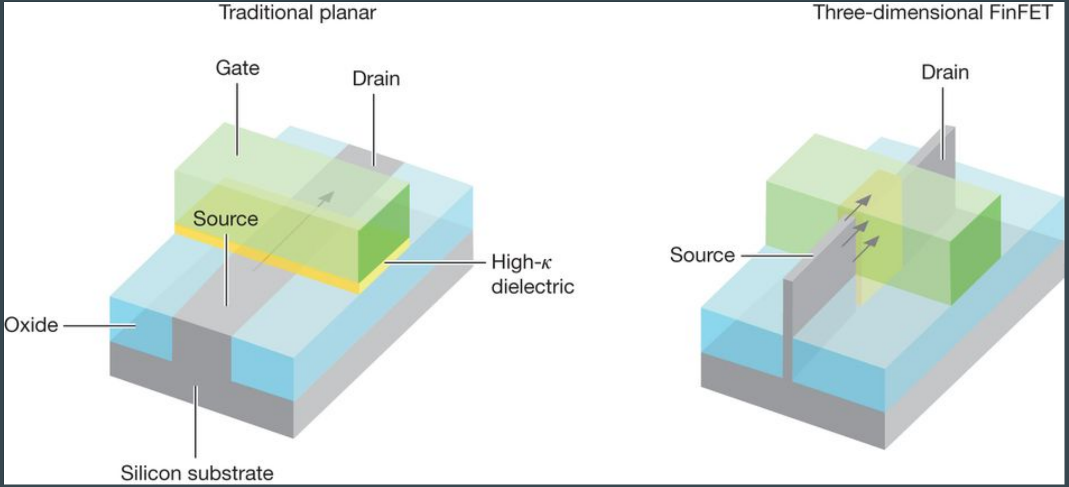
Current issue- Leakage current

- Low Source/Drain and Gate/Source voltage differentials- varies exponentially
- Quantum Effect- Electrons forget which side of the channel they are on

Future Issue- fabrication

- Smallest features are already same width as the wavelength used in Photolithography
- Current estimates say Moore's Law *really* expires in 2030 (for real, this time).

MOSFET vs FinFET



Source: http://www.nature.com/nature/journal/v512/n7513/fig_tab/nature13570_F4.html

MOSFET

- Planar channel controlled by voltage from one side

FinFET

- Well-defined, wire like channel shut off from 3 sides
- “Pinch closed”

NVIDIA NVLink

High Speed Interconnect

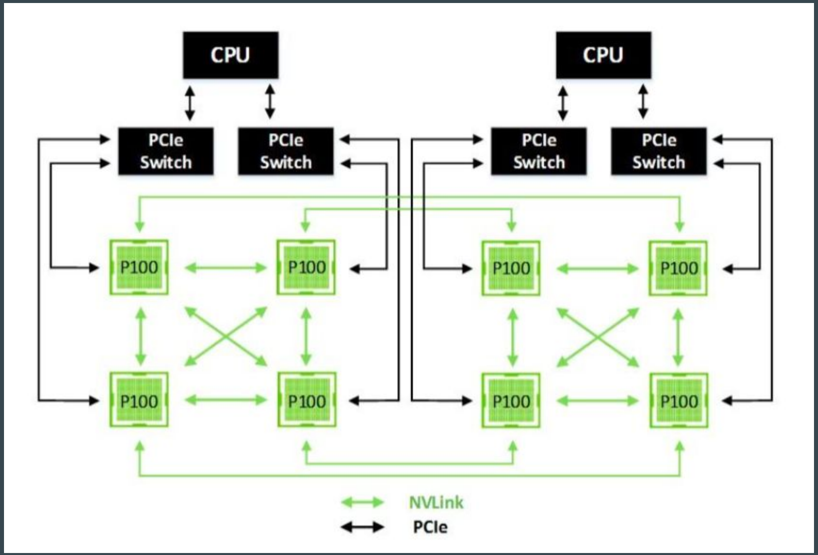
Multiple-GPU systems becoming increasingly prevalent

Interconnect GPUs while reducing load on PCIe bus

Execute directly on memory attached to another GPU

Uses NVIDIA High-Speed Signaling (NVHS)

