AMD Vitis™ Functional Simulation

AMD
together we advance_

Agenda

- **What is Functional Simulation?**
- Introduction to AMD Vitis™ Functional Simulation (VFS)
- Vitis Array (varray)
- VFS in MATLAB® / Python™ Environment

What is Functional Simulation?

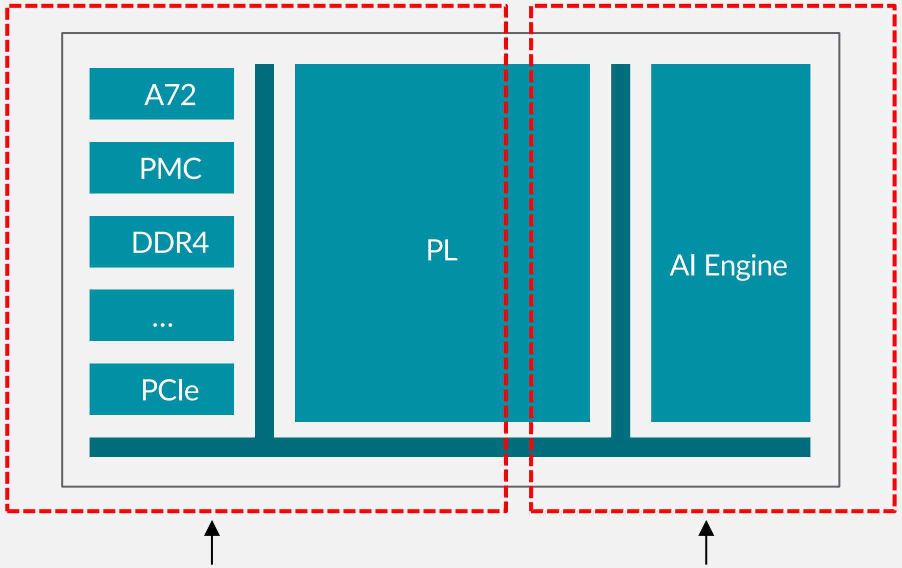
Functional Simulation

Test and verify the functional correctness of the design

- Generate test vectors and simulate the behavior
- Bit-accurate simulation
- Use x86 simulator for AI Engine component
- Use C-Sim for HLS (PL) component
- Custom testbench is required

Programmable Logic (PL) FPGA Logic	**AMD Versal™ AI Engine Array** AIE Function
Simulated using C-Sim (if written in Vitis HLS)	Simulated using x86 simulator

AMD Versal™ Adaptive SoC



Platform
Developed by the HW Team
Using AMD Vivado™ Tool

AIE+PL Subsystem
Developed by the
Algorithm Team
Using AMD Vitis™ Tool

Heterogeneous Design with Versal AI Engine and PL

Simulation in High-Level Languages

Why?

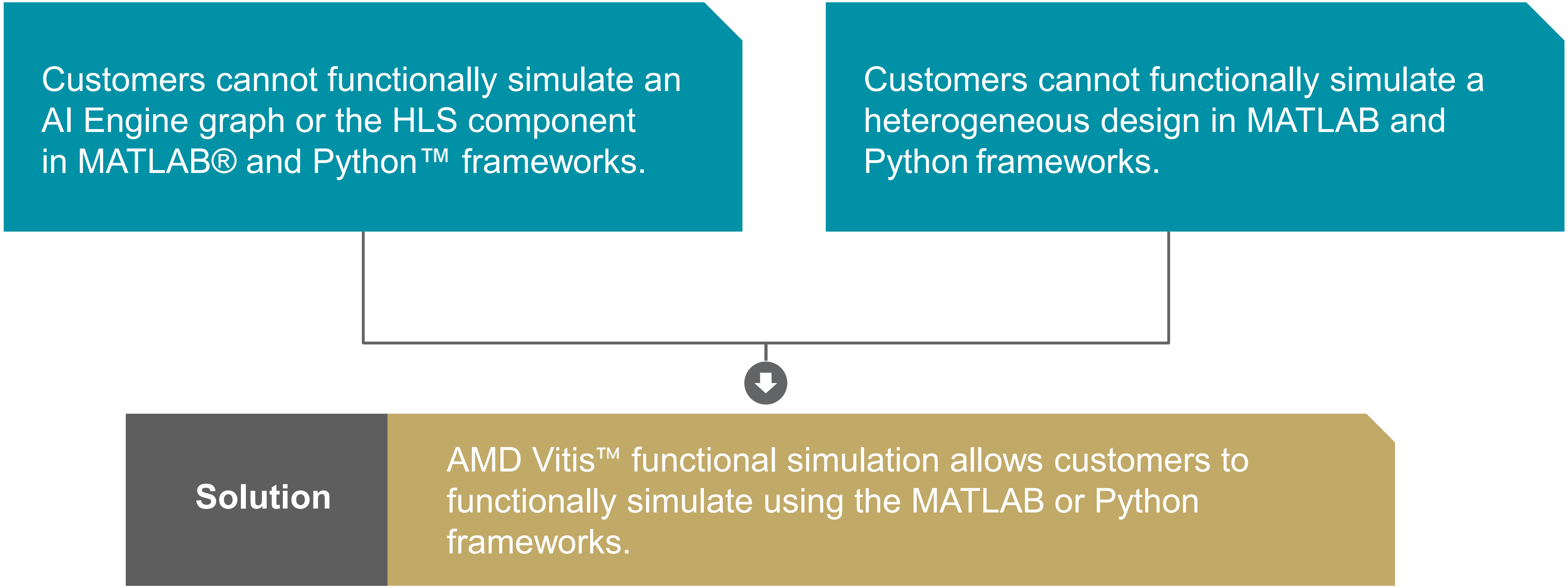
Simulation of the algorithmic part of a design in high-level languages is critical in shortening development time and accelerating time to market.

What?

AMD Versal™ devices introduce AI Engines in addition to programmable logic and a simulation solution should seamlessly incorporate both compute domains.

The simulation should be done in the customer’s preferred language.

Challenges



Customers cannot functionally simulate an AI Engine graph or the HLS component in MATLAB® and Python™ frameworks.

Customers cannot functionally simulate a heterogeneous design in MATLAB and Python frameworks.

Solution

AMD Vitis™ functional simulation allows customers to functionally simulate using the MATLAB or Python frameworks.

Agenda

- What is Functional Simulation?
- **Introduction to AMD Vitis™ Functional Simulation (VFS)**
- Vitis Array (varray)
- VFS in MATLAB® / Python™ Environment

Introduction to AMD Vitis™ Functional Simulation (VFS)

Heterogeneous design with AMD Versal™ AI Engine and HLS (targeting the PL)

Enables functional verification of:

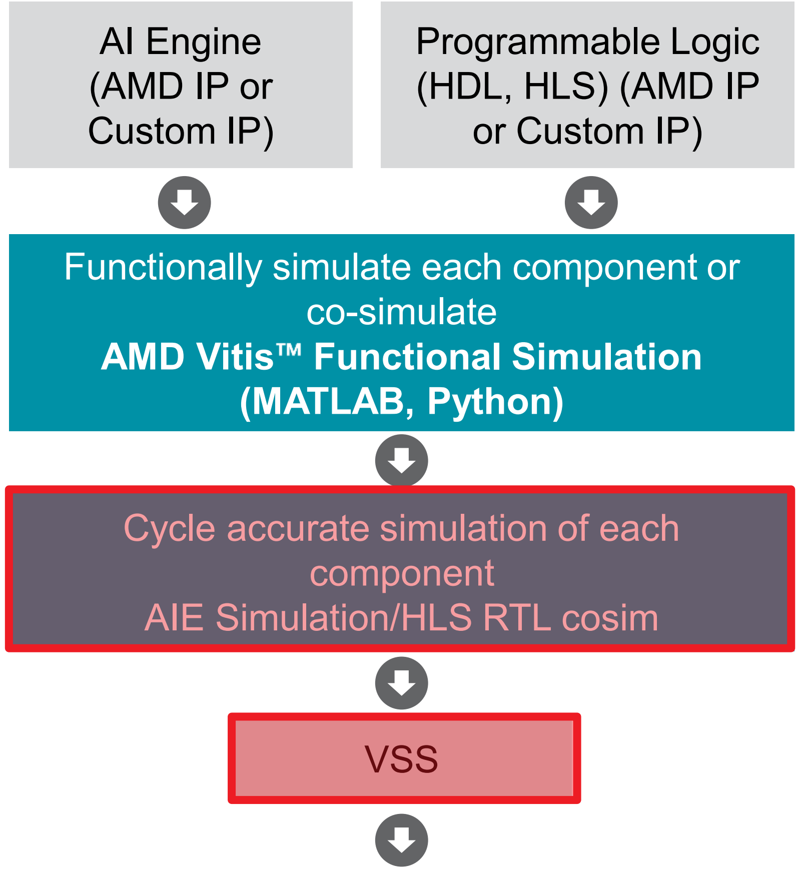
- AI Engine graph natively in the MATLAB® or Python™ framework
- HLS kernel natively in the MATLAB or Python framework
- AIE-PL (HLS) subsystem natively in the MATLAB or Python framework

MATLAB python

Maintain original environment – minimizes changes during verification

Support heterogeneous systems

Bit-accurate functional simulation



AI Engine (AMD IP or Custom IP)

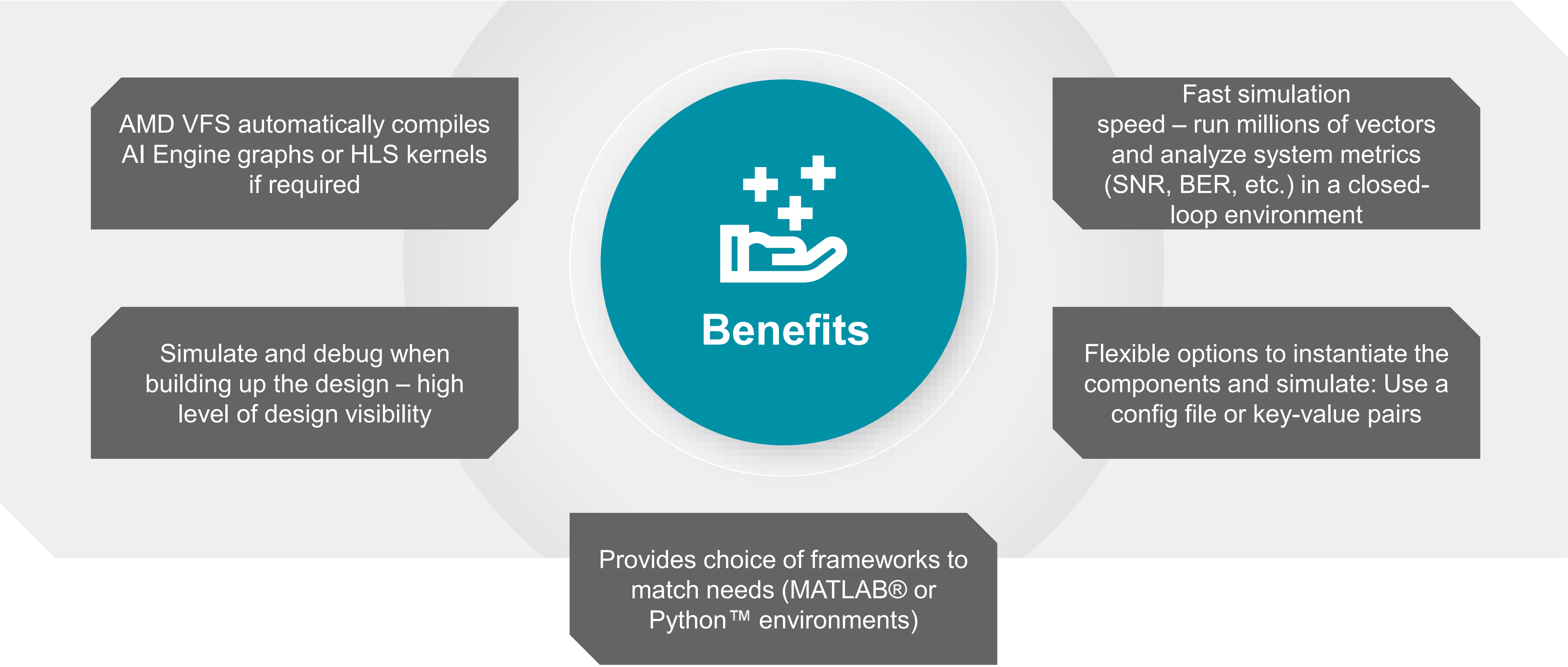
Programmable Logic (HDL, HLS) (AMD IP or Custom IP)

Functionally simulate each component or co-simulate **AMD Vitis™ Functional Simulation (MATLAB, Python)**

Cycle accurate simulation of each component AIE Simulation/HLS RTL cosim

VSS

Benefits of AMD Vitis™ Functional Simulation



Benefits

AMD VFS automatically compiles AI Engine graphs or HLS kernels if required

Simulate and debug when building up the design – high level of design visibility

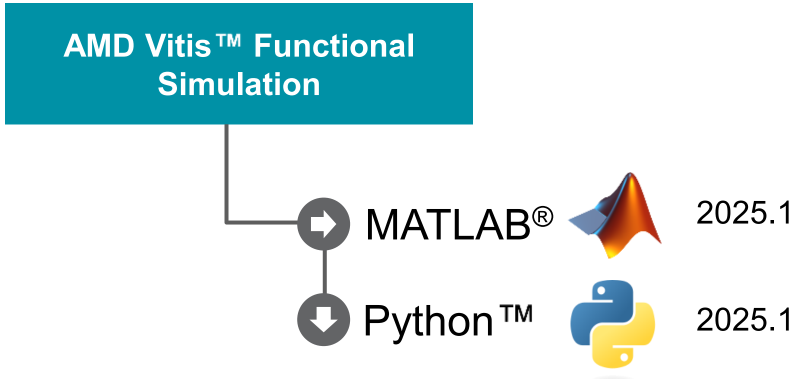
Fast simulation speed – run millions of vectors and analyze system metrics (SNR, BER, etc.) in a closed-loop environment

Flexible options to instantiate the components and simulate: Use a config file or key-value pairs

Provides choice of frameworks to match needs (MATLAB® or Python™ environments)

Current Tool Constraints (as of 2025.1)

- No cycle count information – cannot measure latency and throughput
- AMD Versal™ AI Engine memory and stream access conflicts are not modeled
- Some programming model constructs not fully supported



AMD Vitis™ Functional Simulation

MATLAB® 2025.1

Python™ 2025.1

Embedded Design Development Using Vitis User Guide (UG1701)

Agenda

- What is Functional Simulation?
- Introduction to AMD Vitis™ Functional Simulation (VFS)
- **Vitis Array (varray)**
- VFS in MATLAB® / Python Environment™

