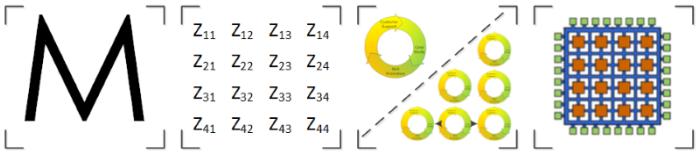
Optimized for Vivado HLS

MatrixCore™

Developed by X-Star Team

A Pioneer in HLS

- Numerical Linear Algebra (NLA)
  - What is NLA
  - Ten Surprises from NLA
- MatrixCore™ Technology
  - Existing Solutions
  - Why MatrixCore
  - MatrixCore Functions
  - Deliverables
- Application Engineering
  - Solving Ax=b
  - Benchmark



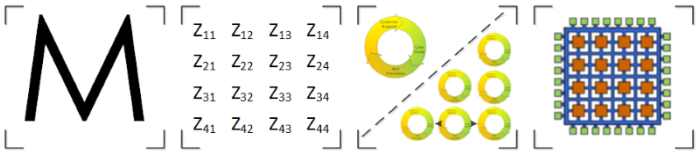
Optimized for Vivado HLS
MatrixCore™

Numerical Linear Algebra (NLA)

What is NLA

Numerical linear algebra is the study of algorithms for performing linear algebra computations, most notably matrix operations, on computers.

It is often a fundamental part of engineering and computational science problems, such as image and signal processing, telecommunication, computational finance, materials science simulations, structural biology, data mining, bioinformatics, fluid dynamics, and many other areas.

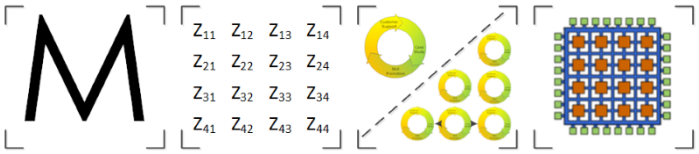


Optimized for Vivado HLS
MatrixCore™

Numerical Linear Algebra (NLA)

What is NLA

Modern software relies heavily on the development, analysis, and implementation of state-of-the-art algorithms for solving various numerical linear algebra problems, in large part because of the role of matrices in finite difference and finite element methods.



Optimized for Vivado HLS
MatrixCore™

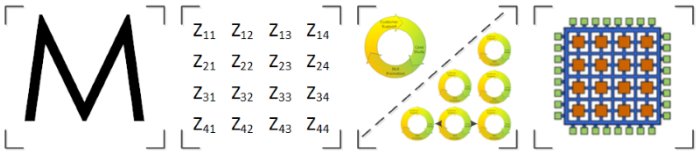
Numerical Linear Algebra (NLA)

Ten Surprises from NLA

Here are ten things about numerical linear algebra that you may find surprising if you're not familiar with the field.

1. Numerical linear algebra applies very advanced mathematics to solve problems that can be stated with high school mathematics.

2. Practical applications often require solving enormous systems of equations, millions or even billions of variables.



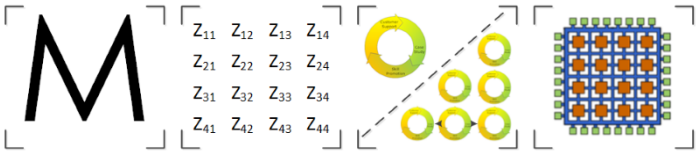
Optimized for Vivado HLS
MatrixCore™

Numerical Linear Algebra (NLA)

Ten Surprises from NLA

3. The heart of Google is an enormous linear algebra problem. PageRank is essentially an eigenvalue problem.

4. The efficiency of solving very large systems of equations has benefited at least as much from advances in algorithms as from Moore's law.



