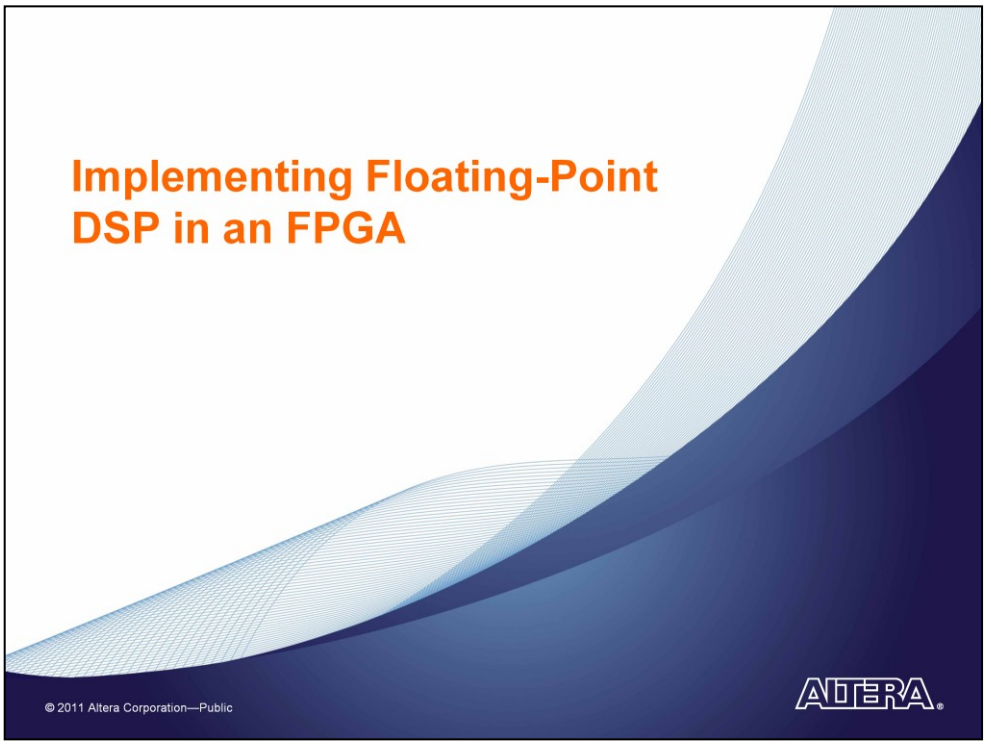
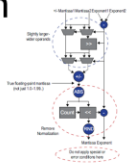



For questions or comments, please contact Jordon Inkeles, DSP Marketing

## Agenda

- Floating-point overview
- Options and challenges of implementation
- Floating point in Altera® FPGAs
- Altera's floating-point digital signal processing (DSP) flow
- BDTI white paper



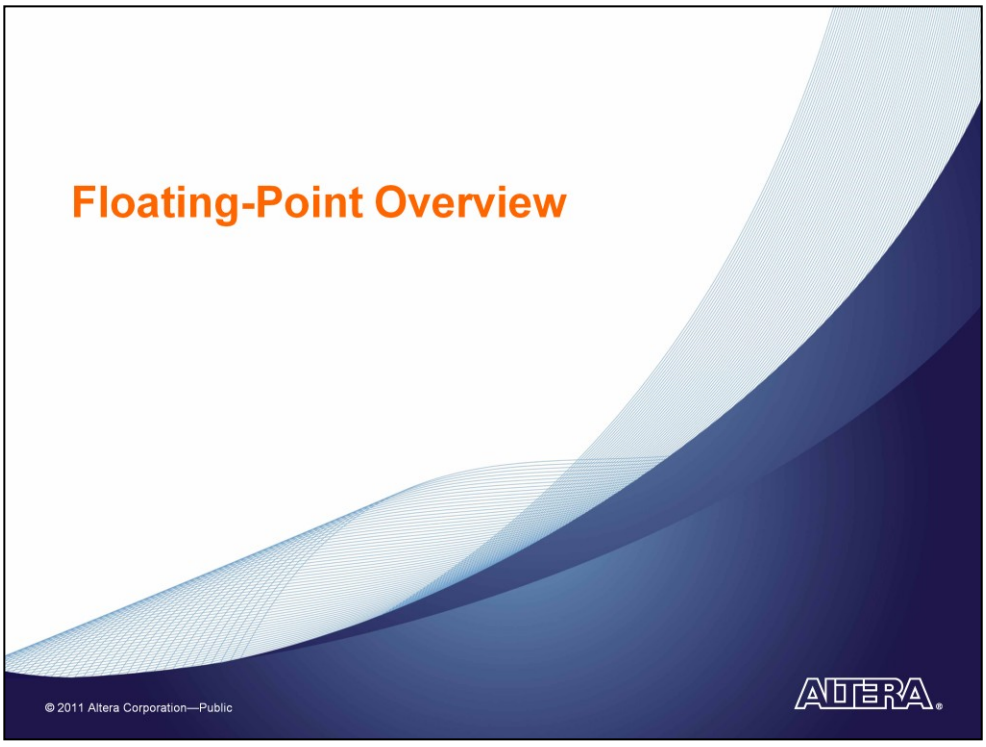





Engineers have long developed DSP algorithms using floating point due to the wide dynamic range. Often in C using float or double type designation or within Mathworks MATLAB modeling environment. However, when it comes to implementing DSP algorithms in HW, the algorithms are most often turned into fixed point algorithms. This conversion can take weeks or months and lead to trade-offs in the algorithm. The reason for the conversion is the inability for HW to implement Floating Point in a cost efficient manner while maintaining adequate performance. Altera has been working to solve this problem for some time and have developed a powerful new floating point design flow for FPGAs.

So, in this webcast we will start with the basics of floating point. Then we then discuss options for implementing floating point DSP, this includes CPUs DSPs, GPUs, and FPGAs. Next we will discuss the innovations Altera has made to achieve, cost-effective high-performance Floating point DSP in an FPGA, we will then walk through Altera' s DSP Builder tool and finally wrap-up with a couple examples and validation conducted by BDTI, one of the leaders in independent DSP analysis.

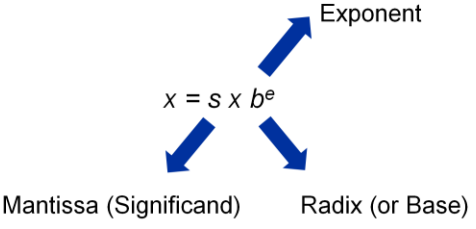
If you are well-versed in floating point notation, IEEE floating point standard, and the basics of floating point arithmetic. Please feel free to jump to slide 11, and skip the first section "Floating Point Overview"




Let’s begin with the Floating Point Overview

## What is Floating Point?

- **Fixed-point** representation
  - Computer representation of an integer → 1250
- **Floating-point** representation
  - Computer representation of a real number
  - Example: 1250.357 → $1.250357 \times 10^3$
- Mathematical convention





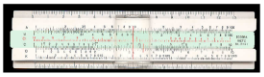


What is Floating Point? It's easiest to start with fixed point. A Fixed point number is simply the computer representation of an integer. So, floating point is the computer representation of a real number. Let's look at the two examples. One thousand two hundred and fifty is a number that can be represented in fixed point decimal with 4 digits. If the number has a fractional portion, like in our second example, .357, you can represent the number in scientific notation, once normalized this is essentially a floating point representation of the number

A floating point number has several components: a matissa, an exponent and radix or base. Which for computers is typically is binary or base 2. But this wasn't always the case. Let's have some fun and take a quick tour of the history of floating point as described in the handbook for floating point arithmetic, a great book if you don't have it.

## Interesting History of Floating Point

- Numerical representation
  - Babylonian era: √2 approximated with radix 60
  - 1630: slide rule was invented



- In computers
  - 1941: Z3 computer (developed by Konrad Zuse)
    - Radix 2, 14-bit significand, 7-bit exponent, 1-bit sign

  - 1958: Setun (developed by Moscow University)
    - Radix 3, used 3-valued ternary logic for the whole computer
  - 1965: IBM 360
    - Radix 16, 24-bit significand, 7-bit exponent, 1-bit sign

  - 1980: Intel 8087 mathematics coprocessor
    - 50,000 floating-point operations per second (FLOPs)

  - 1993: Intel Pentium
    - Includes floating-point unit (FPU)


**Note**: History Details from J.-M. Muller et al., *Handbook of Floating-Point Arithmetic*, Birkhauser Boston , a part of the Springer Science+Business Media, LLC 2010



5

ALTERA.

Scientific notation or floating point representation of numbers can be seen dating all the way back to the Babylonian era with the square root of 2 with a radix of 60. The slide rule, invented in the sixteen hundreds, was also a form of floating point. While this was long before the computer era, some of the earliest computers did use floating point. For example, the Z3 computer used a radix 2 a 14-bit mantissa, a 7 bit exponent and 1-signed bit.

The Setun, developed in Russia, used a Radix 3. The IBM 360 main frame used radix 16. In 1980, the PC market was taking off and Intel developed the 8087 math coprocessor for use with it x86 CPUs. This was able to implement fifty thousand floating point operations per second. Later x486 and pentium became one of the first CPUs to integrate a Floating Point unit on the same die.

As floating point became more pervasive, as in desktop computers, it was clear there needed to be a standard.

….Other notes…..

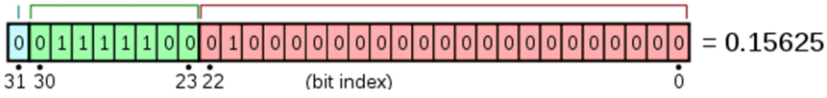
IBM was a  main frame

Motorola had equivalent FPU

1993 Intel Pentium (include FPU)

# IEEE 754 Standard for Floating Point

- In 1985, the floating-point standard was set
  - IEEE 754 in 1985, 1987, and 2008
- The standard gave accuracy, reliability, and portability
  - Includes formatting, encoding, rounding algorithm, exceptions, etc.
- IEEE 754 single-precision standard




| Standard | Format | Sign (# of Bits) | Mantissa (# of Bits) | Exponent (# of Bits) |
|---|---|---|---|---|
| binary16 | Half precision | 1 | 10 | 5 |
| binary32 | Single precision | 1 | 23 | 8 |
| binary64 | Double precision | 1 | 52 | 11 |
| binary128 | Quadruple precision | 1 | 112 | 15 |




In 1985, IEEE created the first widely accepted standard for floating point numbers. The standard was originally developed for binary numbers but subsequently updated in 1987 to include decimal and other bases. In 2008, the standard added quadruple floating point and standardized on the fused multiply-add operation.