Source: NVIDIA



Hybrid Mesh Cube Topology

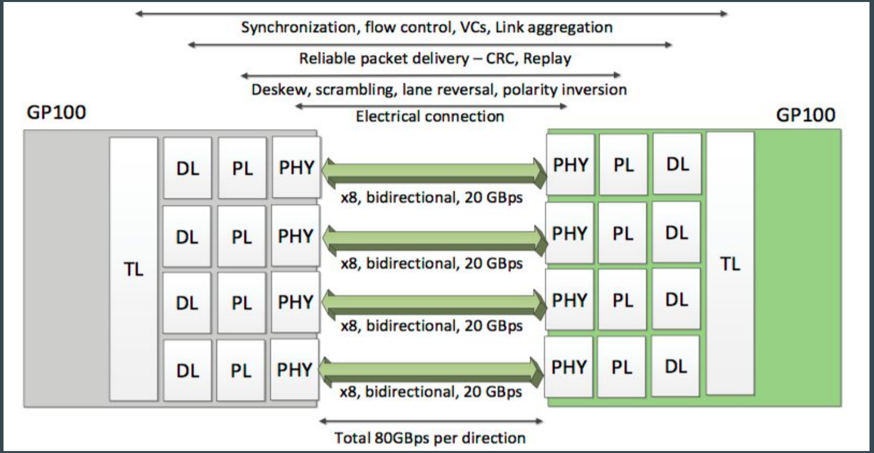
DGX-1

NVIDIA High-Speed Signaling (NVHS)

- Differential Pair @ 20 Gbps
- Eight differential pairs form uni-directional *Sub-Link*
- Two Sub-Links - one in each direction- form *Link*
  - Link supports up to 20 GB/s in each direction for total 40 GB/s bidirectional bandwidth
- Multiple simultaneous Links called *Gangs*
  - Flagship Pascal Tesla P100 supports Gangs of 4 per GPU for max bidirectional bandwidth of 160 GB/s
- NRZ Signalling
  - Non return to zero: 0s and 1s represented by DC Voltages
    - As opposed to Return to Zero (RZ) where between each bit voltage returns to “neutral” zero
- Variable length packets from 1 to 18 Filts
  - Filts: 128 bit data words

NVHS Controller Layers

- Physical Layer (PL)
  - Interfaces with the physical physical layer
  - Responsible for deskew, framing, scrambling/descrambling, polarity inversion and lane reversal
- Data Link Layer (DL)
  - Responsible for reliable transmission
    - 25 bit CRC, replay buffering, ACK
  - Also handles link bringup and maintenance
- Transaction Layer (TL)
  - Responsible for synchronization, link flow control, virtual channels, and link aggregation


Synchronization, flow control, VCs, Link aggregation
Reliable packet delivery – CRC, Replay
Deskew, scrambling, lane reversal, polarity inversion
Electrical connection
GP100
GP100
TL
DL
PL
PHY
x8, bidirectional, 20 GBps
Total 80GBps per direction