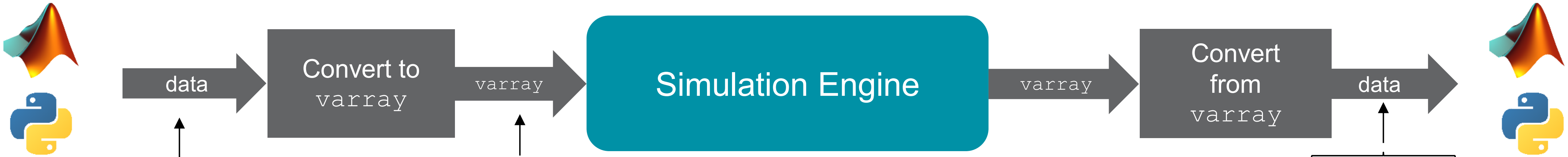
AMD VFS and AMD Vitis™ Array (varray)

AMD Vitis™ Array (varray)

- A module to support all data types for AMD devices in MATLAB® / Python™
- Allows conversion and casting between the data types

The simulation engine needs to support all data types supported by AMD devices

Not all AMD supported data types are natively supported in MATLAB / Python

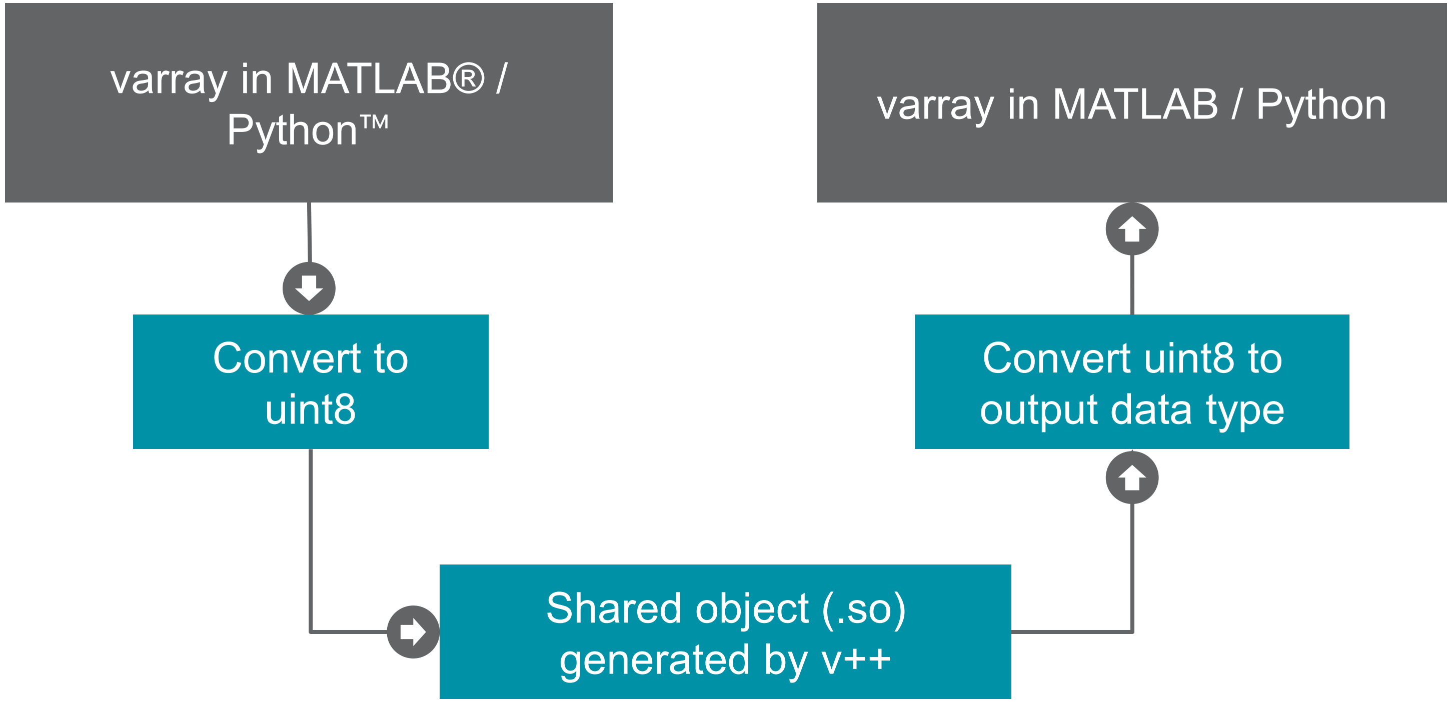


MATLAB array
Python Numpy array or lists/tuples

Vitis Array supports all AMD data types

MATLAB array
Python Numpy array or lists/tuples

Varray and Supported Data Types

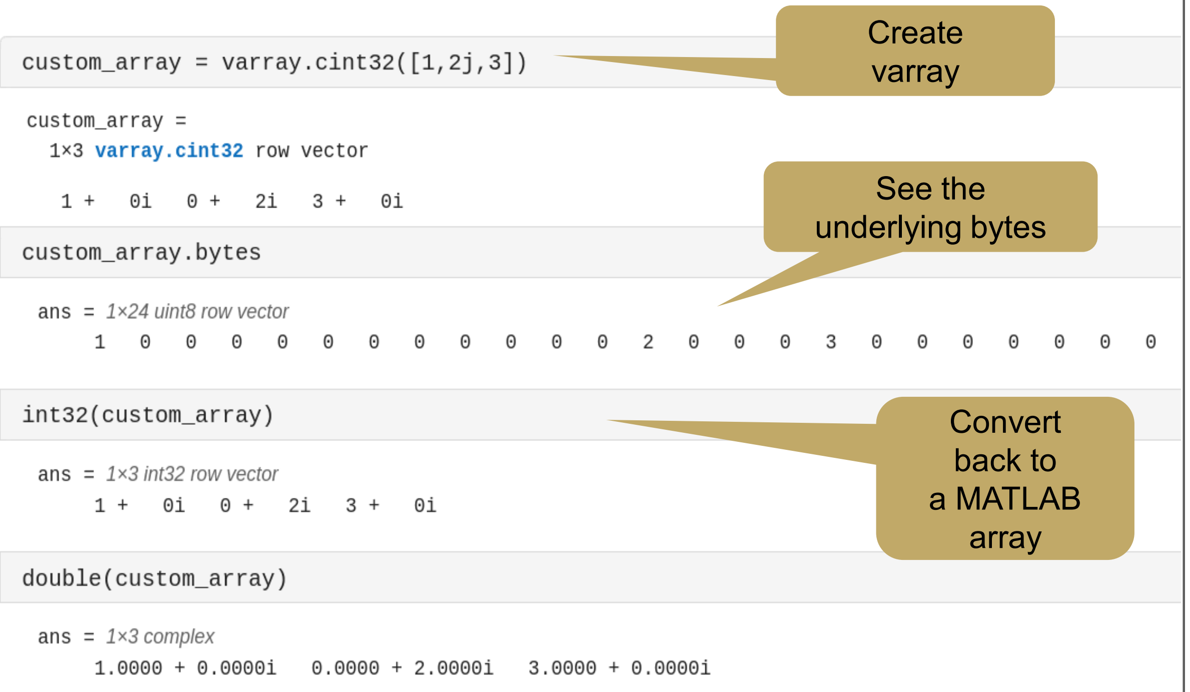

varray in MATLAB® / Python™
Convert to uint8
Shared object (.so) generated by v++
Convert uint8 to output data type
varray in MATLAB / Python