The standard gave accuracy, reliability and portability for the representation of numbers in computers. The consistent format, which you can see in the pink and green diagram show how many bits are allocated to each portion of the floating point number. For single precision, there are 32 total bits, 23 bits for the mantissa, 8 for the exponent and 1-signed bit. There is also an encoding scheme for the standard. For example, there is an implicit 1 in front of the mantissa leading to 24 actual bits of precision and in order to have a negative exponent, the exponent is biased by 127. For example, if you wanted to represent and exponent of 3 you would actually use 130 (127+3). That leaves 1-127 to represent negative exponents.

In the table you can see the different formats. The most popular being single precision and double precision. You may also hear single extended and double extended, they aren't shown in the table.

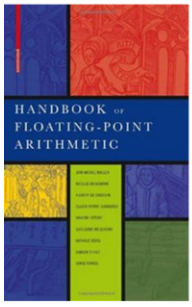
## Floating-Point Arithmetic

- Floating-point addition (in decimal)

$$\begin{array}{l} 1.05 \times 10^{3} \\ 2.25 \times 10^{5} \\ \hline \end{array} \Rightarrow \begin{array}{l} 0.0105 \times 10^{5} \\ 2.25 \times 10^{5} \\ \hline 2.2605 \times 10^{5} \end{array} \Rightarrow \begin{array}{l} 0.01 \times 10^{5} \\ 2.25 \times 10^{5} \\ \hline 2.26 \times 10^{5} \end{array}$$

- Floating-point multiplication (in decimal)

$$\begin{array}{l} 1.05 \times 10^{3} \\ 2.25 \times 10^{5} \\ \hline 2.3625 \times 10^{8} \end{array}$$




- Other key notes
  - Rounding (e.g. round-off error)
  - Non-representable (e.g. 0.1 in binary)
  - Normalize and denormalize (less precision)
  - Overflow (exponent too large) and underflow (exponent too small)




ALTERA.

To understand why floating point is more challenging to implement in HW than fixed point, we can look at a few examples. For the sake of simplicity we use the decimal format, The first example is floating point addition. When adding two floating point numbers, the exponents must be the same, if they are not, the decimal point needs to be shifted. This is called demoralization. In the example shown, 1.05…. times 10 to the three….is converted to…. point 0 1 0 5 times 10 to the 5. Once the exponents are the same then you can add the mantissas and carry the exponent. However, in the right most addition example, the two least significant bits were dropped as part of the denormalization. this maybe the case if you don't have enough bits of precision to keep them. This shows that denormalization can lower your precision. This is an example why double precision with more bits in the signficand is more accurate that single precision. It also shows that not all numbers can be represented in computers exactly as they are but floating point provides the best representation given the bits of precision.

In the multiplication example, the mantissas are multiplied and the exponents are added.

The IEEE standard also covers other aspects including rounding, numbers that need a special designation…. like infinity, normalization and denormalization… and over and under flows.

While floating-point addition and multiplication are both commutative ($a + b = b + a$). they are not necessarily associative. That is, ($a + b$) then added $c$ is not necessarily equal to ($b + c$) then added to a. The order matters… this is due to various factors including rounding and/or de normalization.

## Floating-Point Benefits

- Greater dynamic range
  - Represent large numbers and small numbers
  - At the expense of precision


- Example 1: largest 3-digit number (decimal/positive)
  - Fixed → 999
  - Float → $99 \times 10^9 = 99{,}000{,}000{,}000$
    - (2 digits for mantissa and 1 for exponent)
- Example 2: precision of 3-digit number (decimal/positive)
  - How to represent 125 in fixed and floating point?
  - Fixed → 125
  - Float → $12 \times 10^1 = 120$; or $13 \times 10^1 = 130$ – can't represent 125




ALTERA.

Given the added complexity of floating point arithmetic in a computer, there must be a good reason to use floating point vs. Fixed point. The reason is dynamic range, or the ability to represent very large and very small numbers at the same time. However, floating point comes at the expense of precision. For example, a 32-bit fixed point number has better precision that a 32-bit floating point number. This is due to all 32 bits in the fixed point number being used for the mantissa and only 23 bits being used for the mantissa in the floating point example. The floating point number needs some of the bits to communicate the exponent, the exponent on the other hand is what gives the dynamic range.

Lets again use an example using decimal numbers to clarify. In example 1, we want to create the largest number we can using only 3-digits. That leads us to 9 9 9 or nine hundred and ninety-nine. Now we want to create the largest number we can using floating point with the same 3 digits. I have arbitrarily chosen two of the digits for the mantissa 9 9 and one digit for exponent 9. In this case, 9 9 9 would represent 99 times 10 to the 9 or 99Billion. This is signficanly larger than 999. So using the same number of digits (or bits in binary), floating point offers much larger numbers. We can also create a negative exponent to create really small numbers in floating point. This is a simple example of greater dynamic range.

Let's looks at example 2 and see where we loose precision when using floating point. Using the same constraints of 3 digits in decimal. We try to represent the number 125. Very straighforward in fixed point. Now let's try and represent 125 in floating point. Using the same number of digits for the mantissa and exponent as the last example. We can represent 12 times 10 to the 1 or 120, we can also represent 13 time 10 to the 1 or 130. but we can't represent 125, therefore we would need to round to 130 and loose some precision. If we had another digit in the mantissa, we could represent 125.

## Floating-Point Applications (1)

- **Military**
  - Radar and electronic warfare, requiring back-end space-time adaptive processing (STAP) and Doppler radar using fast Fourier transform (FFT)
- **Wireless**
  - Channel card, requiring multiple-in multiple-out (MIMO) algorithms
  - RF card, requiring digital predistortion (DPD) feedback
- **Medical imaging**
  - MRI and CT scan, requiring back-end data processing
- **Test and measurement**
  - Spectrum analyzers and VSAs, requiring large FFTs



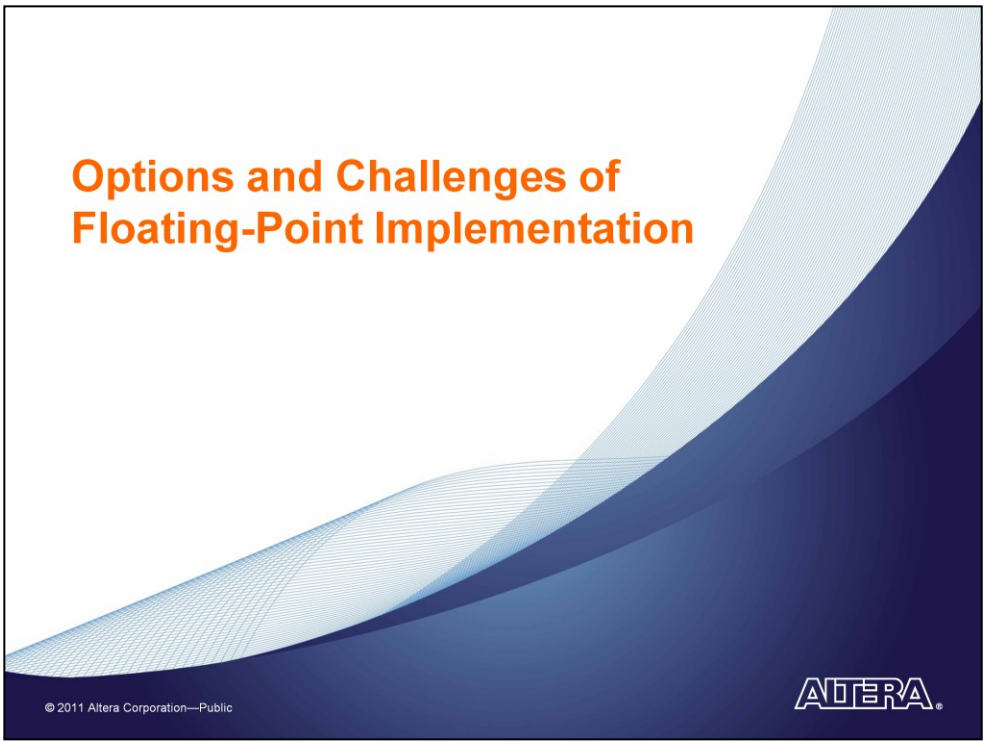

Now that we have completed a refresher in floating point representation and arithmetic. Let's briefly look at some of the markets and applications that would benefit from floating point. Military market, is the first that comes to mind…. Specifically radar and electronic warfare are good candidates for floating point, this includes pulse doppler or more advanced radar……. such as….. space time adaptive processing. Another example, is with MIMO algorithms and beam-forming algorithms in the wireless market. Medical imaging and specifically MRI and CT scans….. in addition to test and measurement applications requiring very large FFTs would also benefit from floating point.

# Floating-Point Applications (2)

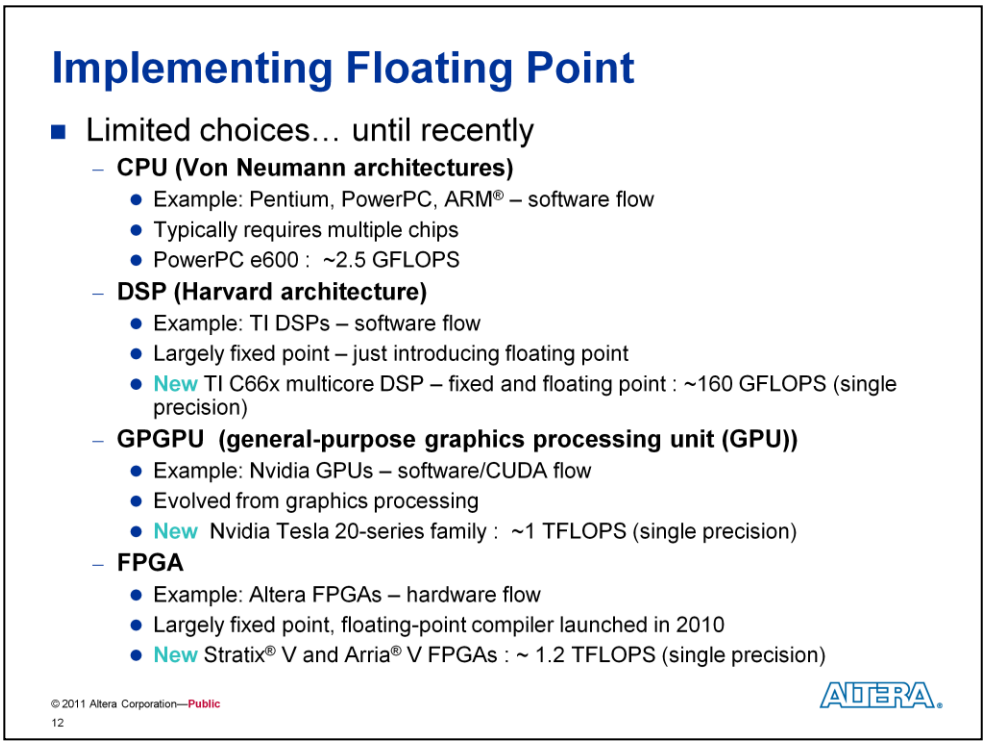
- **High-performance computing**
  - Financial and Monte Carlo modeling
  - Weather and seismic system, Synthetic Aperture Radar (SAR), molecular dynamics, etc.
  - Cryptography and National Security applications
- **Industrial**
  - Motor control




The High performance computing market…. who has long used CPUs to implement floating point are very interested in accelerating their algorithms. Other examples include Motor control in the industrial market. Towards the end of this webcast, I will briefly cover two examples, a matrix inversion and a motor control design.




