Super Sample Rate Digital Fourier Transform Design with AI Engine

XAPP1400 (v1.0) September 20, 2023

Summary

This application note introduces a high-throughput low-latency inverse discrete fourier transformation (IDFT) implementation in AI Engine arrays of AMD Versal™ AI Core adaptive SoCs. DFT and IDFTs are widely used in various signal processing applications that often require high throughput and low processing delay. This application note includes a 1536-point IDFT reference design to illustrate a super sample rate (SSR) implementation using AI Engine. SSR refers to a scenario in which the sample rate exceeds the clock frequency of the processing unit, and the input data must be processed in a parallel manner to meet the throughput and latency requirements. To quickly validate the design in hardware, high-level synthesis (HLS) is employed to develop programmable logic (PL) kernels as a test bench that provides test stimuli and monitors outputs. The hardware test result shows that this design achieves 5 giga samples per second (GSPS) throughput with a latency of around 2 µs.

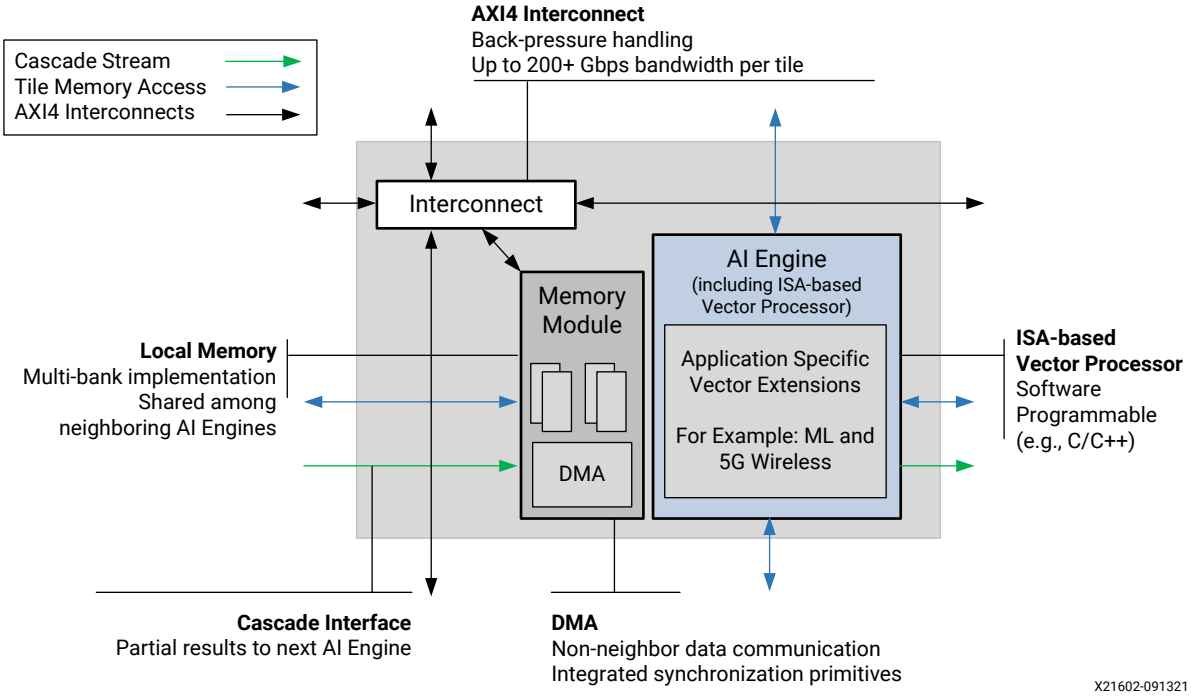
Download the reference design files for this application note from the Xilinx website. For detailed information about the design files, see Reference Design.

Introduction

AI Engines are an array of very-long instruction word (VLIW) processors with single instruction multiple data (SIMD) vector units. They are highly optimized for compute-intensive applications (specifically digital signal processing (DSP)), 5G wireless applications, and artificial intelligence (AI) technology, such as machine learning (ML). AI Engine is able to execute a vector MAC operation, load two 256-bit vectors for the next operation, store a 256-bit vector from the previous operation, and increment a pointer or execute another scalar operation in each clock cycle at 1 GHz in the slowest Versal device.