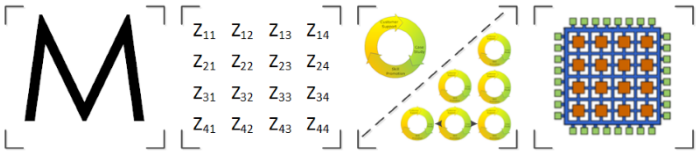
Optimized for Vivado HLS
MatrixCore™

Numerical Linear Algebra (NLA)

Ten Surprises from NLA

5. Many practical problems - optimization, differential equations, signal processing, etc. - boil down to solving linear systems, even when the original problems are non-linear. Finite element software, for example, spends nearly all its time solving linear equations.

6. A system of a million equations can sometimes be solved on an ordinary PC in under a millisecond, depending on the structure of the equations.



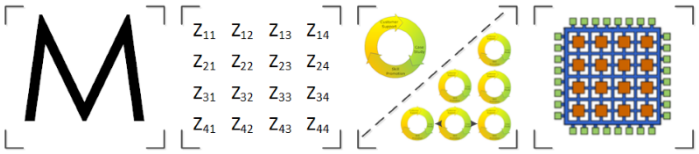
Optimized for Vivado HLS
MatrixCore™

Numerical Linear Algebra (NLA)

Ten Surprises from NLA

7. Iterative methods, methods that in theory require an infinite number of steps to solve a problem, are often faster and more accurate than direct methods, methods that in theory produce an exact answer in a finite number of steps.

8. There are many theorems bounding the error in solutions produced on real computers. That is, the theorems don't just bound the error from hypothetical calculations carried out in exact arithmetic but bound the error from arithmetic as carried out in floating point arithmetic on computer hardware.



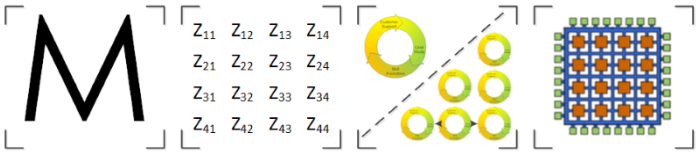
Optimized for Vivado HLS
MatrixCore™

Numerical Linear Algebra (NLA)

Ten Surprises from NLA

9. It is hardly ever necessary to compute the inverse of a matrix.

0. There is remarkably mature software for numerical linear algebra. Brilliant people have worked on this software for many years.



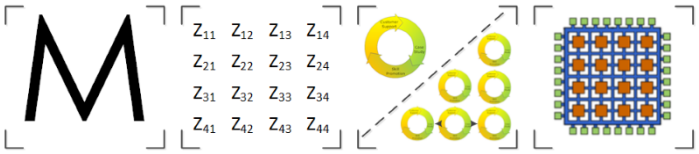
Optimized for Vivado HLS
MatrixCore™

MatrixCore™ Technology

Existing Solutions

There's dozens of commercial and open-source linear algebra libraries which are developed with numerous language (C/C++, Java, Matlab, Fortran, even scripting language) targeting different platforms, such as EP, CPU, DSP, etc. The platforms are essentially based on instruction-by-instruction running, even with multi-threading and multi-core, the processing capability is somewhat unbearable when tackling with large and ultra-large dimension matrix.

The only way to accelerate the processing power is to use pure hardware chips, such as ASICs or FPGAs.



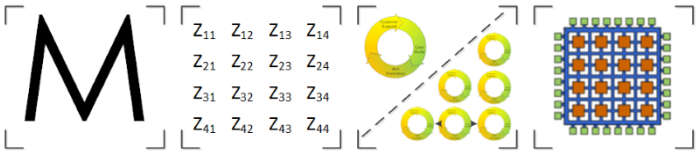
Optimized for Vivado HLS
MatrixCore™

MatrixCore™ Technology

Existing Solutions

The natural parallel architecture and fine granularity endows a much faster processing speed and more flexible ways by fully or partially pipelining the inter-calculations of matrix operations.