- int8, uint8, cint8, cuint8
- int16, uint16, cint16, cuint16
- int64, uint64, cint64, cuint64
- float, cfloat
- double, cdouble
- float8, bfloat8 (2025.2)
- float16 (AIE-ML v2)
- Bfloat16, cbfloat16 (AIE-ML)
- mx9,mx6,mx4 (AIE-ML v2)[1]
- uint4,int4 (AIE-ML v2)
- fi(sign, width, fractional)[2,3]

1 - You can specify a rounding mode for mx data types.
2 - For fi, Overflow mode: *Saturate*, Rounding mode: *Nearest*
3 - In HLS, we specify the bit width and the integer bits
ap_fixed<16,16> is equivalent to fi(1,16,0)

Varray in MATLAB® / Python™ Environment

MATLAB



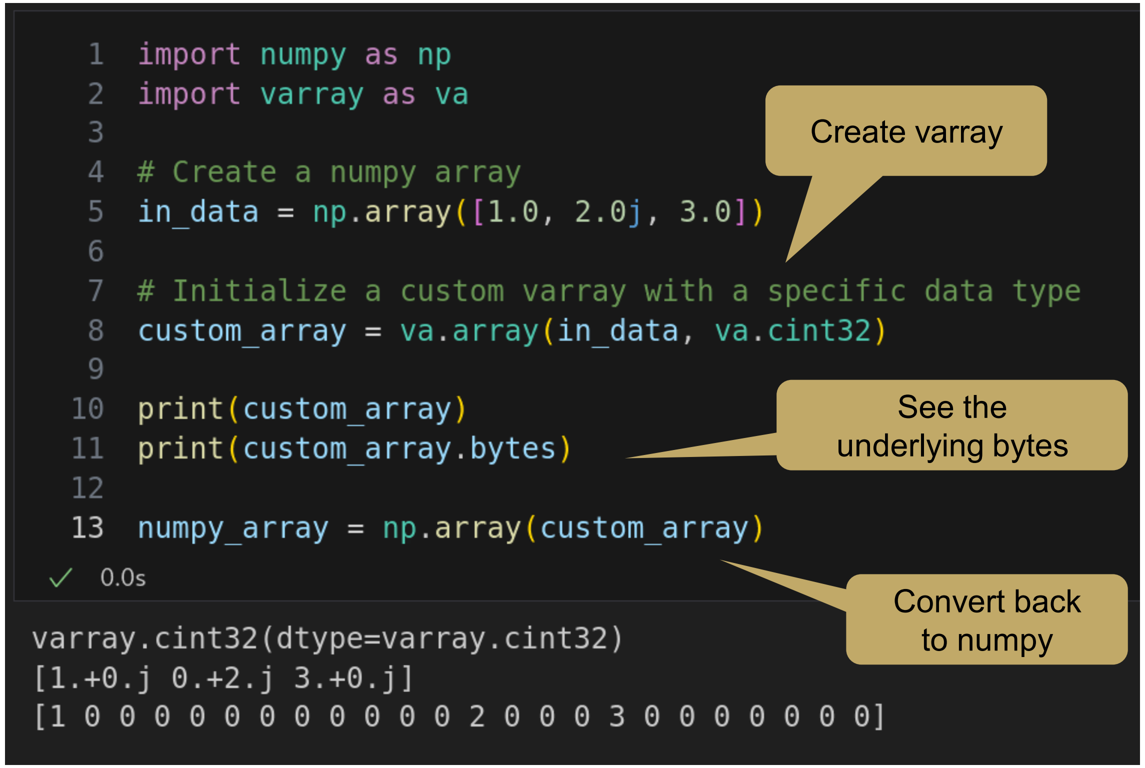
```
custom_array = varray.cint32([1,2j,3])

 custom_array =
   1×3 varray.cint32 row vector

     1 +   0i    0 +   2i    3 +   0i

custom_array.bytes

  ans = 1×24 uint8 row vector
       1    0    0    0    0    0    0    0    0    0    0    0    2    0    0    0    3    0    0    0    0    0    0    0

int32(custom_array)

  ans = 1×3 int32 row vector
       1 +   0i    0 +   2i    3 +   0i

double(custom_array)

  ans = 1×3 complex
       1.0000 + 0.0000i   0.0000 + 2.0000i   3.0000 + 0.0000i
```

Create varray

See the underlying bytes

Convert back to a MATLAB array

PYTHON



```python
import numpy as np
import varray as va

# Create a numpy array
in_data = np.array([1.0, 2.0j, 3.0])

# Initialize a custom varray with a specific data type
custom_array = va.array(in_data, va.cint32)

print(custom_array)
print(custom_array.bytes)

numpy_array = np.array(custom_array)
```

✓ 0.0s

```
varray.cint32(dtype=varray.cint32)
[1.+0.j 0.+2.j 3.+0.j]
[1 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 3 0 0 0 0 0 0 0]
```

Create varray

See the underlying bytes

Convert back to numpy

Agenda

- What is Functional Simulation?
- Introduction to AMD Vitis™ Functional Simulation (VFS)
- Vitis Array (varray)
- **VFS in MATLAB® / Python™ Environment**

AMD VFS Initialization Methods

Using a Configuration File

- Specify a .cfg file with build details
- VFS checks for availability of build; updates

```
aie_graph =
vfs.aieGraph(config_file=‘<path_to_cfg_file>’)
hls_kernel =
vfs.hlsKernel(config_file=‘<path_to_cfg_file>’)
```

Using Key-Value Pairs

- Specify simulation parameters as key-value pairs
- VFS generates .cfg file and checks for build

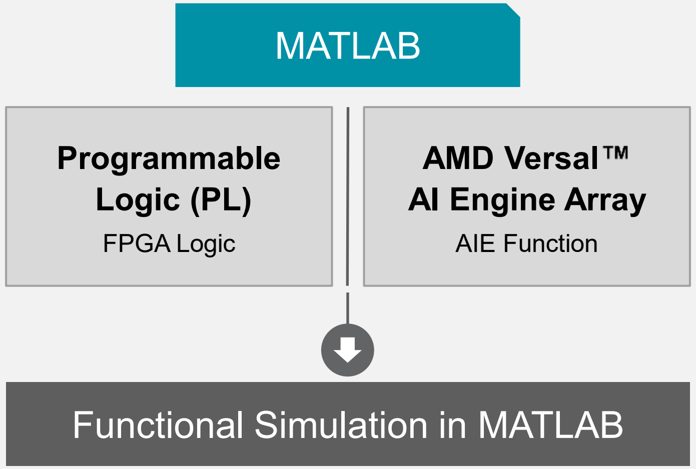
```
aie_graph = vfs.aieGraph(input_file: ‘…’, part:
‘…’,                        include_paths:
[‘…’])
hls_kernel = vfs.hlsKernel(part: ‘…’,
hls_function: ‘…’,
          input_files: [‘…’])
```