Let's look at some of the implementation options for floating point DSP algorithms, this includes CPUs, DSPs, GPUs, and of course FPGAs.




There have been limited options to implement floating point DSP algorithms, until recently. Let's take look at the options. The CPU with a Floating Point Unit is where most engineers have been implementing floating point DSP algorithms. This is using the typical Von Neumann architectures in a Pentium, ARM or PowerPC. The PowerPC was used extensively in radar designs for the past 10-15 years. However, the performance was limited and peaked at a few GFLOPS per device. To implement higher performance you often saw 10-15 CPUs on a single board, causing a challenge on how to partition your algorithm and running into power constraints.

DSP processors are largely fixed point and not until recently had a real floating point solution. TI is trying to change this with the new C66x multicore DSP processor that offers both fixed and floating DSP capability. However, the floating point implementation is only a ¼ efficient in terms of multipliers compared to implementing in fixed point. They claim about 150 GLOPS in single precision…. though this a peak number and probably far the sustained GFLOP number

The rise of the GP GPU or general purpose graphics processing unit starts to provide some high performance floating point capability. Nvidia who has been developing Graphics Processing Units for desktops for years is adapting the technology for general purpose calculations. This requires an entirely new software flow. For some time, engineers often represented their floating point arithmetic in terms graphics…. or vertex and fragment shaders to take advantage of the floating point capability in GPUs. Then Nvidia developed CUDA a few years ago to allow engineers to engage floating point capabilities of GPUs without knowing about graphics. This is essentially the General Purpose – GPU. Nvidia claims their new Nvidia Tesla series can hit 1 TFLOPS. The biggest challenge with GPUs, other than learning the new software flow, is GPUs are very power hungry.

FGPAs have long been used for fixed point DSP algorithms…. given the inherent parallelism of FPGAs and the large number of DSP blocks containing dedicated multipliers. Altera recently launched the floating point compiler within the DSP Builder Advanced Block set tool. DSP Builder can be used for both fixed and floating point algorithms and allow you to mix fixed and float within the same design. Using the new floating point compiler from Altera, you can now achieve high performance floating point in an FGPA. An example includes the Stratix V FPGA with a sustained ability to achieve over 1 Teraflops.

Let's look at all these options in a table

## Floating-Point Comparison Table

| | CPU | DSP | GPU | FPGA |
|---|---|---|---|---|
| Performance | 1X | 10X | 50X – 100X | 100X |
| Power | 1X | 1X | 5X –10X | 1X – 2X |
| I/Os | Very limited | Limited | Limited | High bandwidth |
| Longevity | Medium | Medium | Worst | Best |
| Complete system | Yes | No (requires CPU) | No (requires CPU) | Yes |
| Tool flow | Software | Software | Software with multicore | Model-based |

***FPGAs Offer Performance Advantage over CPUs/DSPs***

***FPGAs Offer Power Advantage over GPUs***





The next slide shows a table comparing the implementation options. Let's walk through the table. From a performance perspective, GPGPUs and FPGAs will offer the highest performance floating point capability. From a power perspective, the power hungry GPUs start to become a challenge. As we look at I/Os, DSP Algorithms are very data intensive, the ability to move data on an off the chip is also a challenge, FPGAs offer very high throughput….. transceivers from 3, 6, 11, and even 28Giga bits per second are supported. For longevity, FPGA companies rarely…. if ever…. obsolete a product line as they can be used in broad applications and typically don't have 100s of variants, which is not the case in many CPUs and DSPs. For a complete system, the CPU offers the ability to often incorporate a complete system in a single device, not typically the case for DSPs or GPUs as they both rely on a CPU for the control function as they handle the data computations workload. An FPGA can host an entire system, in many cases, engineers implement a CPU, or many CPUs, on the same FPGA. The Nios II 32-bit RISC processor is a popular example. For tool flow, CPUs and DSPs offer a simple to use c-based flow, GPUs offer a hybrid of C and HW or API based flows. FPGA designs are typically implemented in Verilog or VHDL, but using the floating point compiler, engineers can stay in the popular MATLAB/Simulink environment, often where the algorithms were originally created. Altera's DSP Builder tool automatically generates the performance and logic optimized Verilog or VHDL.

In general, there may be several factors when choosing your architecture, but just looking at floating point implementation…. FPGAs offer the best performance over CPUs and DSPs….. While offering the best power advantages over GPUs. Most easily seen in GFLOPS/watt

http://cache.freescale.com/files/32bit/doc/data_sheet/MPC8641DEC.pdf

## Floating-Point Implementation in FPGAs

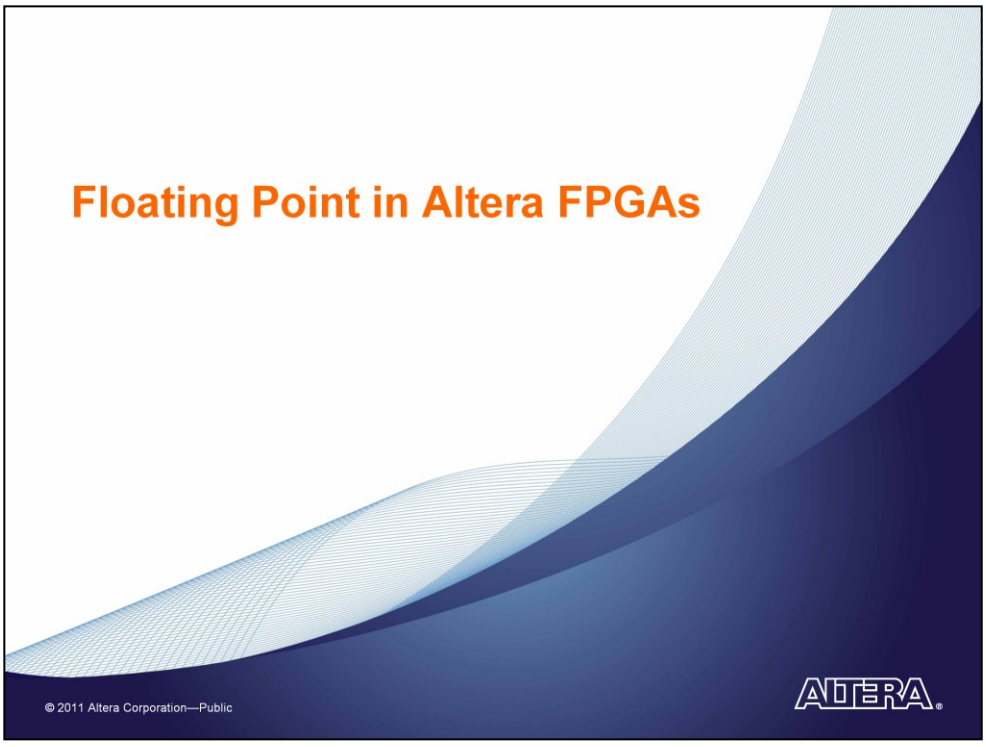
- Historical challenges
  - **Efficiency** – floating point required significant logic and multipliers
  - **Performance** – floating point stressed the routing resources in an FPGA leading to low $f_{MAX}$
  - **Floating-point tool flow** – No C-based or model-based flow
- How did Altera solve these challenges?



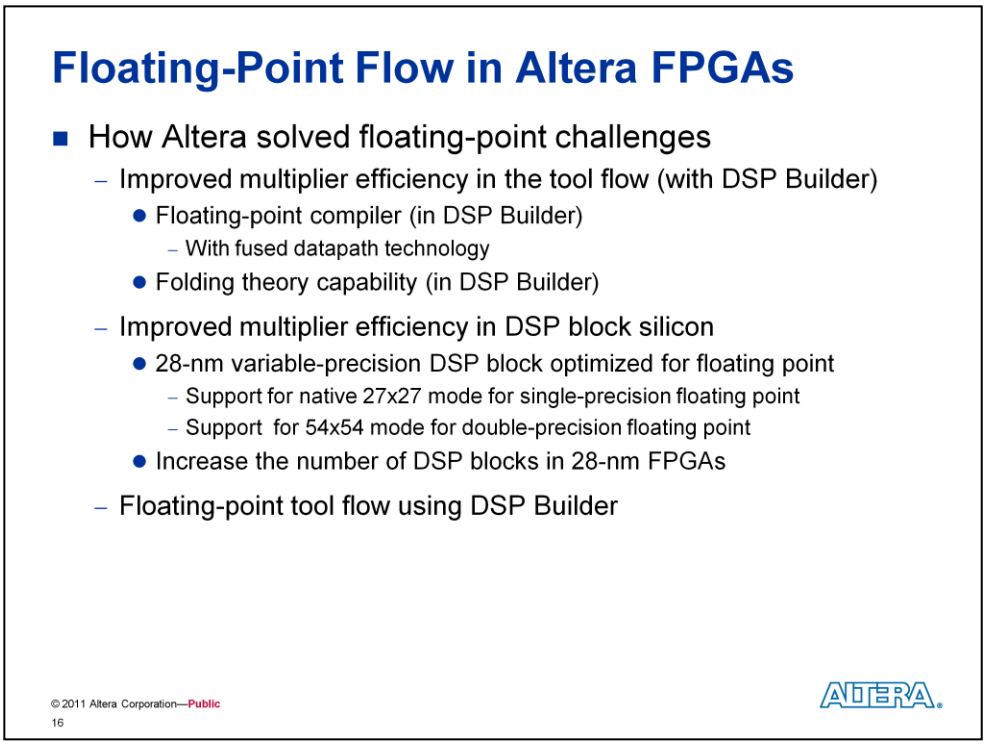

ALTERA.

So what's changed, why is it now possible to efficiently and with high-performance implement floating point DSP algorithms on an FGPA. FPGAs have certainly had their challenges, traditionally floating point has required significantly more logic than fixed point, more multipliers are required, and given the wider data path, the routing was often stressed creating bottle necks and leading to low performance.

Let's see how Altera has solved these challenges and now offers the most compelling floating point implementation on the market.




Floating Point in Altera's FPGAs




To overcome some of the traditional challenges of implementing floating point in an FGPA. Altera made some significant Innovations.