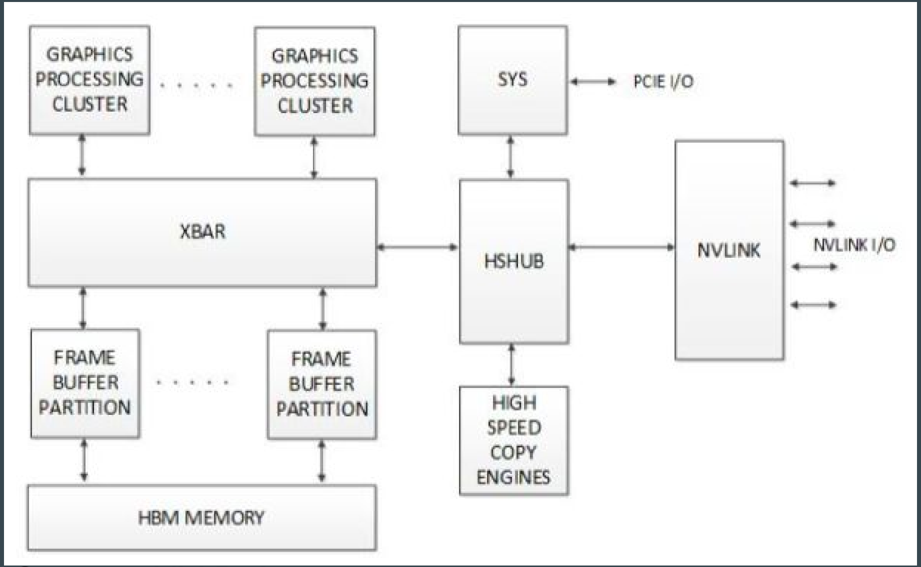
Source: NVIDIA

PCIe Relief

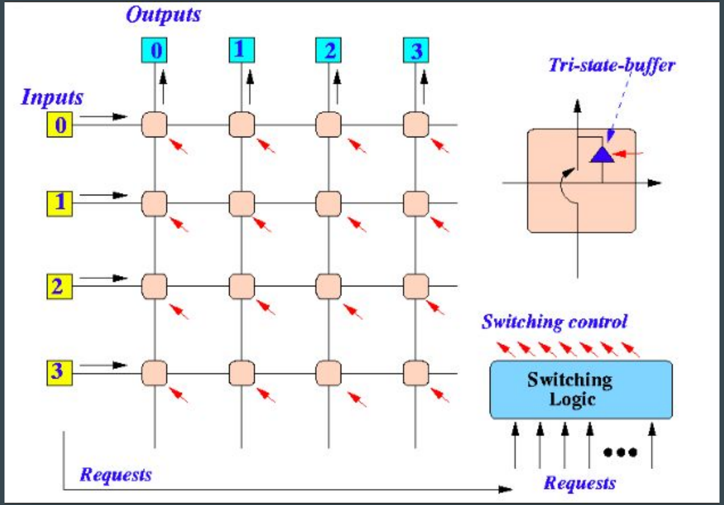
- GPUs can share memory with remote nodes through DMA
- NVIDIA claims 5x bandwidth (160 GB/s over 32 GB/s) and ⅓ energy cost of PCIe 3.0
- Free up PCIe for system memory and NIC access
    - High demand in general for PCIe lanes...
- In IBM POWER8 CPUs, NVLink can circumvent PCIe entirely
    - Partnership with IBM for Oakridge and Livermore National Laboratories exascale supercomputing projects

NVLink in P100

- Two 400 pin connectors: one for NVLink, one for everything else
- NVLink Controller communicates with GPU through High-Speed Hub (HSHUB)
  - HSHUB connected to GPU wide crossbar, High Speed Copy Engines (HSCE), and other elements