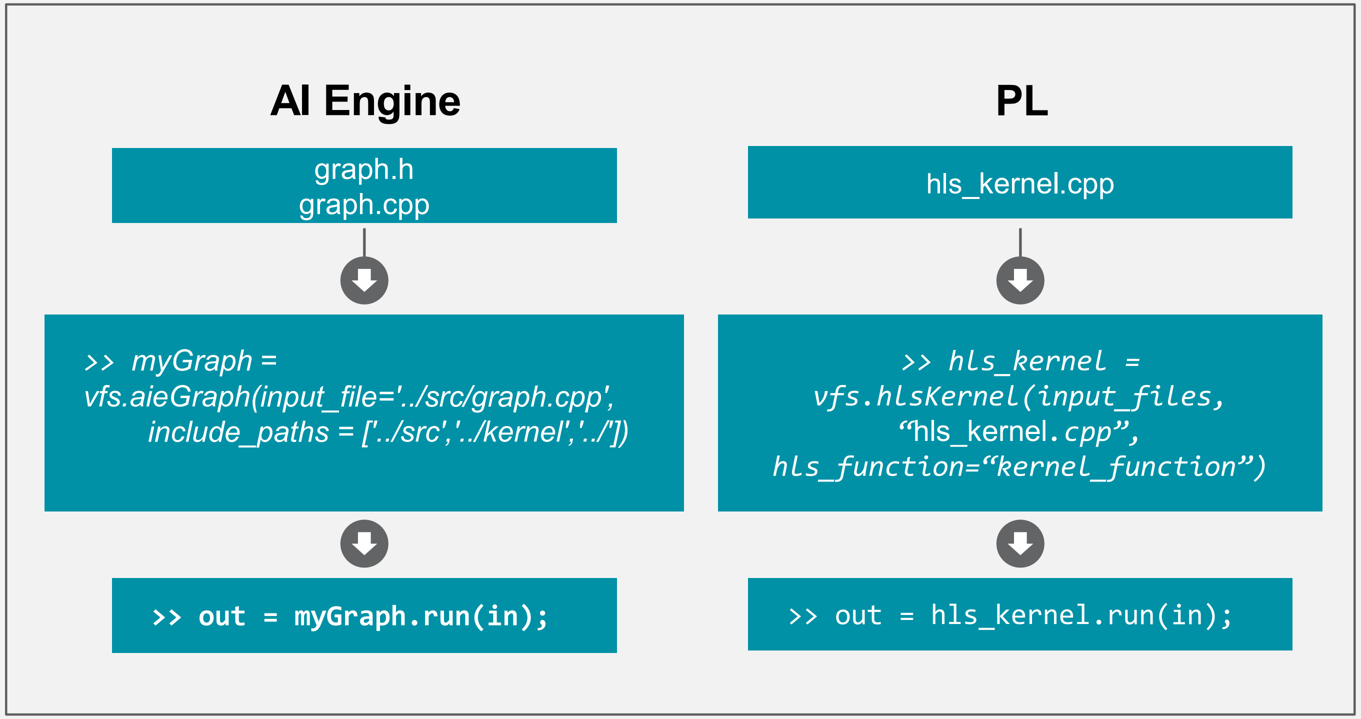
AMD Vitis™ Functional Simulation – MATLAB Environment

MATLAB® tool is the DSP environment of choice

Allows calling AIE graphs and PL (HLS) kernels directly from m-code

Generate data and simulate and verify functionality in the MATLAB environment





```
>> myGraph =
vfs.aieGraph(input_file='../src/graph.cpp',
      include_paths = ['../src','../kernel','../'])
```

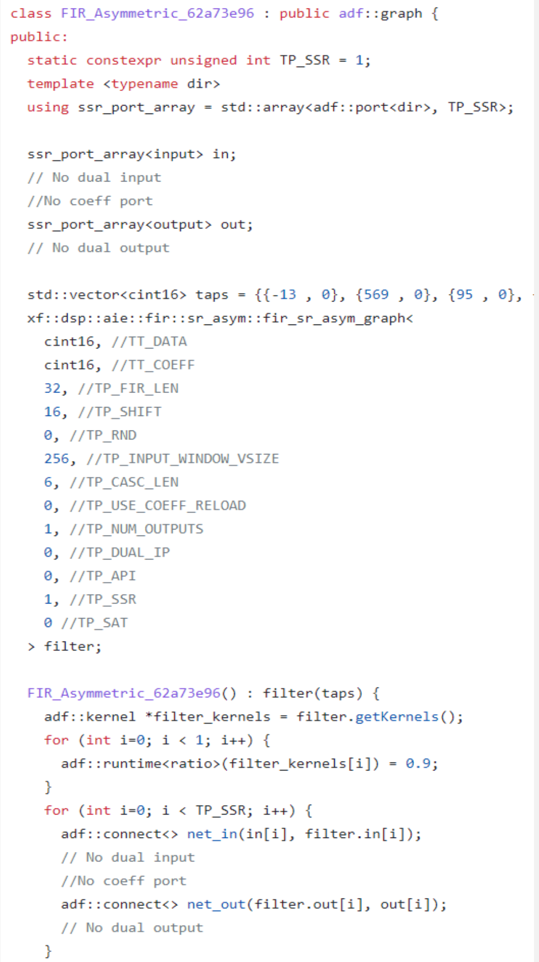
```
>> out = myGraph.run(in);
```

```
>> hls_kernel =
vfs.hlsKernel(input_files,
“hls_kernel.cpp”,
hls_function=“kernel_function”)
```

```
>> out = hls_kernel.run(in);
```