On the tool flow…

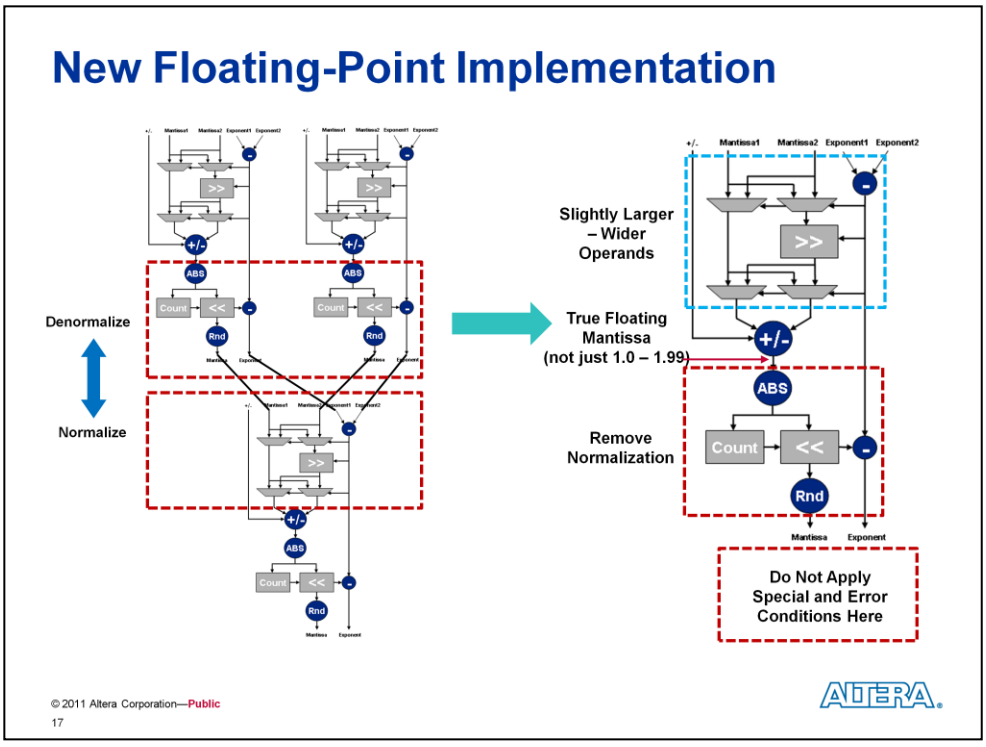
Altera has made significant improvements to DSP builder including the floating point compiler with fused datapath technology and enabled folding capability. This brings floating point capability to all current Altera FPGAs.

On the silicon side….

Altera has improved the multiplier efficiency in the DSP Block by adding a native 27x27 bit mode, perfect for implementing single precision floating point multiplication. If you remember from the IEEE standard, the single precision type has a 23-bit mantissa that needs to be multiplied, well within the 27x27 capability but not as efficient as chaining 18bit multipliers together…. (as competing FPGAs have to do). In addition, Altera has a 54x54 mode to implement double precision floating point.

In addition to the improved multiplier efficiency, Altera has added significantly more multipliers than previous generations.

Let's look at some of these Innovations in more detail.

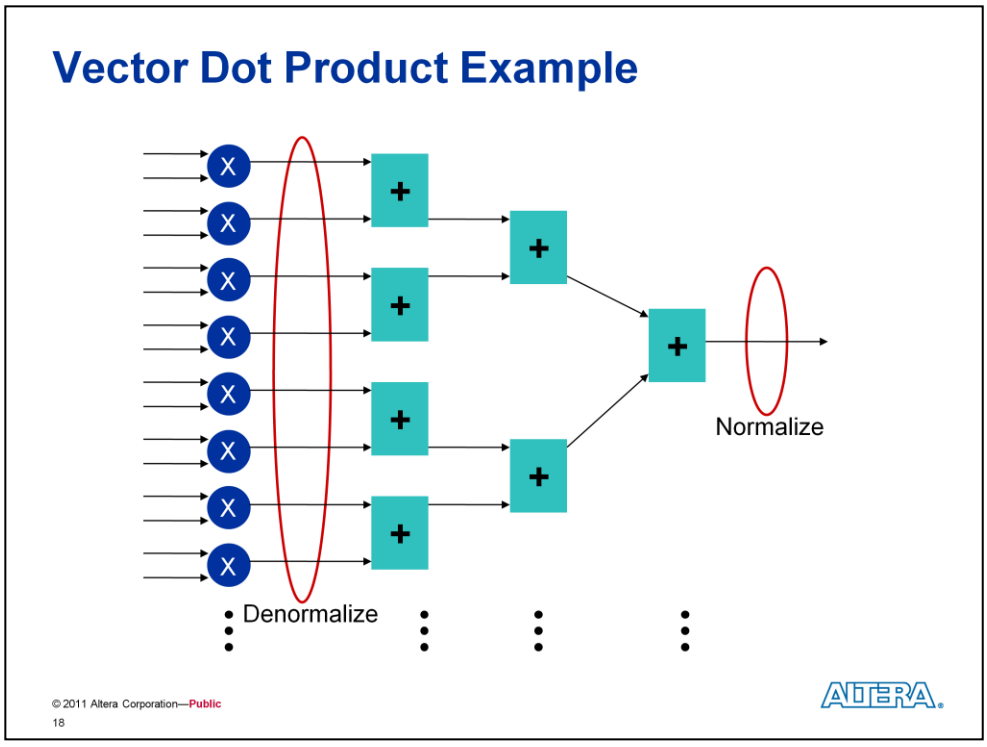



First is the fused-data path technology in the floating point compiler.

If we look at the IEEE754 floating point standard. It is not well suited for FPGAs, and using this approach through-out the data path will lead to poor performance, and excessive routing and logic usage.

Instead, Fused Datapath supporst IEEE754 only at the functional boundaries. Within a function or datapath, the circuit design is optimized, which results in a “fusing” of various floating point operations

For example, rather than using a 23-bit *mantissa* in the compiler. The fused datapath extends it to the full 27 bits of the multiplier. This reduces the number of normalzizations and denormalizations required. By taking advantage of the extended precision capability of the multiplier, significantly less logic  resources can be used.

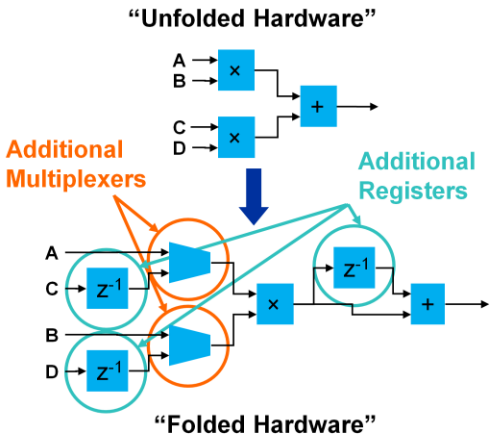



A simpler example of the fused datapath technoloy can be seen by looking at a vector dot-product. This function is fundamental in many floating point processing algorithms, such as in matrix inversion or matrix multiplication.

Note how many normalizations and denormalizations can be removed by extending the mantissa size. All multiplier products are denormalized to the largest exponent… plus some overhead depending upon adder tree size.. Then the adder tree…. sums all results. The end result is normalized. All the intermediate steps do not need the normalization and denormalization steps, as the extra mantissa bits compensates for the removed intermediate steps. For large vector sizes, the fused data path technology can reduce about ½ of the logic required in a DSP algorithm which also boosts performance as less routing is needed.

# DSP Builder Primitive Folding

- Folding
  - Also known as time-division multiplexing technique
  - Function units are underutilized when the sample rate is less than the clock rate
- Automatic time-sharing of hardware
  - Allows the user to create simple, parameterizable primitive designs without wasting resources
  - Allows for multiple channels being processed per clock cycle with DSP Builder





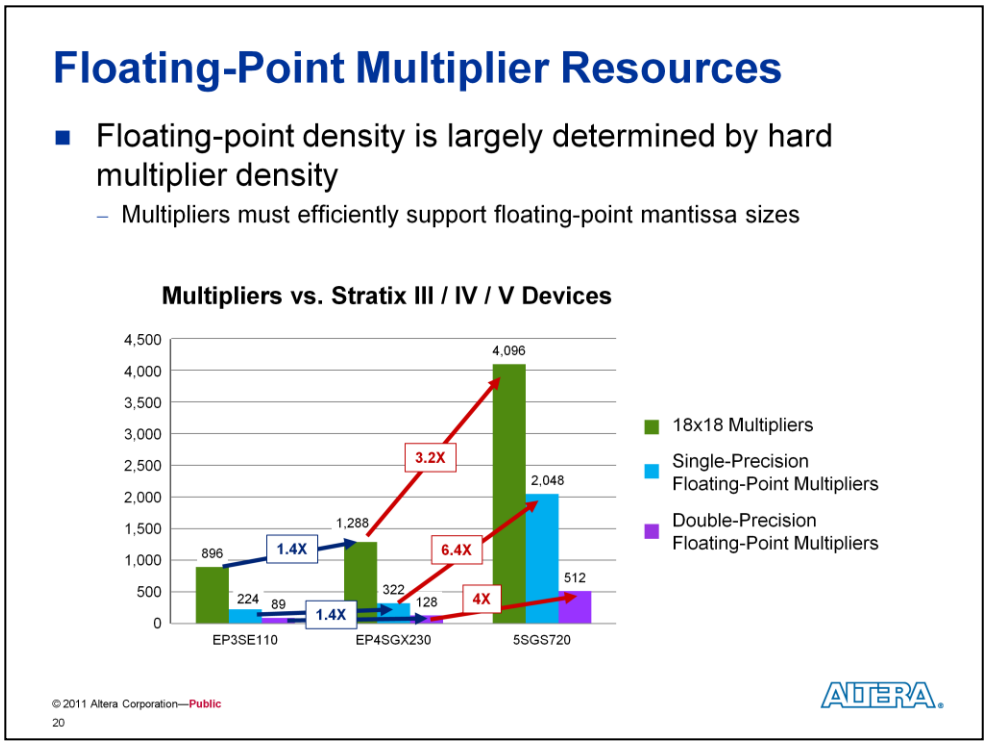


Another Innovation is ...folding.. This can be automatically implemented in the floating point flow within DSP Builder.

The purpose of this feature is automaticly provide resource savings by sharing underutilised 'functional units' (multipliers, registers, adders, memories, and so on….