Source: NVIDIA

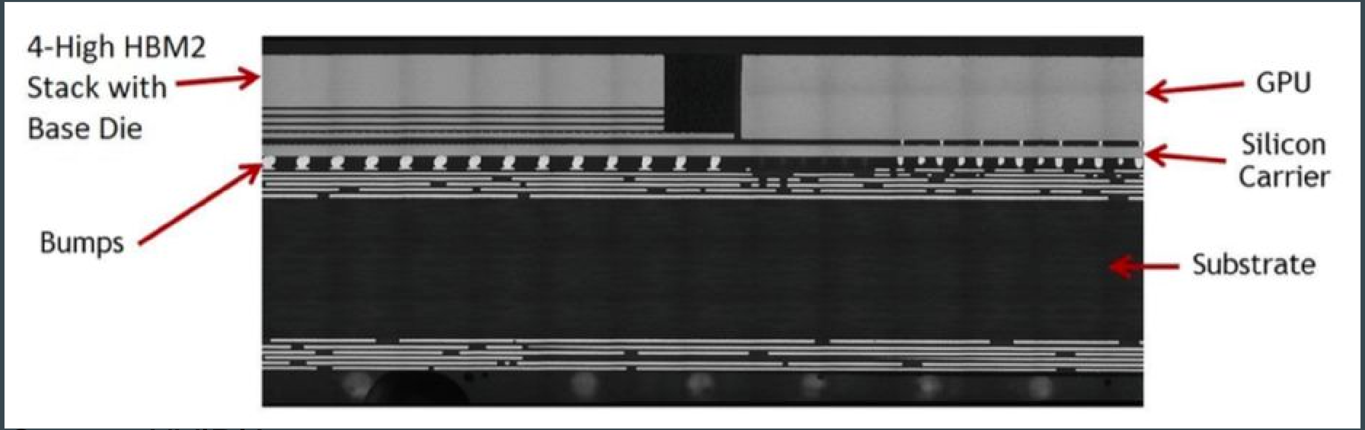


http://www.mathcs.emory.edu/~cheung/Courses/355/Syllabus.linux/90-parallel/CrossBar.html