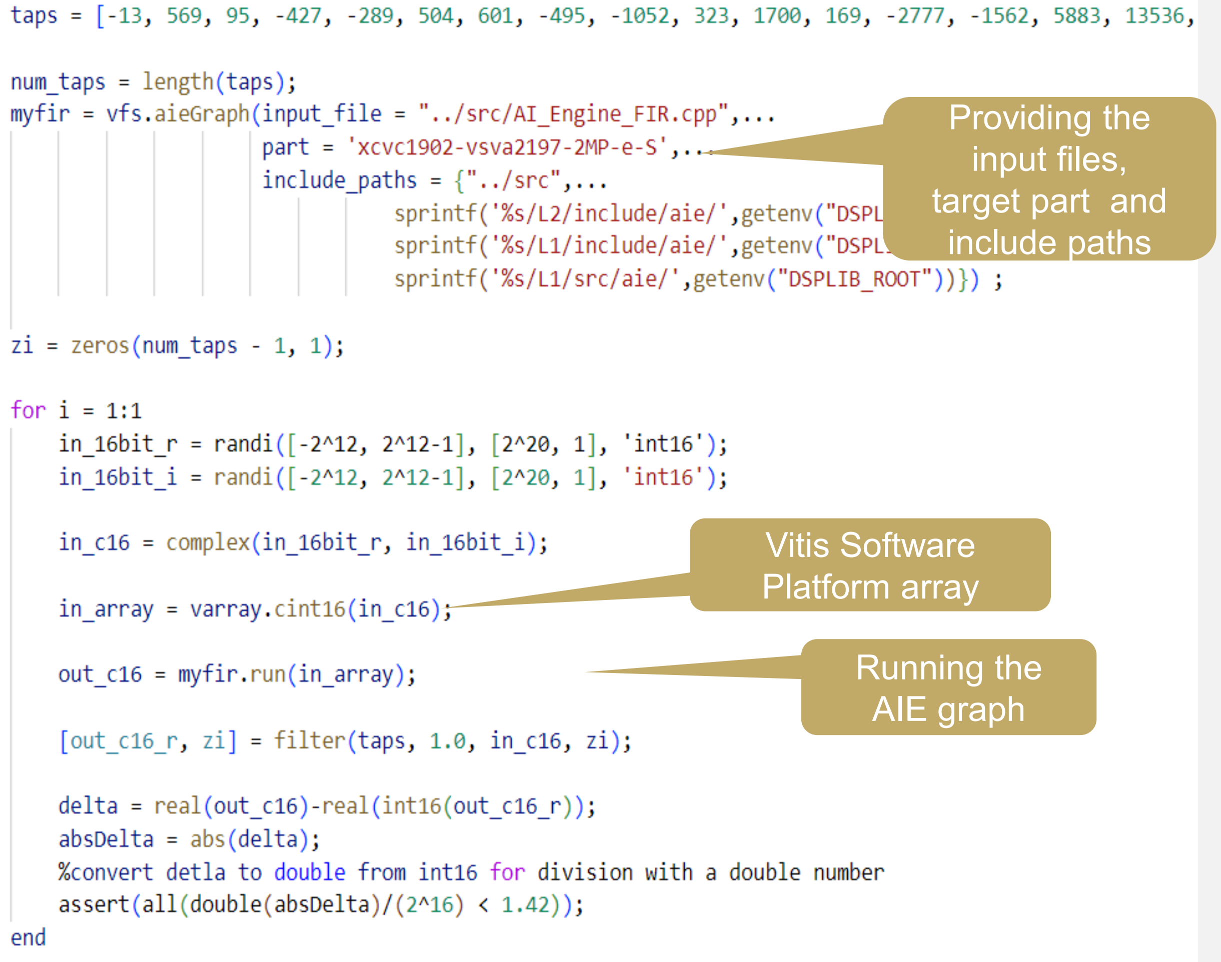
AMD VFS with MATLAB Tool – Example

FIR – AI Engine Graph



```
class FIR_Asymmetric_62a73e96 : public adf::graph {
public:
  static constexpr unsigned int TP_SSR = 1;
  template <typename dir>
  using ssr_port_array = std::array<adf::port<dir>, TP_SSR>;

  ssr_port_array<input> in;
  // No dual input
  //No coeff port
  ssr_port_array<output> out;
  // No dual output

  std::vector<cint16> taps = {{-13 , 0}, {569 , 0}, {95 , 0},
  xf::dsp::aie::fir::sr_asym::fir_sr_asym_graph<
    cint16, //TT_DATA
    cint16, //TT_COEFF
    32, //TP_FIR_LEN
    16, //TP_SHIFT
    0, //TP_RND
    256, //TP_INPUT_WINDOW_VSIZE
    6, //TP_CASC_LEN
    0, //TP_USE_COEFF_RELOAD
    1, //TP_NUM_OUTPUTS
    0, //TP_DUAL_IP
    0, //TP_API
    1, //TP_SSR
    0 //TP_SAT
  > filter;

  FIR_Asymmetric_62a73e96() : filter(taps) {
    adf::kernel *filter_kernels = filter.getKernels();
    for (int i=0; i < 1; i++) {
      adf::runtime<ratio>(filter_kernels[i]) = 0.9;
    }
    for (int i=0; i < TP_SSR; i++) {
      adf::connect<> net_in(in[i], filter.in[i]);
      // No dual input
      //No coeff port
      adf::connect<> net_out(filter.out[i], out[i]);
      // No dual output
    }
```

VFS Call within MATLAB® to Graph Code



```
taps = [-13, 569, 95, -427, -289, 504, 601, -495, -1052, 323, 1700, 169, -2777, -1562, 5883, 13536,

num_taps = length(taps);
myfir = vfs.aieGraph(input_file = "../src/AI_Engine_FIR.cpp",...
                     part = 'xcvc1902-vsva2197-2MP-e-S',...
                     include_paths = {"../src",...
                                sprintf('%s/L2/include/aie/',getenv("DSPL
                                sprintf('%s/L1/include/aie/',getenv("DSPL
                                sprintf('%s/L1/src/aie/',getenv("DSPLIB_ROOT"))}) ;

zi = zeros(num_taps - 1, 1);

for i = 1:1
    in_16bit_r = randi([-2^12, 2^12-1], [2^20, 1], 'int16');
    in_16bit_i = randi([-2^12, 2^12-1], [2^20, 1], 'int16');

    in_c16 = complex(in_16bit_r, in_16bit_i);

    in_array = varray.cint16(in_c16);

    out_c16 = myfir.run(in_array);

    [out_c16_r, zi] = filter(taps, 1.0, in_c16, zi);

    delta = real(out_c16)-real(int16(out_c16_r));
    absDelta = abs(delta);
    %convert detla to double from int16 for division with a double number
    assert(all(double(absDelta)/(2^16) < 1.42));
end
```

Providing the input files, target part and include paths

Vitis Software Platform array

Running the AIE graph

Creates an AI Engine graph / HLS kernel object

AMD Vitis™ Software Platform array appears, behaves, and operates like MATLAB array

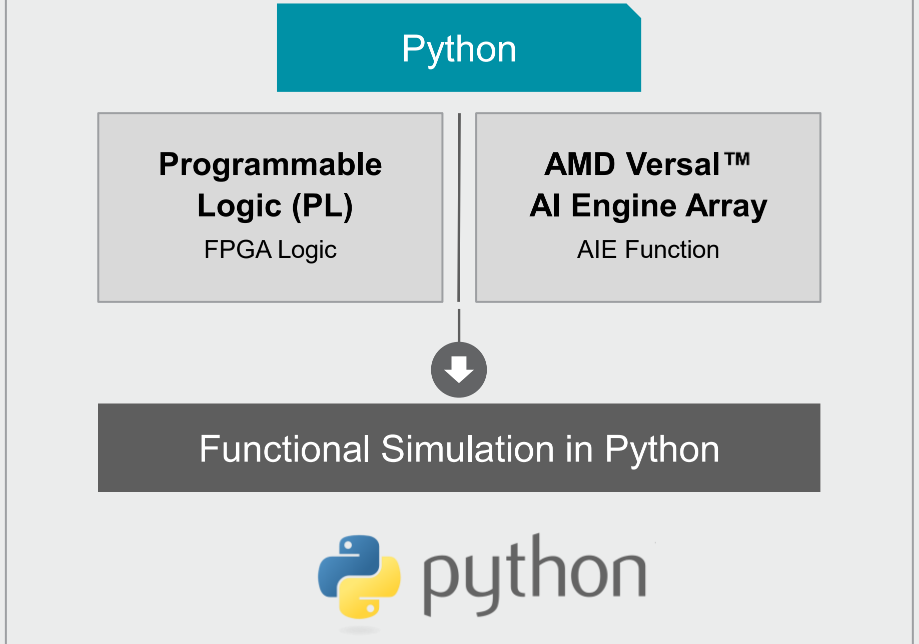
“run” takes Vitis Software Platform array input, runs the graph, and produces the output