One AI Engine can execute eight MACs of 16-bit complex data in one cycle. In order to maximize efficiency, the system structure and data path should be vectorized to take full advantage of the computation capability.

Figure 1: **AI Engine Architecture**



This IDFT design targets 5 Gbps throughput which requires six AXI buses running at 834 MSPS for input and output. With a 128-bit wide data bus, the AI Engine/PL interface logic runs at 250 MHz clock frequency, which is within reasonable performance limits for HLS-generated circuits. After rigorous block-level testing, the IDFT block can be integrated into a larger system where the RTL programming language offers higher clock frequency for better performance.

AMD Vitis™ HLS is a high-level synthesis tool that allows C, C++, and OpenCL™ functions to be synthesized onto the FPGA logic and RAM/DSP blocks. The design of this application note uses the AMD Vitis™ acceleration flow to integrate AI Engine with PL kernels developed by Vitis HLS.

Multi-Core AI Engine Architecture

DFT Theory of Operation

DFT is one of the most fundamental operations in signal processing that converts a finite sequence of samples in the time domain into the same length as a finite sequence of samples in the frequency domain. The original equation for DFT is as follows:

Equation 1: **Definition of DFT**

$$X(k) = \sum_{n=0}^{N-1} x(n) \cdot e^{\left(-j\frac{2\pi nk}{N}\right)} \qquad k = 0,1,\ldots,N-1$$

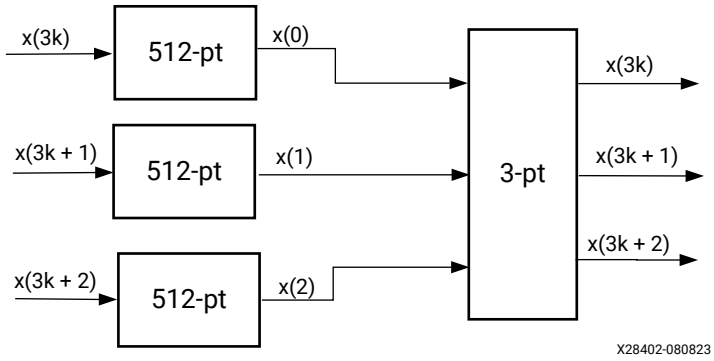
Direct implementation of the previous equation involves large amounts of complex data multiplication when N is large. More efficient implementation has been proposed in the *Split radix FFT algorithm* literature [3]. For this design, N is 1536, which is not a power of 2. The mixed radix FFT algorithm is employed to divide the calculation into a series of radix-2, radix-3, and radix-4 operations. For example, $1536 = 512 \times 3 = 4^4 \times 2 \times 3$.

This equation suggests it be split into three 512-point IFFTs whose outputs are interleaved for 512 3-point IDFTs. The 512-point IFFTs are split into four stages of radix-4 operations and one stage of radix-2 operation.

SSR Architecture for AI Engine Graph

For 5G and beyond communication systems, low latency is as critical of a requirement as throughput which calls for a parallel structure to fully take advantage of AI Engine computation capability. The following figure is a simplified scheme describing the partition of computation and shows the complete graph diagram.