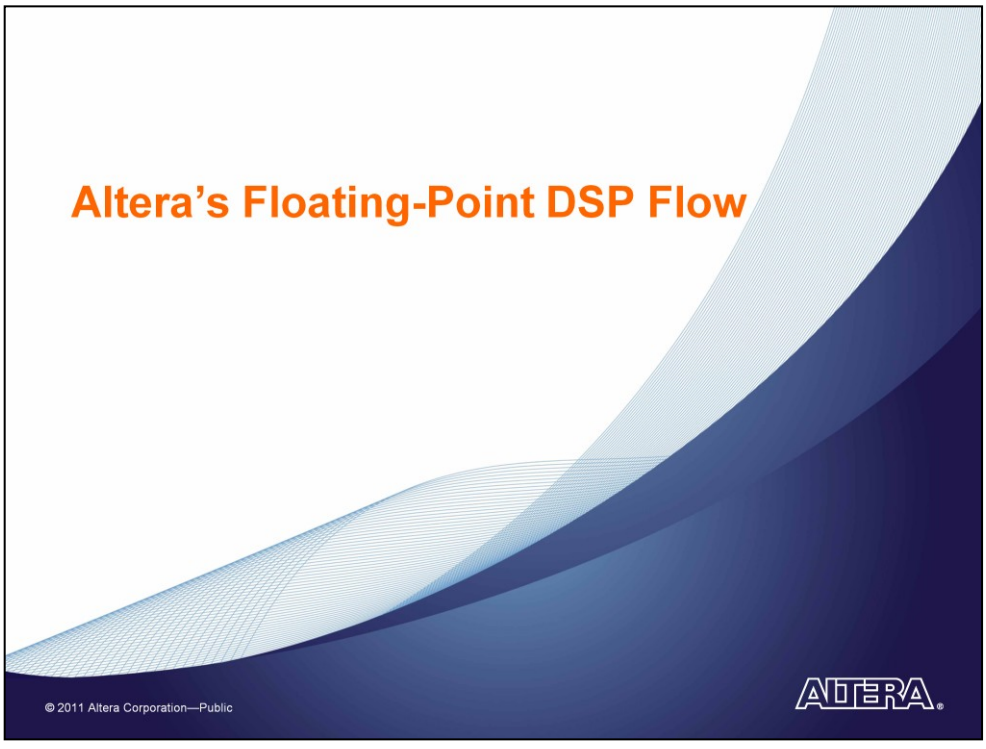
It allows the engineer to create simple, parameterisable design without wasting resources.

•Folding – or time-division multiplexing – could obviously be done by hand; but it typically would need to be redone if the parameterisation (number of channels, ratio of clock to sample rate, etc) changed.




from this chart… we can see the impact of the silicon innovations… Altera has implemented.

Altera multiplier densities have been progressively increasing with each successive family. However, note how the single precision multiplier density has increased over six times from the Stratix IV family to the Stratix V family, and offers the highest single precision FPGA multiplier density in the industry. This is due to the native 27x27 multiplier mode available in the 28nm variable precision DSP block …. as well as…. an overall increase in DSPblocks per device for 28nm fpga families.



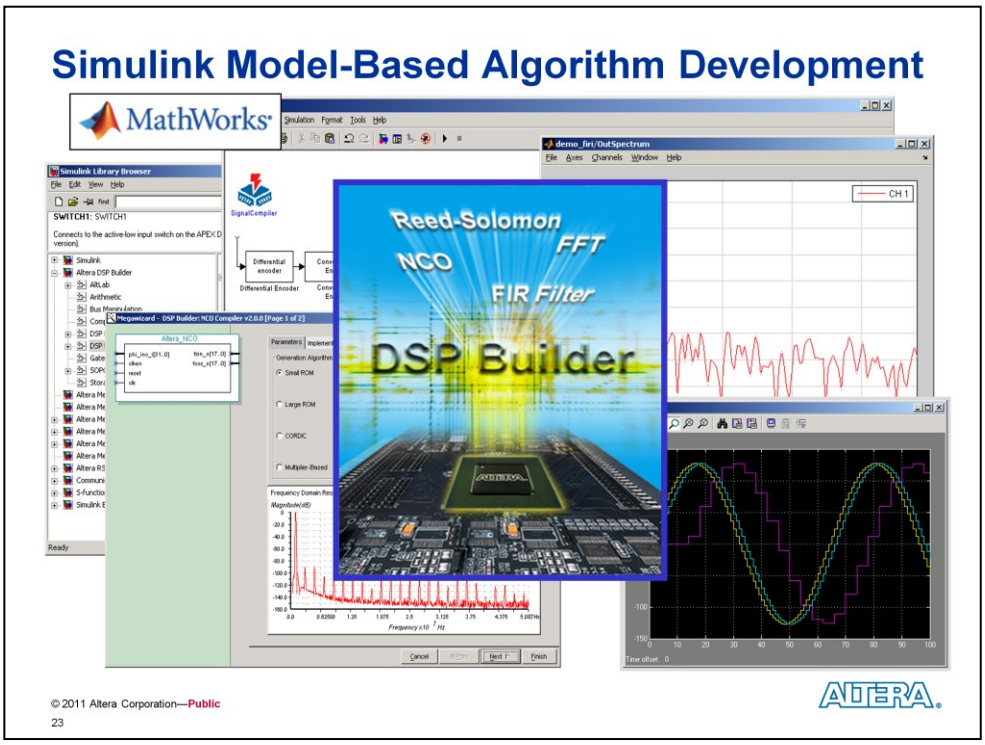

We started the webcast by looking at the IEEE floating point standard and basic arithmetic, then we looked at some options and trade-offs of implementing floating point in CPUs, DSPs, GPUs and FPGAS, and how Altera has overcome some of the challenges of implementing floating point algorithms through significant innovations. Let's now….. look at the actual too flow.

# Altera's Floating-Point Tool Flow

- Model-based tool flow
  - MATLAB, Simulink, and DSP Builder tool flow
- Two design examples
  - Motor control
  - Matrix inversion




Altera's floating point DSP tool flow is implemented using the DSP Builder Advanced Block set within the MATLAB/Simulink environment. We will show the tool flow then provide some quick examples.

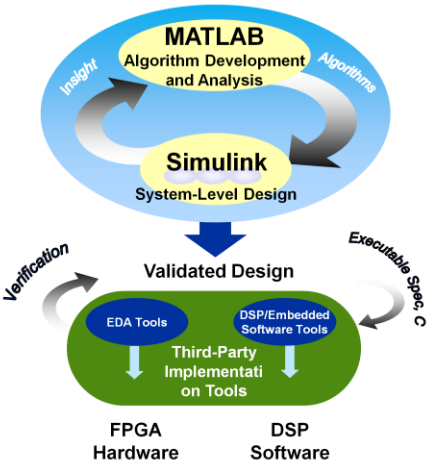



The floating point compiler using the Fused Datapath technology is embedded within Altera's DSP Builder advanced block set using a Simulink front end and design environment. Simulink is a Mathworks product, purchased separately.

## MathWorks Design Environment

- Provides top-down design
- Designs and tests system behavior early in the design process
  - Creates validated reference design
- Detects design flaws early
- Reduces design risk and cost
- Reduces time to market

Source: MathWorks




***MATLAB/Simulink Must Be Purchased Separately from MathWorks***




ALTERA.

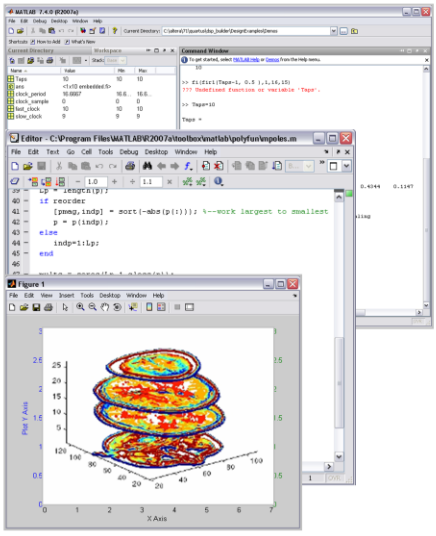
MathWorks provides a top-down solution where one can design and test the entire system modeling the DSP and control logic component interactions. The early detection of design flaws results in

reduced risk of design failure and reduced time-to-market.

It allows the designer to use the same test bench for the simulation as the final FPGA implementation.

# MathWorks – MATLAB

- Computer language of DSP engineers
- MATLAB
  - Develop algorithms and applications
  - Analyze and accesses data
  - Visualize data
  - Perform numeric calculations

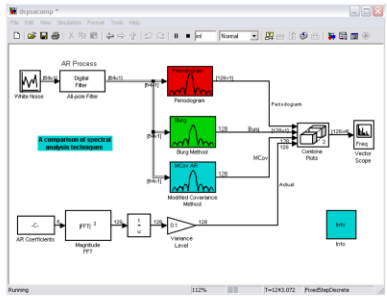


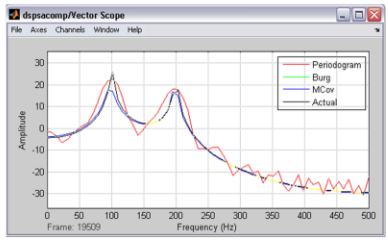



Matlab is often the starting point for system and algorithmic design. It is used to simulate and model system operation. Using DSPBuilder allows the same simulation to be used as the testbench… for the FPGA development.

# MathWorks – Simulink

- Dynamic graphical modeling environment
- Simulink
  - Develop entire system dynamically
  - Simulate and interact with the system
  - Explore different architectures
  - Analyze results
- Simulink adds the concept of cycles (clocks)
  - Becomes hardware realizable



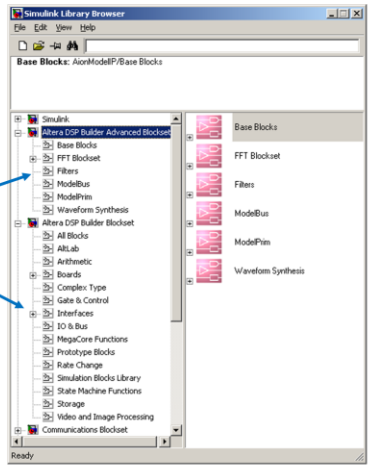




Simulink is a graphical extension to matlab allowing schematic entry and visualization. Unlike Matlab, Simulink has the concept of clock cycles, which is needed for hardware implementation. It allows the user to describe the level of parallelism in the design.