AMD Vitis™ Functional Simulation – Python Scripting

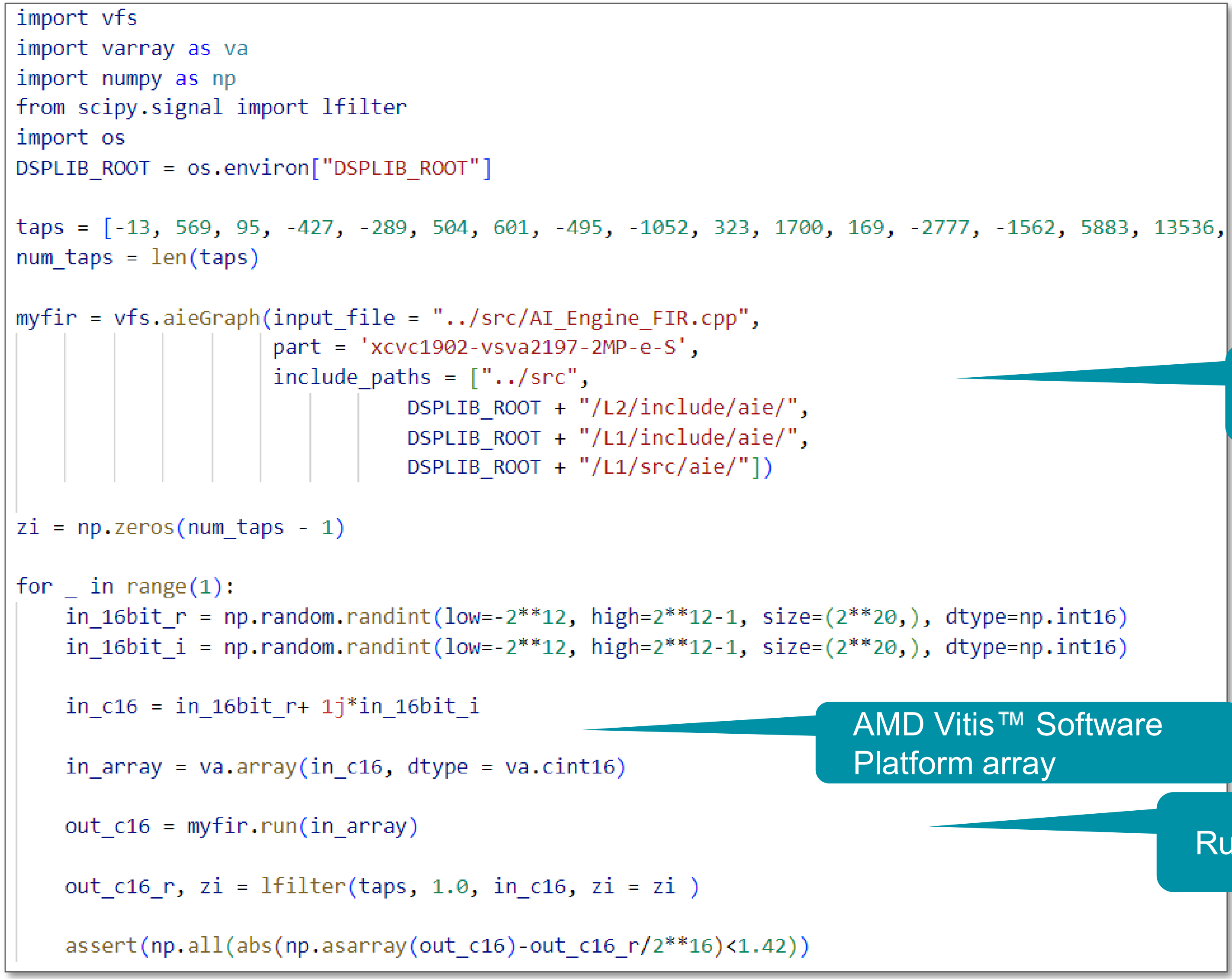
Python™ scripting is a popular choice in machine learning applications

AMD VFS allows calling AI Engine graphs and PL (HLS) kernels directly from Python scripts

Users can take the design to the Python environment – functionally simulate and verify



AMD VFS with Python™ Scripting – Example



```
import vfs
import varray as va
import numpy as np
from scipy.signal import lfilter
import os
DSPLIB_ROOT = os.environ["DSPLIB_ROOT"]

taps = [-13, 569, 95, -427, -289, 504, 601, -495, -1052, 323, 1700, 169, -2777, -1562, 5883, 13536,
num_taps = len(taps)

myfir = vfs.aieGraph(input_file = "../src/AI_Engine_FIR.cpp",
                     part = 'xcvc1902-vsva2197-2MP-e-S',
                     include_paths = ["../src",
                                DSPLIB_ROOT + "/L2/include/aie/",
                                DSPLIB_ROOT + "/L1/include/aie/",
                                DSPLIB_ROOT + "/L1/src/aie/"])

zi = np.zeros(num_taps - 1)

for _ in range(1):
    in_16bit_r = np.random.randint(low=-2**12, high=2**12-1, size=(2**20,), dtype=np.int16)
    in_16bit_i = np.random.randint(low=-2**12, high=2**12-1, size=(2**20,), dtype=np.int16)

    in_c16 = in_16bit_r+ 1j*in_16bit_i

    in_array = va.array(in_c16, dtype = va.cint16)

    out_c16 = myfir.run(in_array)

    out_c16_r, zi = lfilter(taps, 1.0, in_c16, zi = zi )

    assert(np.all(abs(np.asarray(out_c16)-out_c16_r/2**16)<1.42))
```

Providing the input files, target part and include paths

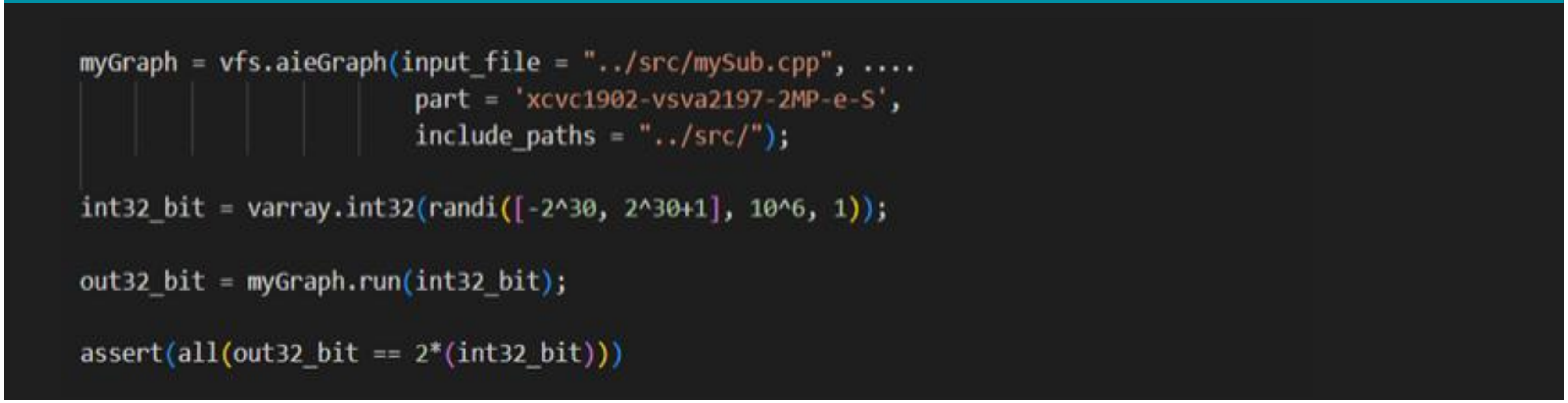
AMD Vitis™ Software Platform array

Running the AIE graph

AMD VFS with MATLAB and Python – Example Comparison

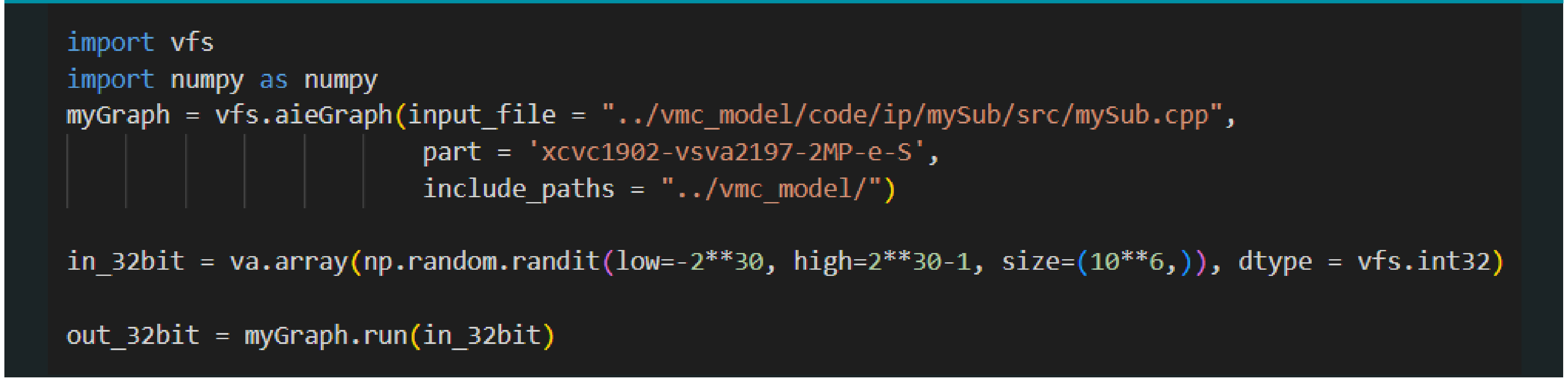
Source: <path_to_tools>/settings64.(c)sh