Figure 2: **IDFT Calculation Stages**

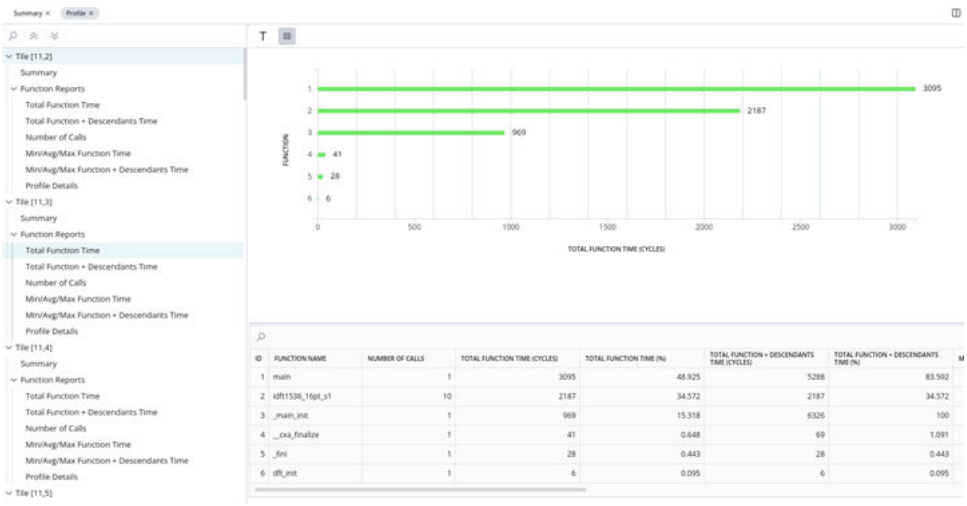


After the IDFT structure is determined, the next step is to consider the mapping of each stage onto an AI Engine kernel. AMD has published many FFT designs with optimized radix-2, radix-3, and radix-4 functions [1, 4].

Because the system throughput is determined by the slowest processing node in a chain, it is often required to iterate multiple times to locate the performance bottleneck, further optimize certain kernel functions, and adjust functional partitions. In this process, the Vitis Analyzer tool offers detailed performance reports and informative trace views for the identification of various issues.

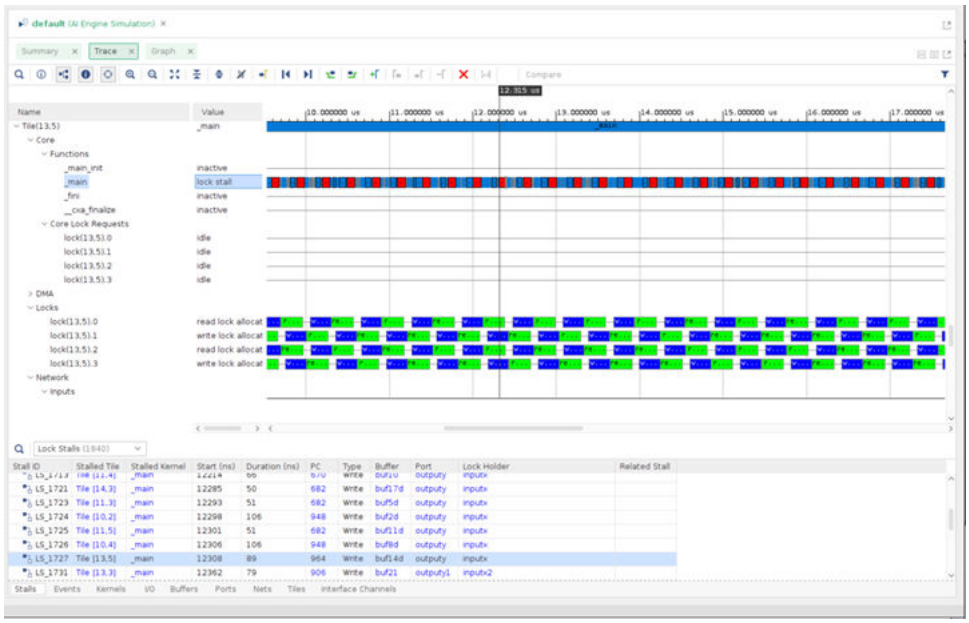
By adding the option `--profile` in AI Engine simulation, the profile summary report can be opened after simulation [2]. You can then check the cycle count for each kernel function to determine which kernel is limiting the overall throughput and thus needs to be optimized. The following figure shows the 4-pt IFFT kernel profiling report.