## Simulink Blockset Libraries

- Simulink
  - Sources
  - Sinks
  - Continuous
  - Discrete
  - Non-linear
  - Math
- *Altera DSP Builder Advanced Blockset*
- *Altera DSP Builder blockset*
- Fixed-point blockset
- DSP blockset
- Real-time workshop
- Communications blockset
- Image acquisition toolbox
- Others

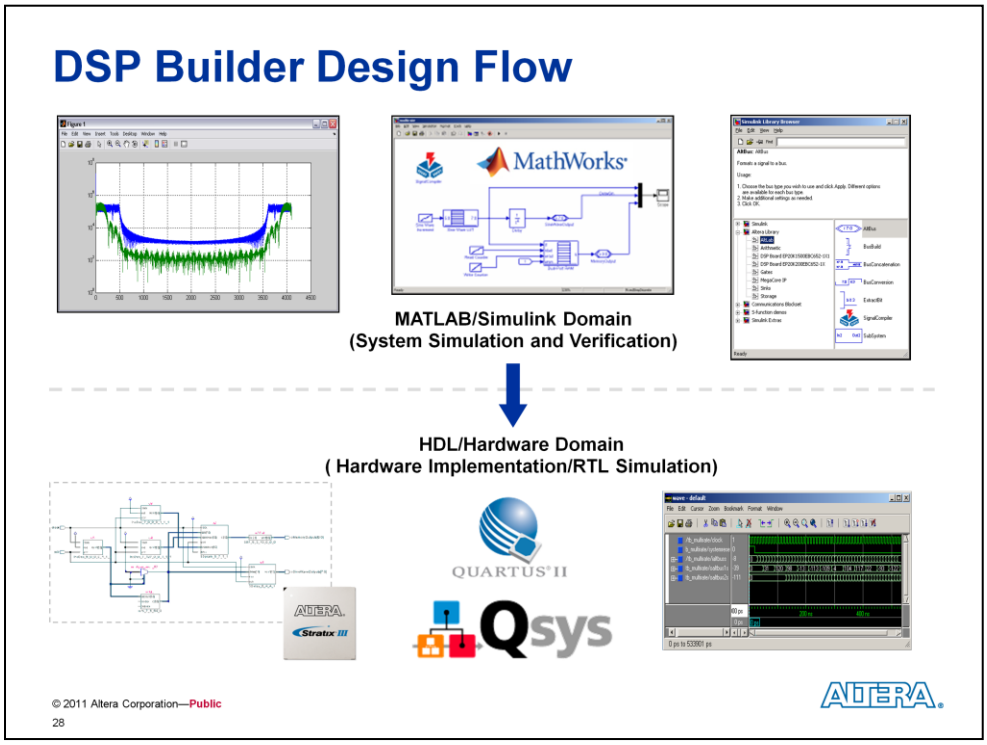






Simulink consists of a number of libraries of building blocks… from which designs can be built and tested.

DSP Builder is shipped with two library add-ons to the Matlab/simulink tools encapsulating two preferences for design methodology – a WYSIWYG-style giving explicit control over pipelining …and reset-and-enable signals in the Classic or Standard blockset, and an abstracted, higher-level style in the Advanced blockset, which allows the tool freedom for automatic optimization and pipelining.



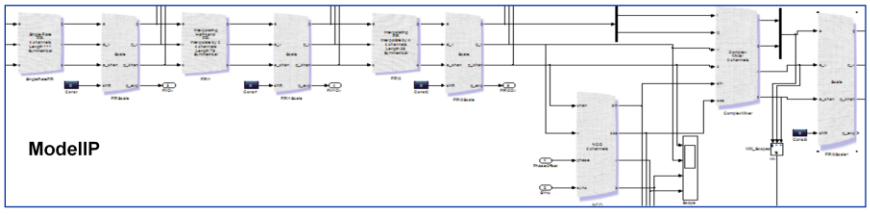

Both block-sets generate HDL, provide automatic verification… that an RTL Simulation of the HDL generated… matches the Simulink simulation, and generate Quartus projects and Qsys integration files.

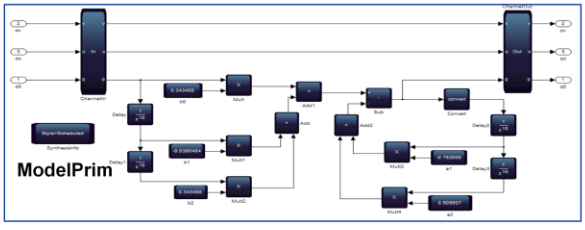
The Classic blockset also provides tools for Simulink-driven hardware simulation and SignalTap signal capture ....

# Constraint-Driven Design (1)

***1. Create Model***

- Use ModelIP or ModelPrim libraries





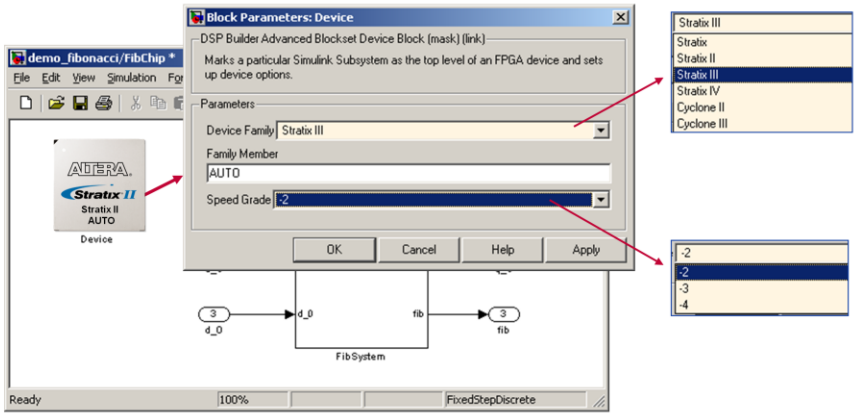

29

The design can be created using IP blocks for common functions like filters, NCOs, saturate and truncate blocks. Or the engineer can build their designs out of primitive blocks, such as multiplers, adders, muxes, delay blocks, and so forth. Or any combination – in any case, all design optimizations are applied across the designs.

# Constraint-Driven Design (2)

***2. Select Device***

- Device independent modeling up to this level





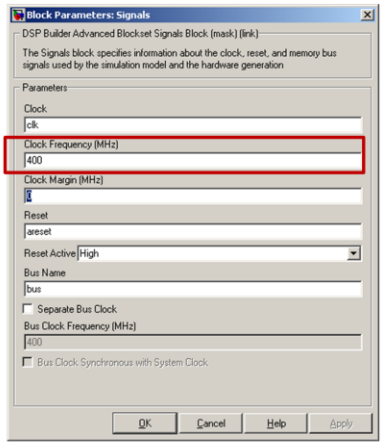


Changing FPGA device famile is a simple as selecting the family and speed grade, or even a specific device. The DSPBuilder tool has access to the timing models of each device in Quartus, which allows the tool to perform device specific optimizations. This facilitates design migration to newer Altera devices.

# Constraint-Driven Design (3)

***3. Set Frequency***

- Automatic pipelining or time sharing (ModelIP)



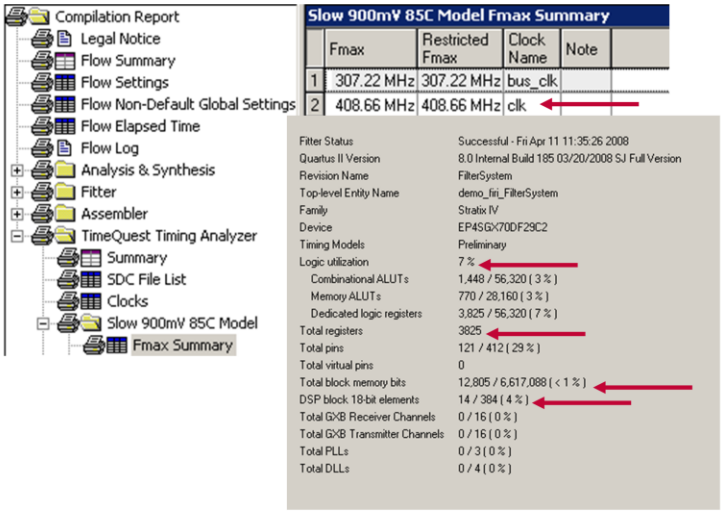




The designer specifies the desired clock frequency. The tool is capable of building designs with performance on par with the best hand-optimized HDL implementations.

# Constraint-Driven Design (4)

***4. Compile***



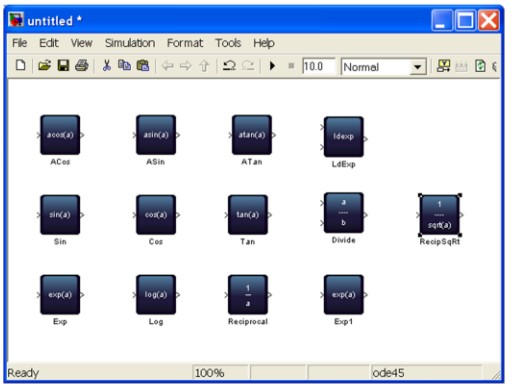





In this case, the tool closes timing with a worst case clock rate of 408 MHz, as confirmed by the Quartus II compile.

## Fused Datapath Integrated in DSP Builder (1)

- Math.h
  - SIN
  - COS
  - TAN
  - ASIN
  - ACOS
  - ATAN
  - EXP
  - LOG
  - LOG10
  - POW(x,y)
  - LDEXP
  - FLOOR
  - CEIL
  - FABS
  - SQRT
  - DIVIDE
  - 1/SQRT




***Implemented in Floating Point***



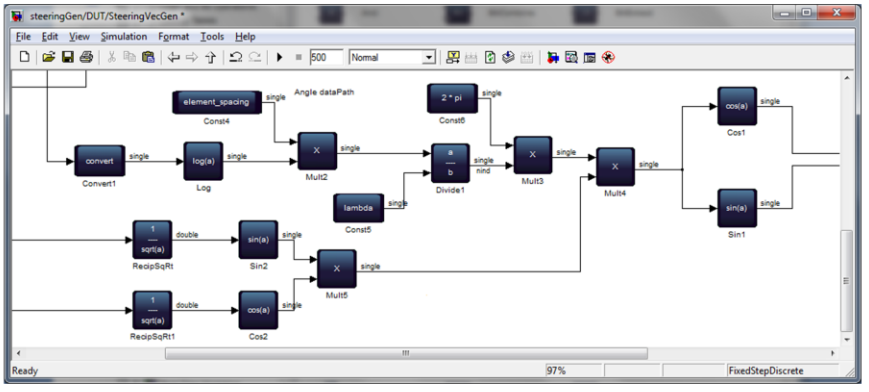

Math.h functions are the simple operations you would expect in the simple 'C' library math.h – trigonometric, log, exponent, inverse square root etc as well as basic operators such as divide.

The implementation is multiplier based, since Altera FPGAs have an abundance of high precision multipliers. Traditionally CORDIC implementations have been used for such operations – but the logic for this choice is based on the outdated ASIC idea that logic is very cheap and multipliers relatively expensive. For floating point operations on FPGA, multiplications are now cheap and have the advantages of predicable performance, low power and low latency. Contrary to accepted wisdom, the multi-stage CORDIC approach is a highly inefficient method to build these functions.

The comprehensive library support within DSPBuilder Advance Blockset allows customers to build large, complex and highly optimized floating point datapaths.