CoWoS with HBM2

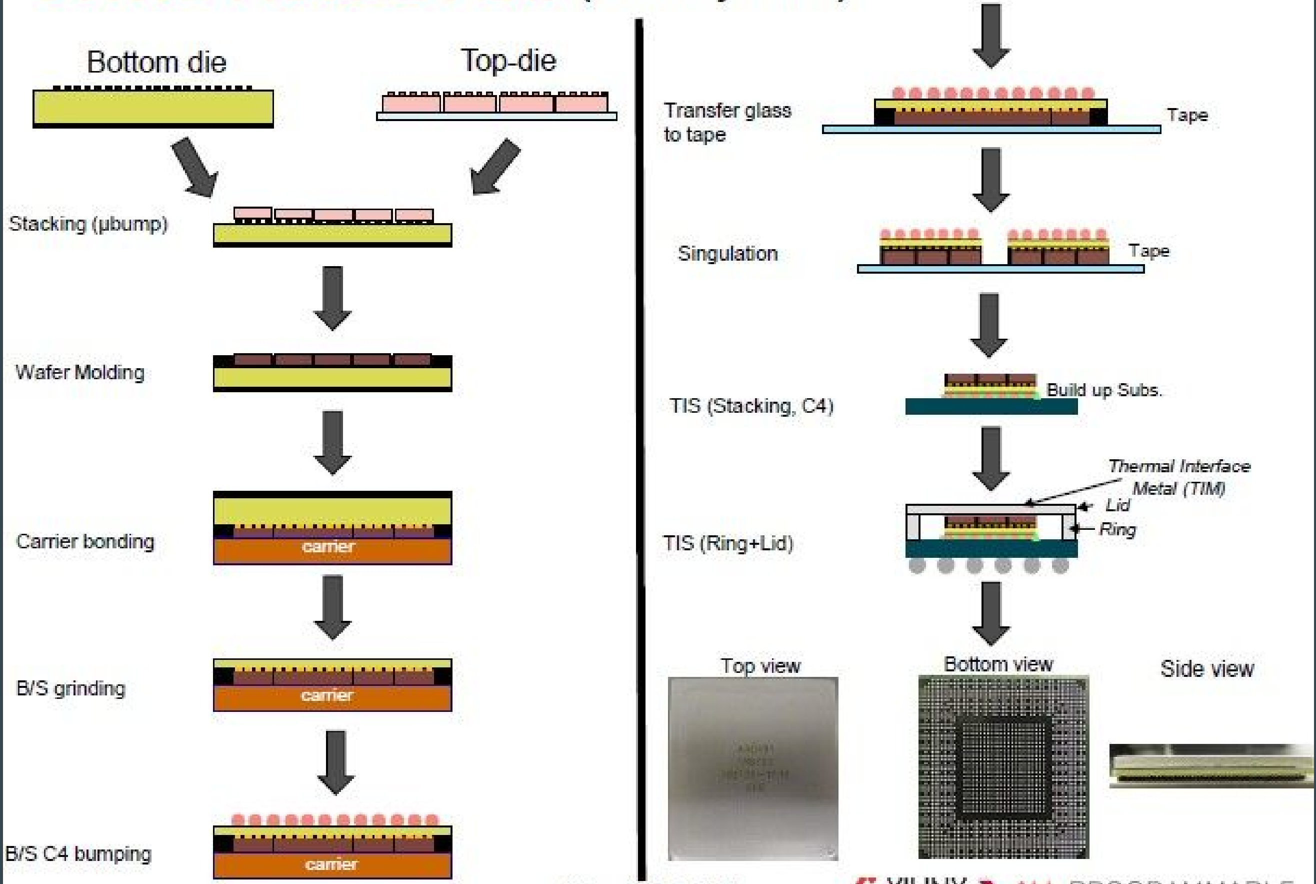
CoWoS with HBM2

- Chip-on-wafer-on-substrate with High Bandwidth Memory 2
- Consists of one or more vertical stacks of memory dies
- Linked with through-silicon vias and microbumps
  - Contrast with GDDR5- discrete memory chips surrounding the GPU
- 3x greater bandwidth compared to GDDR5



Source: NVIDIA

CoWoS Process Flow (Courtesy TSMC)



XILINX ➤ ALL PROGRAMMABLE.

Native ECC Support

- HBM2 supports native Error Correcting Code support.
- ECC detects and corrects memory bit corruption before error propagates to system
- ECC was a implemented in older GDDR5 based systems as a toggleable feature- but at huge performance costs
  - 5%-10% storage space overhead
  - 10%-15% memory bandwidth overhead
- Tesla P100 protected by Single Error Correct Double Error Detect (SECDED) ECC

Error Correcting Codes

Generally, memory corruptions are assumed to be independent of one-another, so multiple cospatial corruptions are considered very very rare. Thus, single correction is considered sufficient but multiple detection is still important.

Triple Modular Redundancy: Store three copies of each bit, majority vote

- Space-expensive
- Very fast (no computation required!)

Hamming Codes: Clever, limited redundancy based on parity

- Extended Hamming Codes are single error correcting, double error detecting
  - Sound familiar?
- Any given bit is included in a unique set of parity bits, erroneous bits are sum of indicating indices

Extended Unified Memory and Compute Preemption

Unified Memory Predecessors

- Single addressing scheme for both GPU and host memory
- Fermi (2009) - Unified GPU address space
  - GPUs only, simpler compilation via single load/store insn for all GPUs, C pointers
- CUDA 4 (2011) - UVA: Contiguous virtual memory address space for CPU + GPUS
  - Pinned CPU memory addressable from GPU over PCIe - no memcpy required
    - This tends to be slow...
- CUDA 6 (2014) - Managed Memory
  - Pool of data accessible to CPU and GPU with single pointer.
    - Looks like GPU memory to GPU and CPU memory to CPU
  - CUDA system software automatically migrate data between CPU and GPU
  - Required synchronization before kernel launches and max managed memory limited to GPU memory size

Pascal Unified Memory

- 49 bit virtual addressing
  - Covers 48 bits of CPU memory + theoretical 48 bits of GPU memory
  - Managed memory no longer limited to GPU memory capacity
- Memory page faulting - i.e. CPU style memory management
  - Because how else do you implement virtual addressing > physical address space?
  - Don’t need to synchronize memory with GPU before kernel calls
    - Non resident memory page faults, swapped in OR mapped over PCIe / NVLink
    - Global data coherency guaranteed
      - Still need to take proper precautions to avoid race conditions
- Memory allocated with default OS allocator accessible from both GPU and CPU

Pascal Unified Memory

- Simpler programming experience, lower barrier to entry
  - Focus on parallel implementation, not memory management
  - Developer device/host memory management is an optimization
    - Managed memory is sometimes optimal
- Structure of data more manageable
  - Data structures and C++ classes, nested structures automatically copied to GPU
  - Operate on larger than memory datasets, let system swap in as necessary
- Data always local
  - Always access from HBM2 rather than going to system memory
  - Doesn’t help if continuously swapping...
    - CUDA 8 contains explicit prefetch instruction cudaMemPrefetchAsync()

NUMA

Question: If GPU supports paging, where does distributed memory live at any given time?