Figure 3: **AI Engine Simulation Profile View**



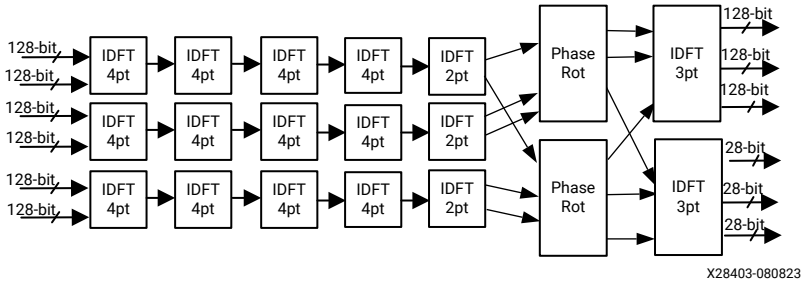
To determine potential performance issues, trace reports can also be used as a reference [2]. The following figure is an example. The red bars that the cursor points to indicate memory stall occurrences. The table at the bottom elaborates on the related buffers which cause memory access delay. Some memory stalls are inevitable and do not need to be resolved; however, others can be caused by kernel design or inappropriate memory bank assignment. Those problems can be identified and addressed accordingly.

Figure 4: **AI Engine Simulation Trace View**



For each AI Engine, input and output data transfers occur in parallel to the computation. The efficiency of the vector processor is maximized when the loops are perfectly pipelined and the time needed by data transfer does not exceed that of computation. With all the considerations of every AI Engine kernel's processing delay, AI Engine tile resource usage, and memory buffer floorplan, the AI Engine implementation architecture is shown in the following figure.

Figure 5: **1536-point IDFT AI Engine Design Architecture**



In the previous figure, every rectangular block represents an AI Engine kernel that occupies one AI Engine tile. All data transfers are based on Windows, and the data types are 16-bit complex at the input/output interfaces and 32-bit complex for intermediate results.

For ease of hardware validation, the AXI4-Stream buses are set to be 128 bits wide so that the PL only needs to run at 250 MHz to support up to 1 GSPS throughput. The target throughput of this design is 5 GSPS, which translates to 834 MSPS per AXI bus and is well within the capability of 1 GSPS.

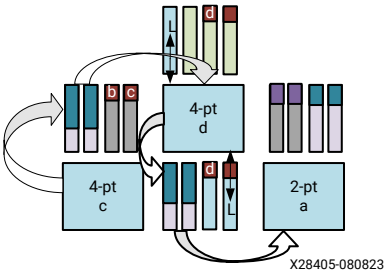
After the architecture is defined, AI Engine kernel and graph coding can start and be verified by AI Engine simulation before integration. All outputs can be saved to files and compared to the golden reference results generated by the ideal MATLAB® model. Also, the time stamps in AI Engine outputs can be analyzed to estimate the average throughput over a large number of iterations.

Floorplan

The Vitis AI Engine compiler can automatically find AI Engine placement and routing solutions for your designs. However, that only serves as a lower bound for the performance because you can always optimize the solution with manual placement including kernel tiles, buffers, PLIOs, and more. The location constraints to the SSR-IDFT design are added with the following steps:

1. Plan initial placement for all AI Engine kernel tiles. In this step, consider the dataflow between each AI Engine kernel and keep in mind the memory buffer access requirements. As shown in the following figure, two pairs of ping-pong buffers are used to connect the 4-pt *c* kernel, 4-pt *d* kernel, and 2-pt *a* kernel. They are arranged in memory banks that can be accessed by both upstream and downstream kernels.
2. Multiple buffers can share one memory bank if there are not too many access conflicts. As shown in the following figure, two LUT buffers are needed to be accessed by the 4-pt *d* kernel. LUT0 and LUT1 might be requested at the same time, so it is best to put LUT0 and LUT1 in different memory banks to avoid memory conflicts.

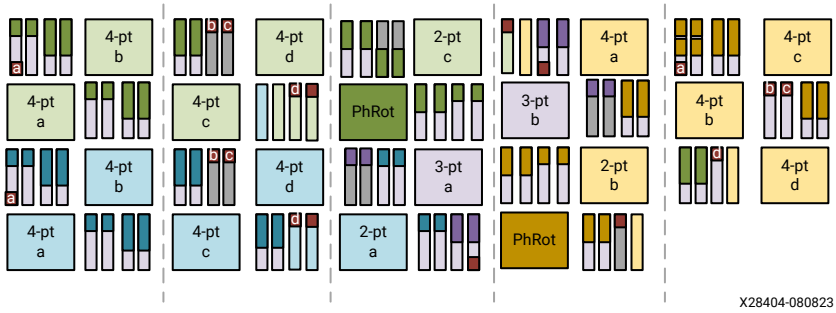
Figure 6: **Design Placement**



3. Try to pack the design into a nice rectangle region and avoid occupying memory banks of unused AI Engine tiles.
4. Run AI Engine simulation and profiling to see the effects of each adjustment. If a memory stall happens, adjust the placement by understanding where conflicts occur.