# Fused Datapath Integrated in DSP Builder (2)

- Complex functions easily realized in FPGA



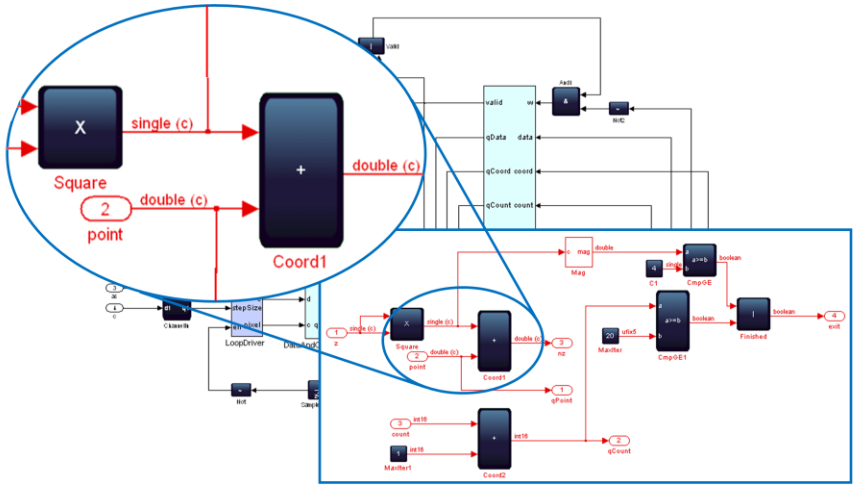




Fused Datapath has been integrated into Altera's DSPBuilder Advanced Blockset toolflow. Complex equations can be easily implemented in DSPBuilder blockset within Mathworks's Simulink environment. This allows for easy fixed and floating point simulation as well as FPGA implementation of engineer designs.

# Supports Single and Double Precision

- Including complex (c)



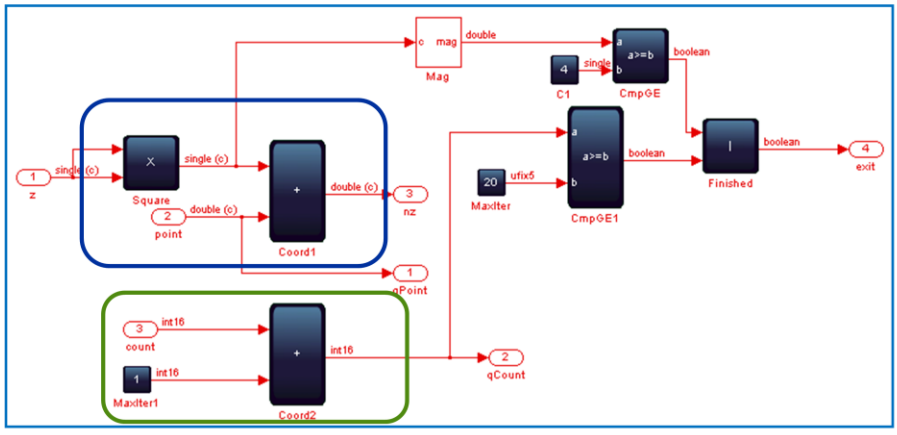





DSPBuilder supports complex numbers, in fixed point or single or double precision. The variable types are inherited, unless overridden by the engineer.

# Supports Fixed Point and Floating Point

- Within the same model

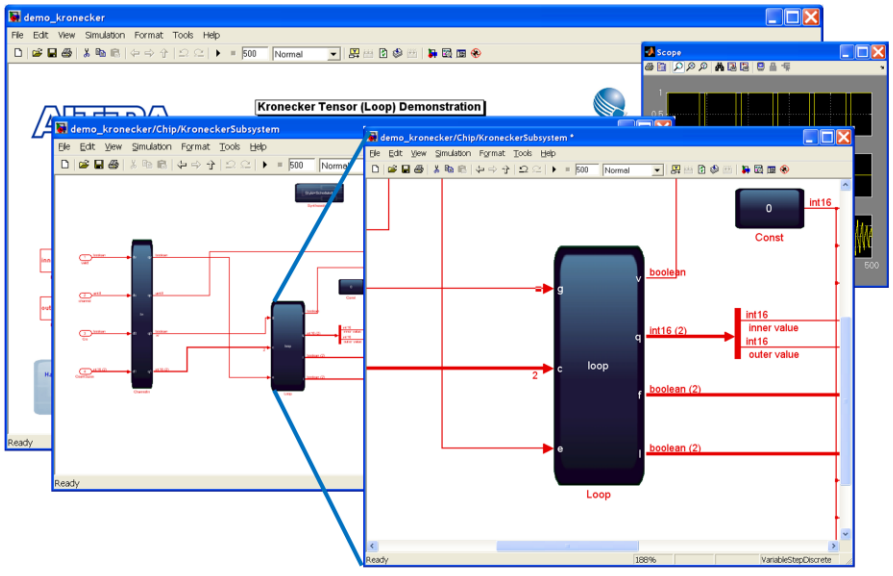






If we look under the subsystem doing the simple math… we note how floating point complex types – both single and double – are used in conjunction with fixed point types. The tool then offers a single environment for building mixed floating and fixed point designs, and offers abstraction of complex and vectors making design simple and productive. All the issues of dealing with mantissa, exponents, normalizations and special conditions are abstracted away, just as when using a floating point software flow.

# Loop Block for “For i” Loops

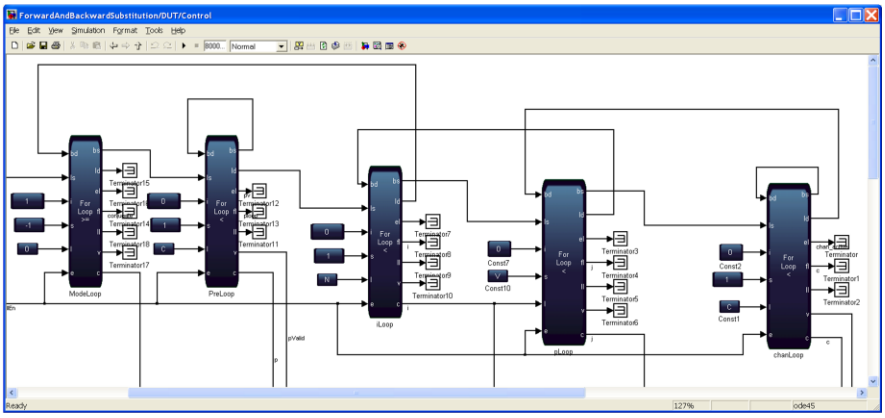





37

For While looping blocks are provided to allow for control functions in DSP Builder.

# Nested Looping Functionality

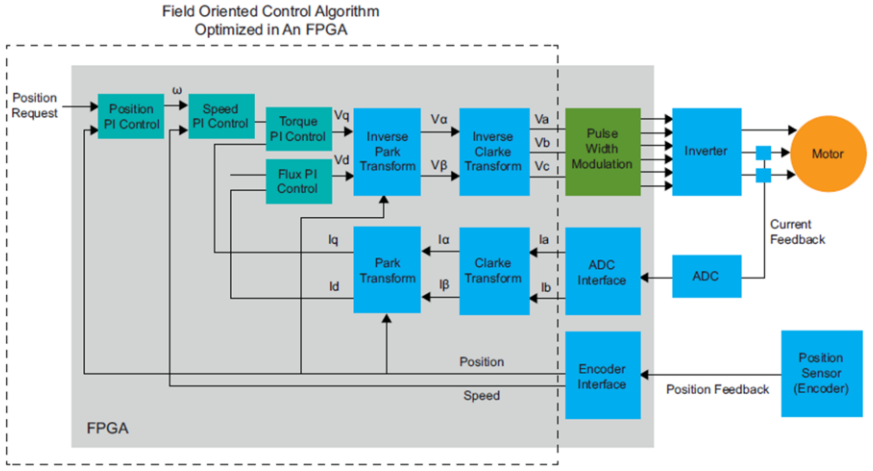






These loop blocks can be nested to provide for complex looping behaviour, similar to software constructs.

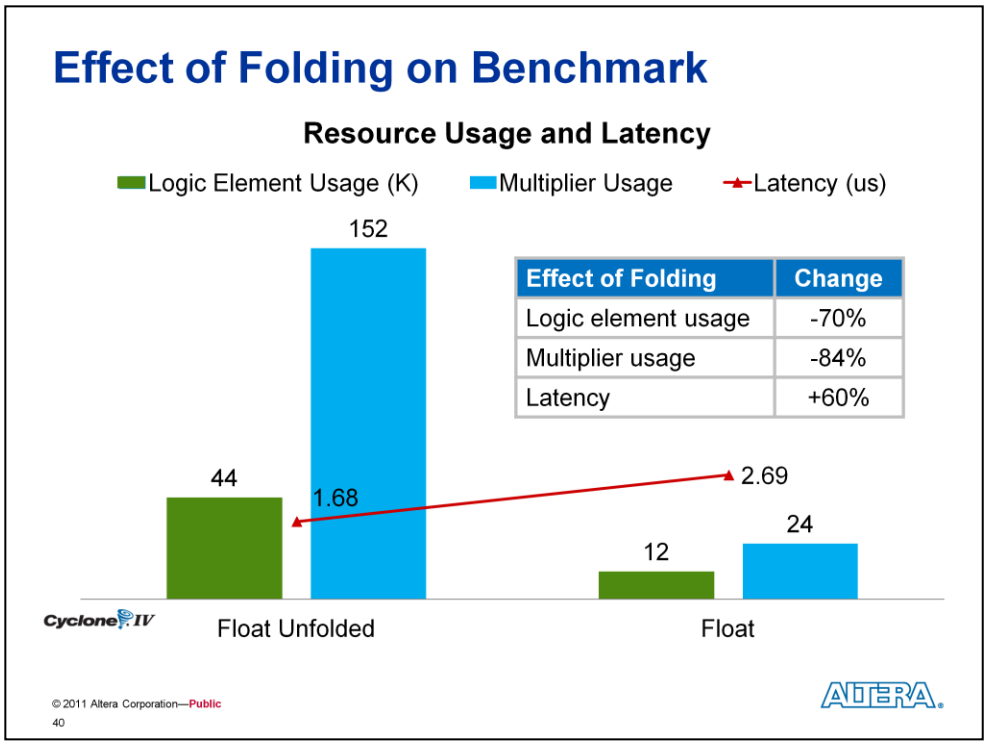
# Example: Motor Control Design







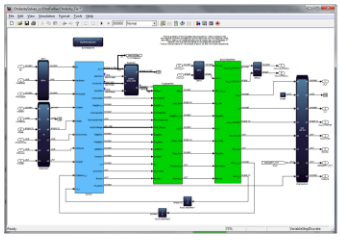
Now let's take a look at two examples, Motor Control design and a matrix inversion function. Here we have a motor control design. If we look at a field orientated control algorithm optimized in an FPGA, you can see the FGPA can manage nearly the whole system. Due to the wide dynamic range needed in the design, single precision floating point was used. Let's look at the advantages folding brought to resource utilization.