Answer: Follow Non-Uniform Memory Access (NUMA) Paradigm

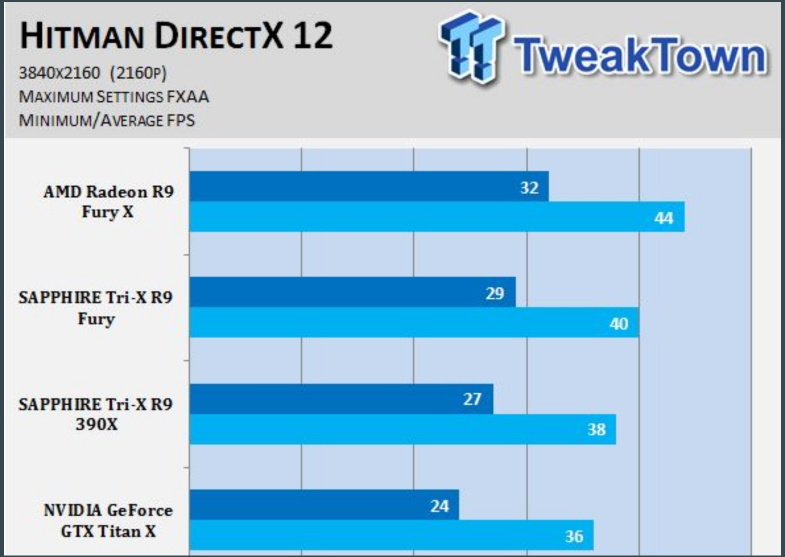
- Likely the basis for NVIDIA distributed memory model
- Idea: improve system performance by bringing memory nearer to the cores
- Memory is classified into Nodes, have *affinity* for best performance device
- *NUMA Placement*: Process of assigning memory from NUMA Nodes
    - Affects the performance, not the correctness
- Supported in many Unix-style OS

NUMA Coherency

- IPC between cache controllers
  - Leads to poor performance when accesses to same memory address on different controllers
  - Cache miss might:
    - Require going to main memory
    - Request valued from multiple caches, get bombarded with responses
- Scalable Coherent Interface (SCI)
    - Maintain list of nodes sharing cache line in associated cache with list status
- MESIF Protocol
    - Extension of MESI (Modified, Exclusive, Shared, Invalid) protocol
      - Cache lines are designated one of these states
      - Reads/writes influenced by state
    - MESIF adds *Forward* state
      - Special Shared State
      - Designated responder to cache requests

Compute Preemption

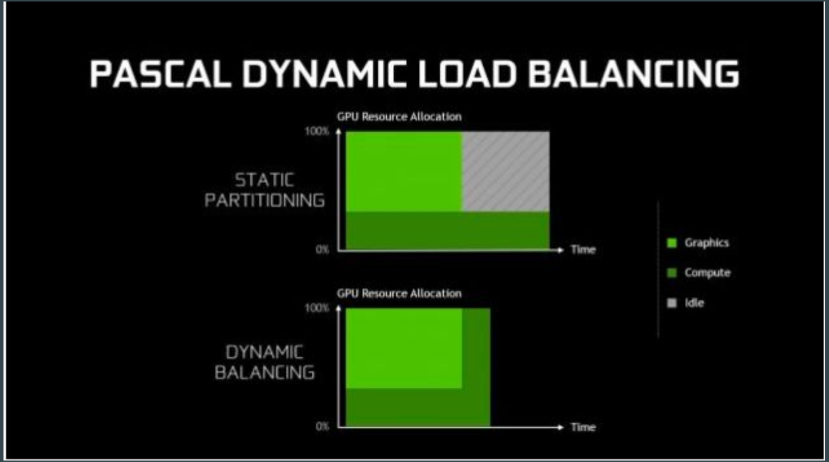
- NVIDIA's attempt to respond to AMD Async Compute
    - DX12 benchmarking showed AMD Fiji best, Intel Skylake ok, and NVIDIA Maxwell dead last
    - Poor performance apparently due to poor resource sharing on NVIDIA GPU



Compute Preemption

- Allows system to pre-empt ill behaved or long running tasks that monopolize the system
  - Previously, careful coding for resource sharing required to execute display + compute jobs simultaneously
- Instruction Level Preemption rather than block level preemption
  - Previously, all threads required to finish before context switch allowed
    - Breakpoints prevented preemption...
  - Allows robust & lightweight debugging, less/no compile time instrumentation

Source: NVIDIA



Finer grain control also used to implement dynamic load balancing.