The following figure shows the final floorplan used for this design.

Figure 7: **1536-point IDFT AI Engine Design Floorplan**

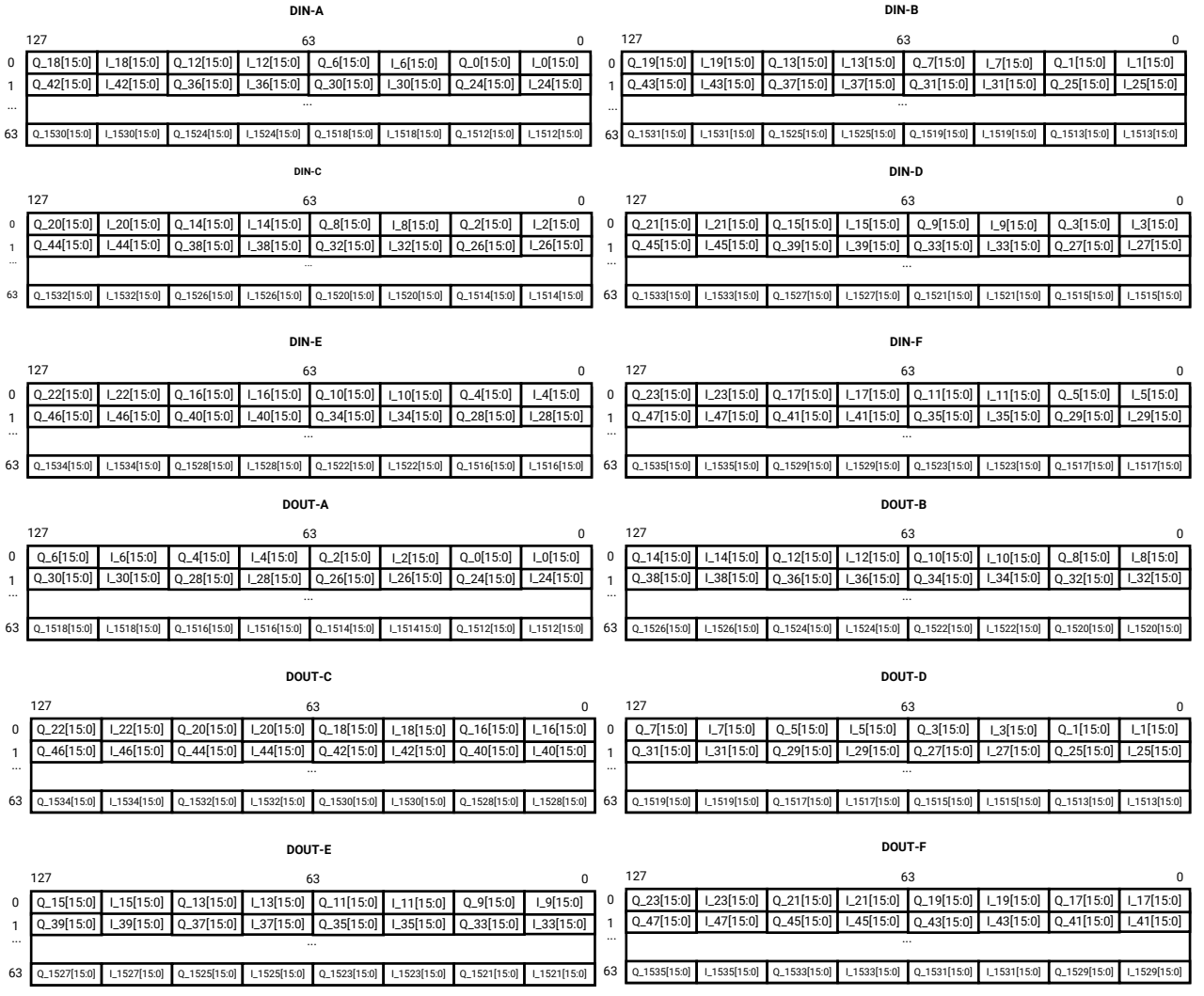


Optimal placement should consider the communications between related AI Engine tiles and the number of buffers that need to be accessed by each tile. It is best to have every buffer occupy the memory bank exclusively. When it is not possible, the trace view in the Vitis analyzer is a handy tool to locate memory conflicts leading to throughput degradation.

Input and Output Data Format

The input and output to IDFT design are six data blocks, each containing 256 samples.

Figure 8: **Input/Output Data Packet**



System Integration Test

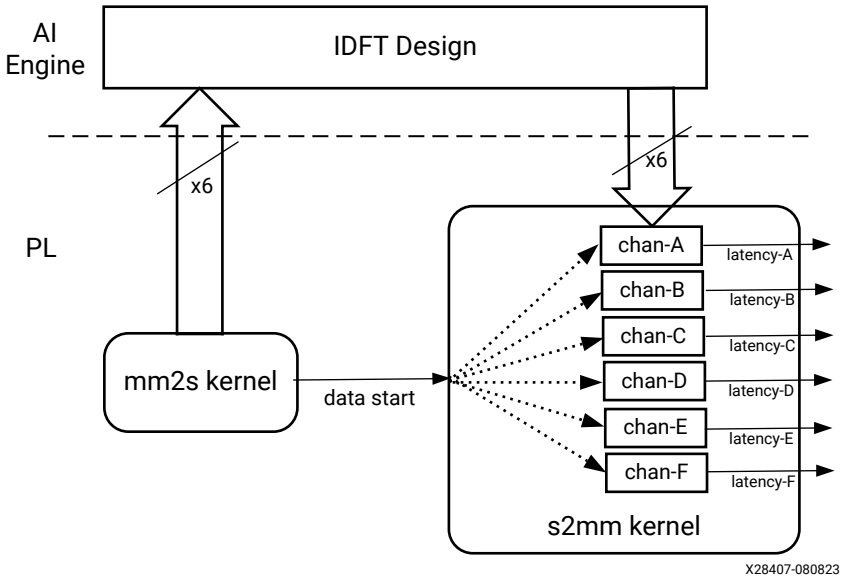
To verify the IDFT design with more accuracy, a complete system is developed that consists of the following modules: Each of these modules is integrated together by the Vitis tool and run through hardware emulation and on-board testing to get more accurate system performance data. The following sections provide more details about each module.

- AI Engine kernels which implement the core IDFT algorithm.
- PL kernels which implement the test vector transmission and AI Engine output data verification. The PL kernels are designed in C++ and synthesized into traditional FPGA logic by Vitis HLS.
- A target platform is required. In this case, the off-the-shelf VCK190 platform is used with minor modifications.
- Host application which implements the control of this system.

PL Kernel Design by HLS

The PL kernels are developed for generating test vectors for the AI Engine kernel and verifying the correctness of the output data produced by AI Engine. The PL kernels are designed in C++ and synthesized by the Vitis HLS tool for high productivity.

Figure 9: **PL and AI Engine Diagram**



The test data generated by MATLAB is stored in FPGA block RAM components. The data mover function fetches the data from internal BRAM and sends it to the AI Engine array through the AXI4-Stream interface. Another PL kernel reads data from the AI Engine output port and compares it with golden reference data which is also stored in memory. The number of errors, total data received, and AI Engine design latency are counted and reported to the processor via registers. Those PL kernels are initiated by the host code running on an Arm® processor. After their tasks are completed, they set the `DONE` flags in the register map monitored by the host program. All interface and register map information is automatically handled by the tool, so you do not need to pay attention to those details.