Historically, there's there are few Verilog/VHDL commercial or open-source libraries for matrix operations. And a well-architected Verilog/VHDL code is specially designed for particular ASIC/FPGA architectures. A high performance library for Xilinx platform might meet performance bottleneck on Altera platform, since they have different fabric architecture, and RTL synthesis tool might not so



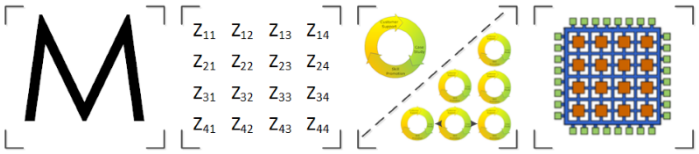
Optimized for Vivado HLS
MatrixCore™

MatrixCore™ Technology

Existing Solutions

intelligent to optimize a particular code for all platforms. Things get even worth when doing trade-offs between resource and performance.

Besides the coding style, a well-designed RTL code has pre-defined micro-architecture which contains data flow and control flow. A particular micro-architecture corresponding to a fixed resource occupation, clock speed, processing capabilities, etc. So when doing trade-offs between different micro-architectures, one have to redesign the total, not just change some parameters or code sections, since the total operation scheduling has to be changed.

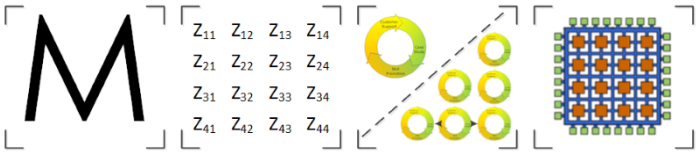


Optimized for Vivado HLS
MatrixCore™

MatrixCore™ Technology

Existing Solutions

Moreover, using low level design language (Verilog/VHDL) to model the high level algorithm (matrix calculation) is extremely time consuming, error-prone, and a nightmare for developers, especially for float-point data type.



Optimized for Vivado HLS
MatrixCore™

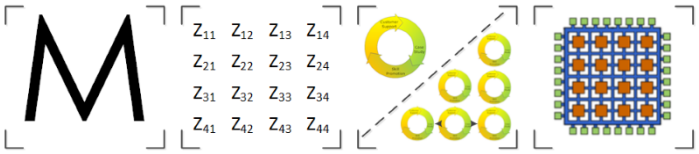
MatrixCore™ Technology

Why MatrixCore

One thing is about to change everything…

With introduction of High Level Synthesis (HLS) technology, ASIC/FPGA developers will get so many benefits that algorithm modeling is based on C/C++/SystemC, which is much more efficient than Verilog/VHDL.

The most important thing is HLS gives opportunities to implement different architecture targeting different performance or resource with one golden C/C++/SystemC source.



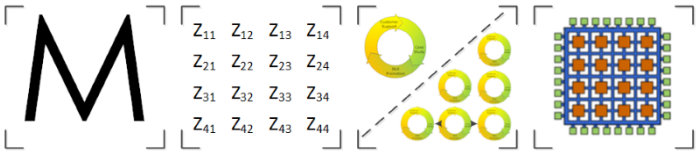
Optimized for Vivado HLS
MatrixCore™

MatrixCore™ Technology

Why MatrixCore

Only different compiler directives are needed to fulfil the target goal, which is much simpler than modifying the code.

The MatrixCore library is built on this technology. It's created by C++ with template parameters which allows users to define arbitrary data types, arbitrary matrix dimensions, etc. The library is specially optimized for Vivado HLS tool from Xilinx, and it's easy to port to other HLS tools, such as Synphony C Compiler, Catapult, CyberWorkBench, etc.