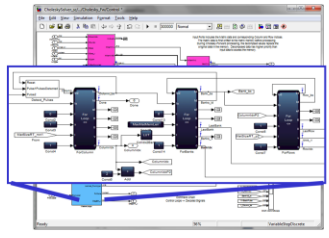



This slide shows the significant impact that the "automatic folding" capability within Altea's DSP Builder Advanced Block Set can have on a design. The motor control design was implemented in floating point with and without folding. With folding… the logic usage dropped by 70% and multiplier usage dropped by 84%. As with any time division multiplexing design, the latency also increased. The details of this design can be found in an Altera white paper titled "Optimize Motor Control Designs with an Integrated FPGA Design Flow "

# Matrix Inversion Benchmark

- Using Stratix IV and Arria II GX FPGAs
- Build the largest matrix inversion possible, scale down, and show tool capability
- Use Altera's floating-point tool flow
  - MATLAB, Simulink, DSP Builder v11.0
    - Includes the floating-point compiler with fused datapath technology









The highly parameterized single precision matrix inversion example, uses a Cholesky decomposition, with forward and back substitution. The goal was to determine the large matrix inversion possible and the performance available in Altera' s Stratix IV and Arria II GX FGPAs. The example used readily available off the shelf tools and was benchmarked by BDTI to determine performance, efficiency and ease-of-use.

# Matrix Inversion Throughput

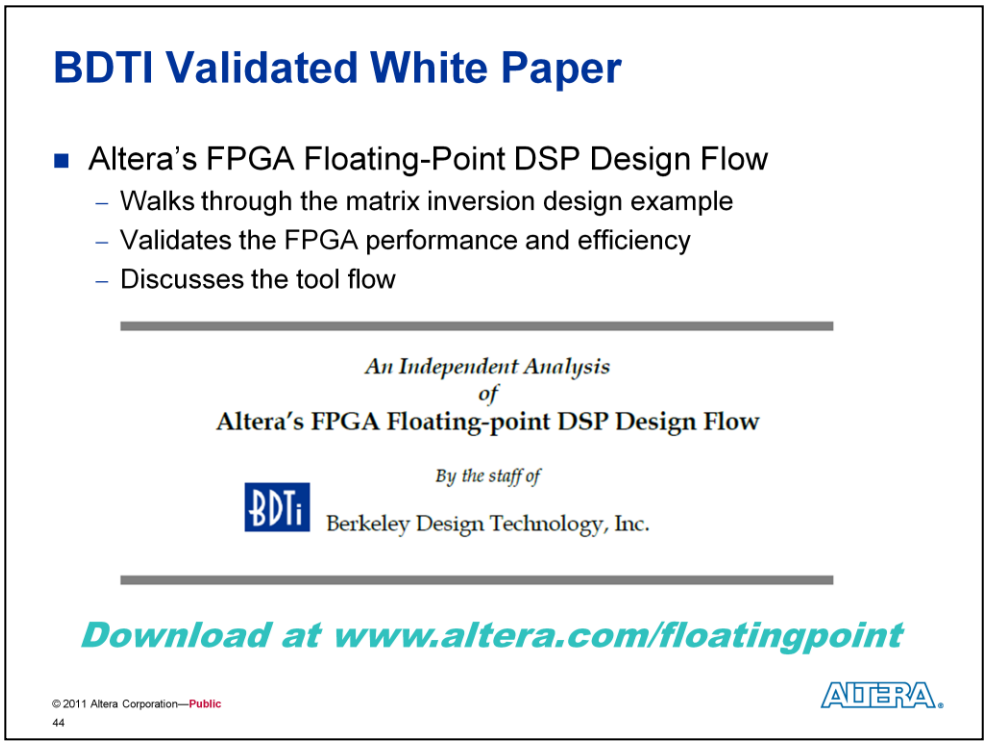
- Matrix inversion – single precision
  - Cholesky decomposition, with forward and back substitution
  - Stratix IV FPGA (vector size = 60)
  - Arria II GX FPGA (vector size = 30)
- Performance of the Cholesky solver

| Configuration (Matrix size / Vector size) | Throughput Reported by Simulink (Matrices/sec) | Throughput Reported by ModelSim (Matrices/sec) | Vector Multiplier Utilization | |
|---|---|---|---|---|
| | | | Reported by Simulink | Reported by ModelSim |
| 240x240 / 60 | 3,204 | 3,204 | 88% | 88% |
| 180x180 / 60 | 6,113 | 6,113 | 78% | 78% |
| 120x120 / 60 | 12,680 | 12,680 | 58% | 58% |
| 60x60 / 60 | 28,998 | 28,998 | 27% | 27% |
| 120x120 / 30 | 9,921 | 9,921 | 69% | 69% |
| 60x60 / 30 | 27,886 | 27,886 | 33% | 33% |
| 30x30 / 30 | 59,665 | 59,665 | 14% | 14% |

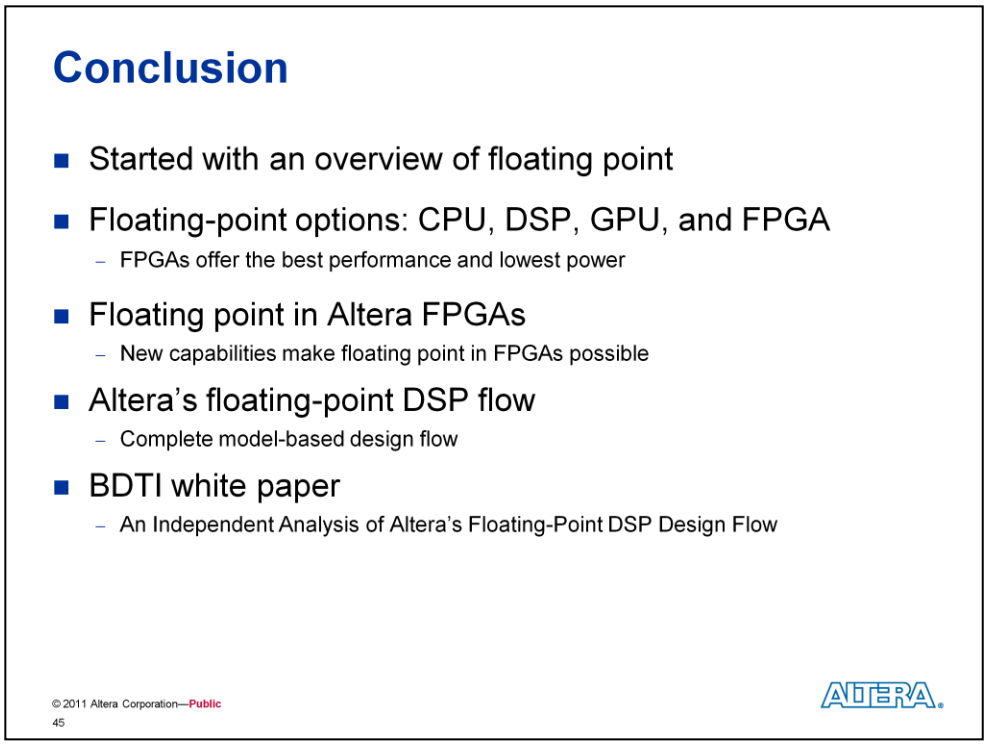



This slides highlight the results of the example using the Stratix IV and Arria II GX device family. Stratix IV could achieve a large matrix size of 240x240 with a vector size of 60. The sustained throughput was over 3000 matrices per second. Arria II also had strong performance. The details of the matrix inversion example are available in BDTIs White paper titled “An independent analysis of Altera’s FPGA floating  point DSP design flow” available on www.altera.com/floatingpoint

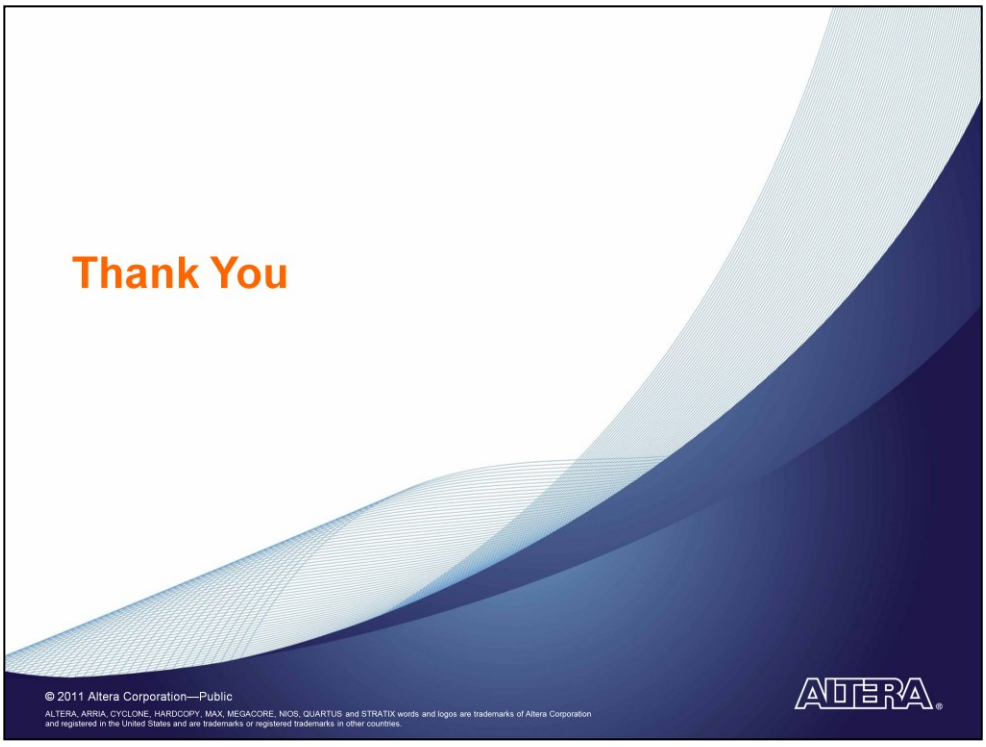
# BDTI White Paper




BDTI's White Paper walks through matrix inversion design example and validates the FPGA performance and efficiency. They also discuss the tool flow. Again, you can download the white paper at www.altera.com/ floating point




First, Thank you for sticking around to the end of the webcast. I simply wanted to re-cap all that we touched on. We started with an overview of floating point, then discussed the different floating point options, then discussed how Altera's significant innovations in DSP Builder led to high-performance and efficient floating flow. We then walked through the flow and showed a couple examples: Motor Control and matrix inversion. For more information, please see the BDTI white paper and online training …Altera offers for floating point. To get started with DSP Builder, please contact your local sale rep for an evaluation license.


Thank You
© 2011 Altera Corporation—Public
ALTERA, ARRIA, CYCLONE, HARDCOPY, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the United States and are trademarks or registered trademarks in other countries.
ALTERA