Host Code and Target Platform

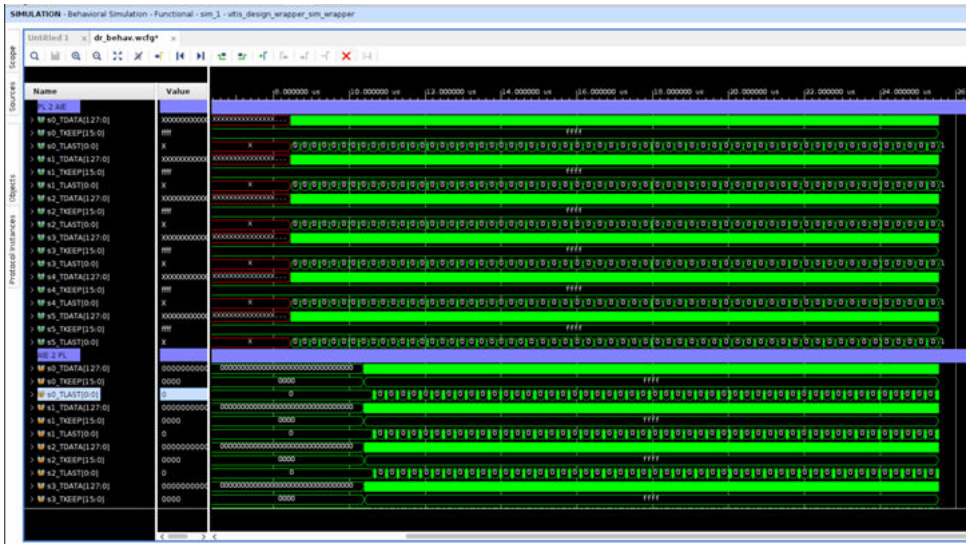
This design uses a bare-metal host application to manage the AI Engine kernel and PL kernel execution. It performs low-level register access commands such as `Xil_Out32` and `Xil_Out32` to set parameters for the PL kernels and to read back the content of registers. This design is built on a customized bare-metal platform which is based on the official VCK190 production board. The design has some changes made to the clocking scheme and some unused peripherals are removed.

Hardware Emulation in Vitis Software Platform

Hardware emulation is an effective way to verify the whole system's function and performance before running the design on a board. It is a co-simulation environment that uses QEMU to model the Arm processor and simulate the PL design in the AMD Vivado™ simulator or other third-party simulator. A SystemC model is used to model the AI Engine design in this co-simulation flow.

Hardware emulation is done in Vitis, which offers a set of features for system debugging. For example, the signal waveform of the interface between PL and AI Engine can be viewed in the Vivado Integrated Design Environment (IDE) so that the AI Engine+PS+PL design can be debugged in the same way as a pure RTL design. The following figure shows an example waveform snapshot of the AXI4-Stream interface traffic between PL and AI Engine.

Figure 10: **PL and AI Engine Traffic Waveform**



Hardware Build and Run

This design targets the VCK190 evaluation board and generates the hardware binary file flow through the Vitis software platform. After the `BOOT.BIN` file is generated, it can be copied into an SD card to boot the system and run the application. The AI Engine executable is loaded and run immediately at boot for this design.

To facilitate on-board testing capability, a traditional ChipScope™ ILA debugging method is also supported in the Vitis design flow [2]. During the Vitis linking stage, enable the ILA auto-insertion using the following command:

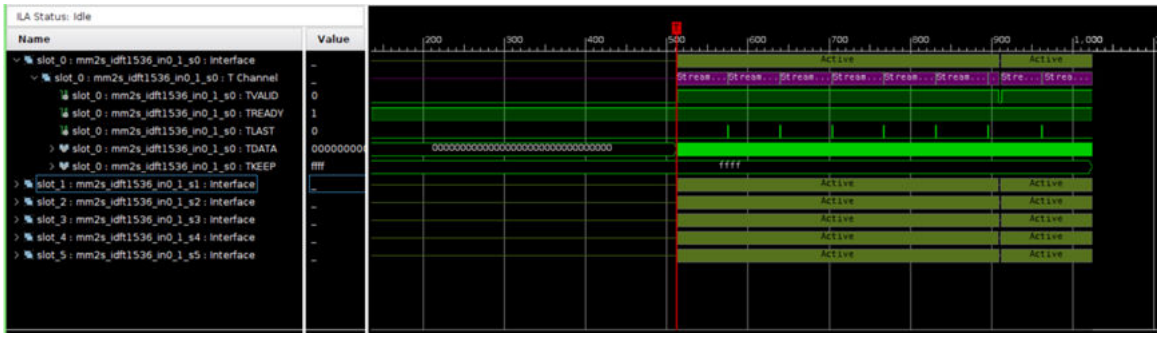
```
--debug.chipscope
        <cu_name>[:<interface_name>]>
```