Optimized for Vivado HLS
MatrixCore™

MatrixCore™ Technology

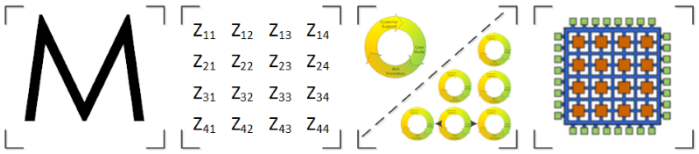
Why MatrixCore

The MatrixCore library has features:

1. Platform-independent, which can be targeted different ASIC/FPGA platforms.

2. Specially optimized for Vivado HLS targeting Xilinx FPGA platforms.

3. C++ template supporting arbitrary data types: arbitrary precision fixed-point, single and double precision float-point.



Optimized for Vivado HLS
MatrixCore™

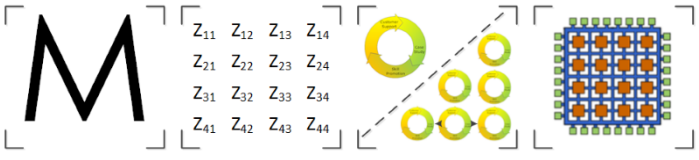
MatrixCore™ Technology

Why MatrixCore

4. Pre-embedded optimization directives to generate least area, high performance, and low power implementations.

5. Low performance mode (fully rolled architecture), which uses least resource.

5. High performance mode (function level pipelined architecture), which is a trade-off between resource-minimization and performance-maximization.

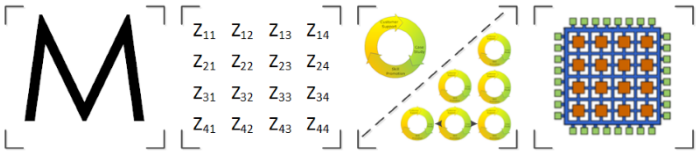


Optimized for Vivado HLS
MatrixCore™

MatrixCore™ Technology

Why MatrixCore

5. Ultra performance mode (function + operator level pipelined architecture) has highest processing power.



Optimized for Vivado HLS
MatrixCore™

MatrixCore™ Technology

MatrixCore Functions

[Prototype]

matrix_real_inv<T, N, K>(i_x[N][N], o_z[N][N])

matrix_cplx_inv<T, N, K>(i_x[N][N], o_z[N][N])

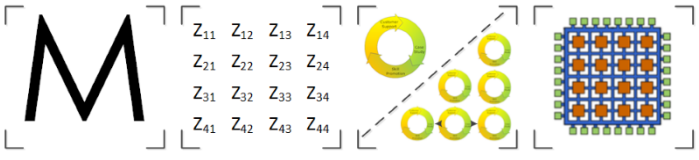
[Argument]

T : “float” or “double”.

N : matrix dimension (N > 1)

K : Error control factor (K > 1)

A higher K means less calculation error, compared with MATLAB standard function “inv”.



Optimized for Vivado HLS
MatrixCore™

MatrixCore™ Technology

MatrixCore Functions

[Pre-defined Mode]

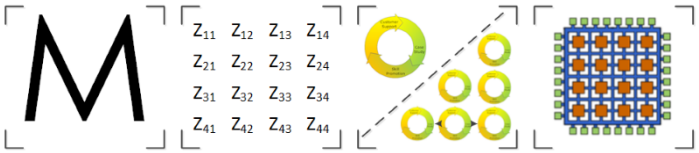
MATRIXCORE_REAL_INV_LOW_PERFORMANCE

MATRIXCORE_REAL_INV_HIGH_PERFORMANCE

MATRIXCORE_REAL_INV_ULTRA_PERFORMANCE

[Description]

Real or complex matrix inversion.