(some fanbois object)

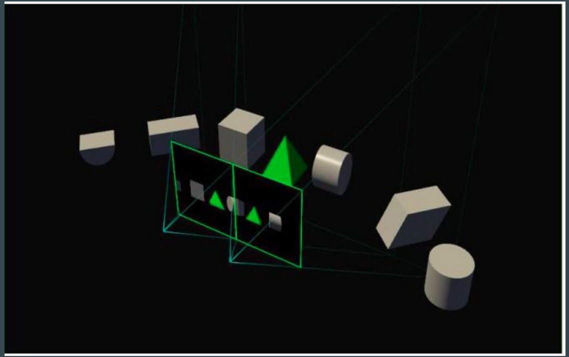
Quick GTX1080 Roundup

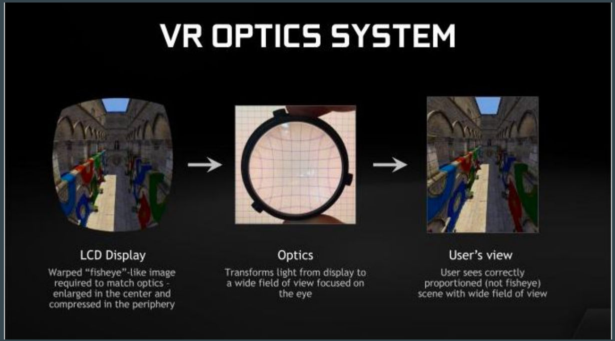
Simultaneous Multi Projection

Engine can reproject geometry from 16 viewports, each with arbitrary angle, tilt, rotation or viewpoint.

Use cases:

- Single Pass Stereo - cheaply render 2 eyes instead of running each eye through entire pipeline
- Lens Matched Shading - approximate lens shape and produce distortion




VR OPTICS SYSTEM
LCD Display
Warped "fisheye"-like image required to match optics - enlarged in the center and compressed in the periphery
Optics
Transforms light from display to a wide field of view focused on the eye
User's view
User sees correctly proportioned (not fisheye) scene with wide field of view

Source: NVIDIA

Misc. GTX 1080 Features

Enhanced SLI

- Faster, new bridging modes

Fast Sync - alternative to V-sync, high latency way of preventing tearing

- Traditional: game engine generates frames sent to DX
- Solution: decouple render pipeline from display hardware. Frames stored in frame buffer, GPU chooses which to scan to display

HEVC Encode / Decode Hardware

- 4k/5k video streaming

Works Cited

Rabaey, Digital Integrated Circuits, Second Edition

Patterson and Hennessy, Computer Organization and Design 4th Edition

https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf

http://international.download.nvidia.com/geforce-com/international/pdfs/GeForce_GTX_1080_Whitepaper_FINAL.pdf

https://devblogs.nvidia.com/parallelforall/inside-pascal/

https://pdfs.semanticscholar.org/21dd/e7a53fe6c26fd99cbba26ee5dde0938672c0.pdf

Works Cited

http://meseec.ce.rit.edu/eecc551-fall2002/551-9-12-2002.pdf

http://queue.acm.org/detail.cfm?id=2513149

https://en.wikipedia.org/wiki/Non-uniform_memory_access

https://en.wikipedia.org/wiki/Scalable_Coherent_Interface

https://en.wikipedia.org/wiki/MESIF_protocol

http://www.rfwireless-world.com/Tutorials/floating-point-tutorial.html

http://www.slideshare.net/zbhavyai/implementation-of-32-bit-alu-using-vhdl?next_slideshow=1

Works Cited

https://developer.nvidia.com/content/life-triangle-nvidias-logical-pipeline

https://en.wikipedia.org/wiki/Triple_modular_redundancy

https://en.wikipedia.org/wiki/Hamming_code

http://www.anandtech.com/show/10325/the-nvidia-geforce-gtx-1080-and-1070-founders-edition-review/11

Thanks for Listening!