Add the following to the host code to pause the host program execution for ILA trigger setting up.

```
std::cout << "Setting up ILA Trigger and click Enter to continue" <<
std::endl;
std::cin >> t_set;
std::cout << "ILA setup completed" << std::endl;
```

After trigger set-up is complete and data capture mode is launched in Vivado hardware manager, continue the host program running until the trigger condition is met. The following figure shows an example of the captured signal waveform.

Figure 11: **Chipscope ILA Waveform**



This provides the capability to debug issues that cannot be reproduced in emulation.

Tool Flow

This design uses a Makefile to ensure results can be reproduced consistently. At the same time, summary reports and waveforms created by the scripts can be loaded into the GUI for visual analysis. This design is verified with the Vitis 2023.1 release on a VCK190 evaluation board.

Table 1: **On-board Testing Result**

Kernel	Start Time (PL Cycles)	End Time (PL Cycles)	Throughput (MSPS)	Latency (PL Cycles)	Errors Count (Samples)	Total Count (Samples)
IDFT1536-0a	493006	614845	840.5	499	0	819200
IDFT1536-0b	493006	614846	840.4	499	0	819200
IDFT1536-0c	493008	614846	840.5	501	0	819200
IDFT1536-0d	493010	614849	840.5	503	0	819200
IDFT1536-0e	493009	614850	840.4	502	0	819200
IDFT1536-0f	493009	614848	840.5	502	0	819200

Reference Design

Download the reference design files for this application note from the Xilinx website.

Reference Design Matrix

The following checklist indicates the procedures used for the provided reference design.

Table 2: **Reference Design Matrix**

Parameter	Description
General	
Developer name	AMD
Target devices	Versal AI Core devices
Source code provided?	Yes
Source code format (if provided)	MATLAB® script, AI Engine C code, HLS C code, and Makefile
Design uses code or IP from existing reference design, application note, 3rd party or Vivado software? If yes, list.	No
Simulation	
Functional simulation performed	Yes
Timing simulation performed?	No
Test bench provided for functional and timing simulation?	No
Test bench format	C
Simulator software and version	AI Engine Simulator and XSIM in Vitis 2023.1
SPICE/IBIS simulations	Yes
Implementation	
Implementation software tool(s) and version	Vitis software platform 2023.1
Static timing analysis performed?	Yes
Hardware Verification	
Hardware verified?	Yes
Platform used for verification	VCK190

Conclusion

This application note introduces a high-throughput low-latency 1536-point IDFT design with AI Engine arrays. This design adopts an SSR architecture to achieve high throughput at low latency. During the AI Engine design process, architectural exploration and kernel optimization are performed to meet throughput and latency targets. Vitis HLS is employed to quickly build PL kernels for hardware testing. Vitis tool flow facilitates the heterogeneous system integration and testing in hardware.

References

These documents provide supplemental material useful with this application note:

1. *Block-by-Block Configurable Fast Fourier Transform Implementation on AI Engine* (XAPP1356)
2. *Vitis Unified Software Platform Documentation: Application Acceleration Development* (UG1393)
3. Duhamel P, Holtmann H. 5 January (1984, January 5). "Split radix" FFT algorithm. https://digital-library.theiet.org/content/journals/10.1049/el_19840012

4. Vitis AI Engine DSP Library
5. *Versal Adaptive SoC AI Engine Architecture Manual* (AM009)

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
09/20/2023 Version 1.0	
Initial release.	N/A

Please Read: Important Legal Notices