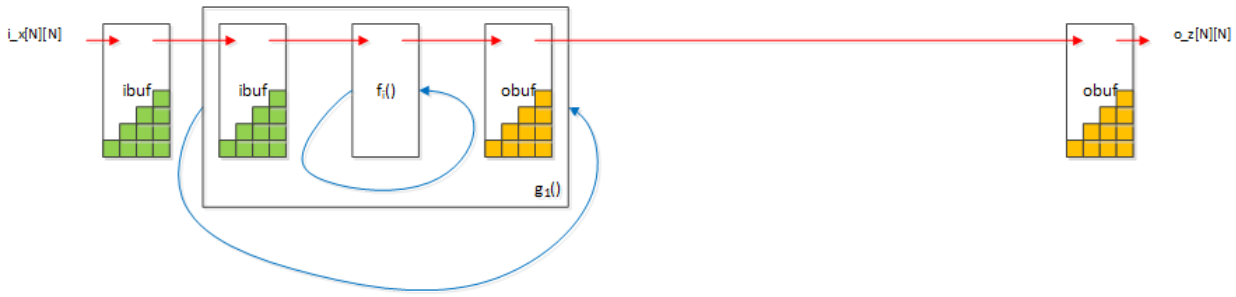


Optimized for Vivado HLS
MatrixCore™

MatrixCore™ Technology

MatrixCore Functions

Low performance mode (fully rolled architecture)



Outer-space function : $g_j()$ iterative

Inner-space function : $f_i()$ iterative

PROs : Most resource saving

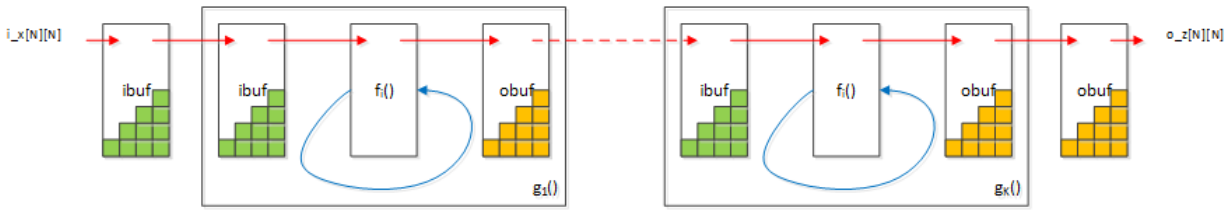
CONs : Least processing power

Optimized for Vivado HLS
MatrixCore™

MatrixCore™ Technology

MatrixCore Functions

Low performance mode (fully rolled architecture)

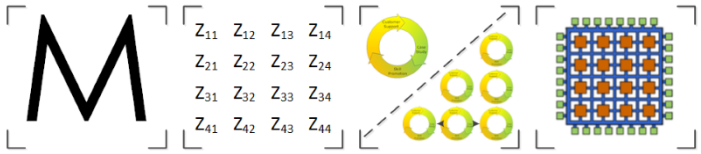


Outer-space function : $g_j()$ pipelined

Inner-space function : $f_i()$ iterative

PROs : Mid-averaged processing power

CONs : Mid-averaged resource cost

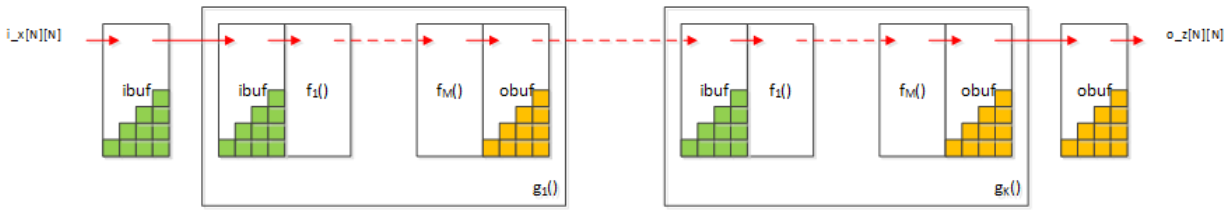


Optimized for Vivado HLS
MatrixCore™

MatrixCore™ Technology

MatrixCore Functions

Low performance mode (fully rolled architecture)