MATLAB® Environment



```
myGraph = vfs.aieGraph(input_file = "../src/mySub.cpp", ....
                       part = 'xcvc1902-vsva2197-2MP-e-S',
                       include_paths = "../src/");

int32_bit = varray.int32(randi([-2^30, 2^30+1], 10^6, 1));

out32_bit = myGraph.run(int32_bit);

assert(all(out32_bit == 2*(int32_bit)))
```

Python™ Environment

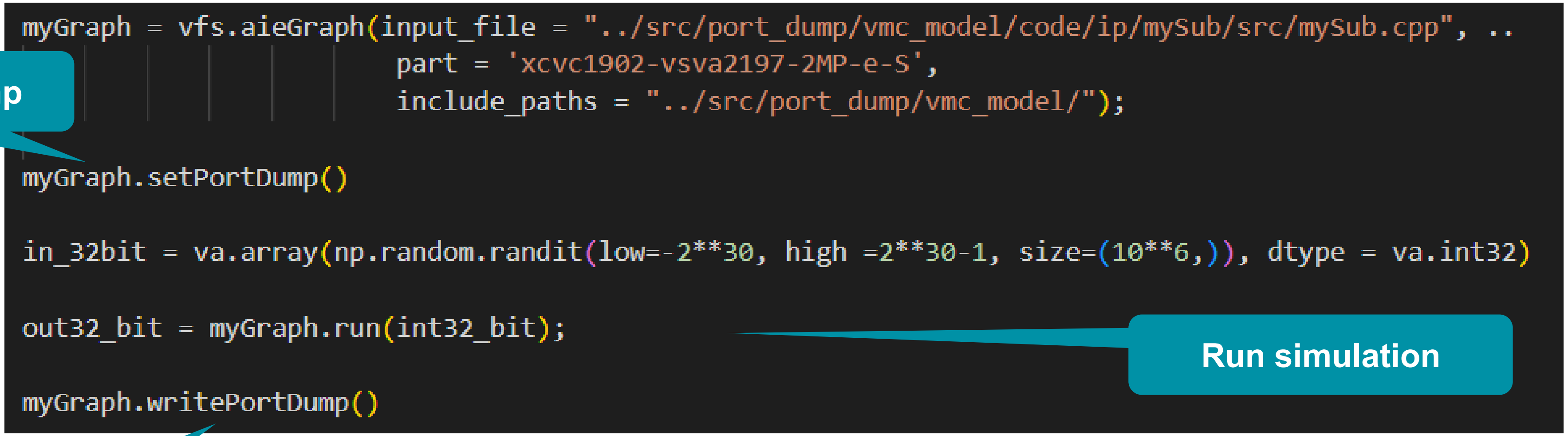


```
import vfs
import numpy as numpy
myGraph = vfs.aieGraph(input_file = "../vmc_model/code/ip/mySub/src/mySub.cpp",
                        part = 'xcvc1902-vsva2197-2MP-e-S',
                        include_paths = "../vmc_model/")

in_32bit = va.array(np.random.randit(low=-2**30, high=2**30-1, size=(10**6,)), dtype = vfs.int32)

out_32bit = myGraph.run(in_32bit)
```

Port Dump for AI Engine Graph

You can dump all the ports within the AI Engine graph into files:

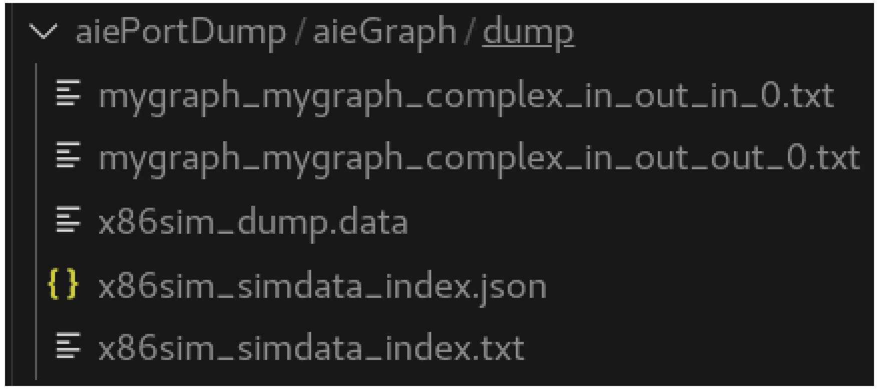
Enable port dump



```
myGraph = vfs.aieGraph(input_file = "../src/port_dump/vmc_model/code/ip/mySub/src/mySub.cpp", ..
                        part = 'xcvc1902-vsva2197-2MP-e-S',
                        include_paths = "../src/port_dump/vmc_model/");

myGraph.setPortDump()

in_32bit = va.array(np.random.randit(low=-2**30, high =2**30-1, size=(10**6,)), dtype = va.int32)

out32_bit = myGraph.run(int32_bit);

myGraph.writePortDump()
```

Run simulation

Dump files



```
aiePortDump / aieGraph / dump
    mygraph_mygraph_complex_in_out_in_0.txt
    mygraph_mygraph_complex_in_out_out_0.txt
    x86sim_dump.data
    x86sim_simdata_index.json
    x86sim_simdata_index.txt
```

AMD
together we advance_

Summary

01 AMD Vitis™ functional simulation (VFS) enables customers to develop subsystems in their own simulation frameworks

02 Vitis Array supports all the datatypes available in AMD devices in both MATLAB® and Python™ environments and allows conversion and casting between the types

03 VFS allows calling AI Engine graphs and HLS kernels directly from MATLAB or Python code

04 Subsystem simulation can be performed either through MATLAB or Python frameworks with VFS

05 VFS can provide a high level of design visibility as it enables simulation and debug when build up a design

General Disclaimer and Attribution Statement 2025

The information contained herein is for informational purposes only and is subject to change without notice. While every precaution has been taken in the preparation of this document, it may contain technical inaccuracies, omissions and typographical errors, and AMD is under no obligation to update or otherwise correct this information. Advanced Micro Devices, Inc. makes no representations or warranties with respect to the accuracy or completeness of the contents of this document, and assumes no liability of any kind, including the implied warranties of noninfringement, merchantability or fitness for particular purposes, with respect to the operation or use of AMD hardware, software or other products described herein. No license, including implied or arising by estoppel, to any intellectual property rights is granted by this document. Terms and limitations applicable to the purchase or use of AMD products are as set forth in a signed agreement between the parties or in AMD's Standard Terms and Conditions of Sale. GD-18u.

Certain AMD technologies may require third-party enablement or activation. Supported features may vary by operating system. Please confirm with the system manufacturer for specific features. No technology or product can be completely secure.

AMD