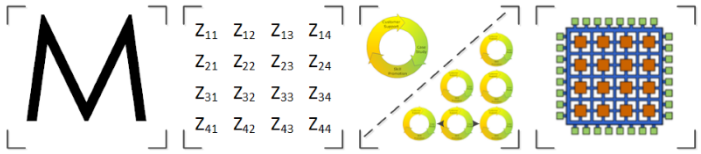


Outer-space function : $g_j()$ pipelined

Inner-space function : $f_i()$ pipelined

PROs : Most processing power

CONs : Most resource cost



Optimized for Vivado HLS
MatrixCore™

MatrixCore™ Technology

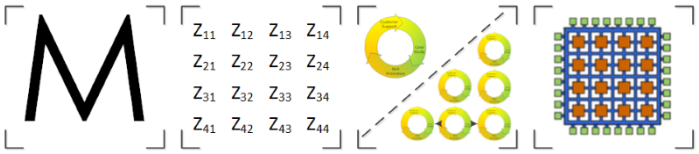
MatrixCore Functions

[Feature]

1. Supporting ill-conditioned matrix, even LU, QR, SVD, Cholesky decomposition method cannot work.

2. Supporting different trade-off architectures:
   fully rolled (low performance mode)
   partially pipelined (high performance mode)
   fully pipelined (ultra performance mode)

3. Supporting arbitrary error level tuning (compared with MATLAB inv() function).



Optimized for Vivado HLS
MatrixCore™

MatrixCore™ Technology

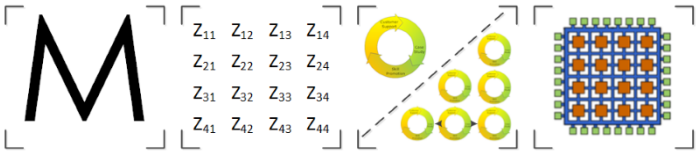
MatrixCore Functions

[Feature]

4. Supporting arbitrary matrix dimension.

5. Supporting single or double precision float-point data type.

6. Supporting real or complex data type.



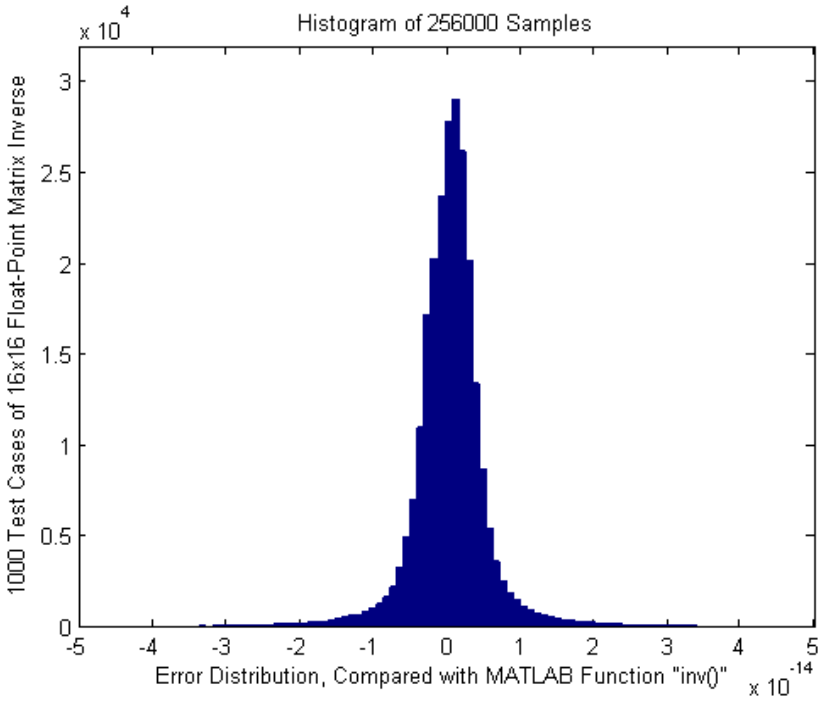
Optimized for Vivado HLS
MatrixCore™

MatrixCore™ Technology

MatrixCore Functions

[Error Analysis]

1,000 test cases of randomized 16x16 float-point matrix inverse compared with MATLAB function “inv()”. 256,000 total samples are compared.



Maximum Negative Error

-5.9237e-10

Maximum Positive Error

+1.4677e-11

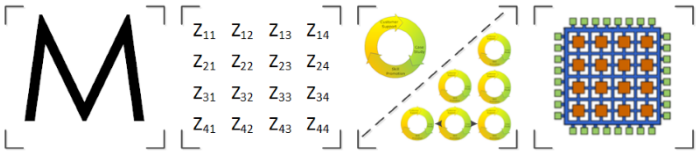
Optimized for Vivado HLS
MatrixCore™

MatrixCore™ Technology

MatrixCore Functions

[Benchmark] [Real Matrix Inverse]

Single Precision Float Dimension = 16x16 Low Performance	Implementation VC707(board)	Single Precision Float Dimension = 16x16 High Performance	Implementation VC707(board)	Single Precision Float Dimension = 16x16 Ultra Performance	Implementation VC707(board)
Clock F_{MAX} (MHz)	≈250	Clock F_{MAX} (MHz)	≈250	Clock F_{MAX} (MHz)	≈250
Throughput (tCLK)	≈453,000	Throughput (tCLK)	≈30,000	Throughput (tCLK)	≈4,300
LUT6	≈7,000	LUT6	≈90,000	LUT6	≈180,000
FF	≈9,000	FF	≈140,000	FF	≈300,000
DSP48	≈25	DSP48	≈270	DSP48	≈570
BRAM36	≈10	BRAM36	≈170	BRAM36	≈590
Dynamic Power (mW)	≈	Dynamic Power (mW)	≈	Dynamic Power (mW)	≈



Optimized for Vivado HLS
MatrixCore™

MatrixCore™ Technology

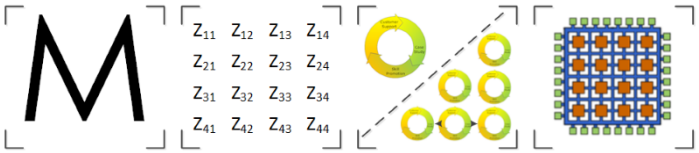
Deliverables

[C++ Code]

C++ template technique supports platform-independent portability, postpone secondary development and re-optimization.

C++ source code contains all algorithm-level details, compiler-oriented optimization directives, and constraints, which is a best way for engineers to learn advanced algorithm and HLS-oriented C++ coding style optimization strategy.

This version has the most flexible implementation since it's platform-independent.



Optimized for Vivado HLS
MatrixCore™

MatrixCore™ Technology

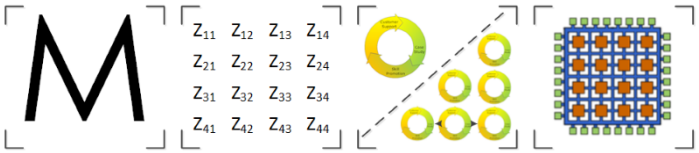
Deliverables

[RTL Code]

RTL source code is delivered as plain-text or encrypted version. The former has RTL-level implementation details, but the latter has everything hidden.

The encrypted version has two deliveries. One is bonded to a particular HOSTID that only authorized PC can compile the code, the other delivery is free to use.

RTL code is generated by HLS tool, which has something to do with platform. The flexibility is somewhat decrease.



Optimized for Vivado HLS
MatrixCore™

MatrixCore™ Technology

Deliverables

[EDIF Netlist]

EDIF netlist is targeting at a specific platform, and all optimizations (logic or physical optimization) is based on details of this platform.

This version has the least flexible implementation since it's platform-dependent.