# Simplify Packet Processing Design with P4 and Vivado Tools

WP555 (v1.0) January 24, 2024

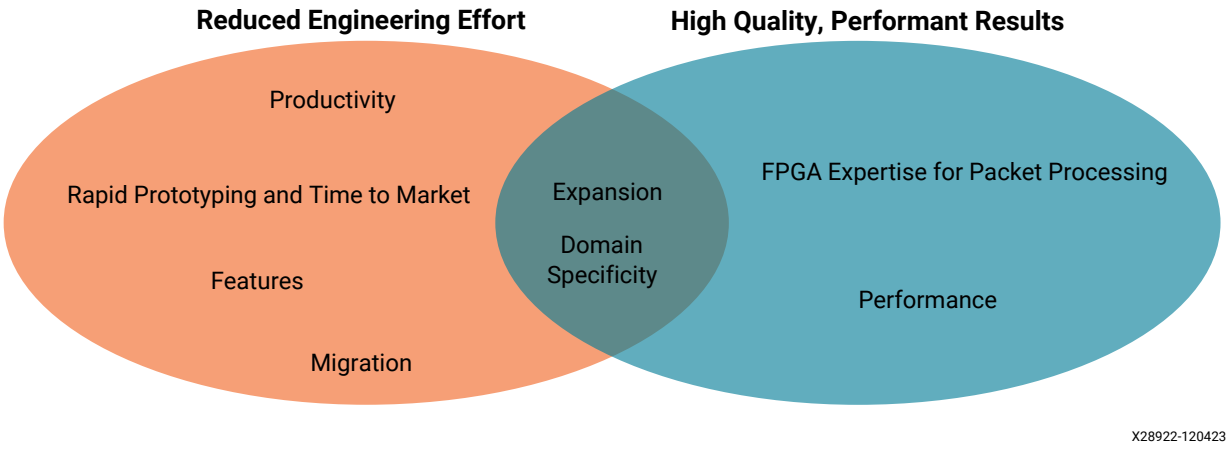
## Abstract

The AMD Vitis™ Networking P4 tool (VNP4) is a high-level design environment used to simplify the design of packet-processing data planes that target FPGAs and adaptive SoCs. It converts designs coded in P4—the ubiquitous network programming language—into device-ready RTL code for optimal hardware implementation. By using this tool, you can significantly reduce engineering effort required to develop device-based packet-processing systems, while still achieving high quality results in terms of performance per LUT or performance per RAM. The benefits of designing with VNP4 are outlined in this document.

# Introduction

The benefits of VNP4 fall broadly into two categories; reduced engineering effort and high quality, performant results.

*Figure 1:* **VNP4 Benefits**




- **Productivity:** The solution reduces development effort.
- **Rapid Prototyping and Time to Market:** Getting your product to market is faster with the accelerated design cycle. Iterating through multiple design options is simple and quick.
- **Features:** Extensive features differentiate your product, including options in *User Metadata* and *User Externs*.
- **Migration:** The design intent can be migrated from one FPGA or SoC to another.
- **Expansion:** Packet-processing blocks generated by VNP4 can be deployed in parallel or serially to support capabilities such as multi-level parsing and multi-data-pipeline systems.
- **Domain Specificity:** This high-level abstraction solution is domain specific, allowing you to take advantage of abstraction without sacrificing performance.
- **FPGA Expertise for Packet Processing:** The solution and quality of hardware implementation reflects years of experience in high-speed FPGA design and memory subsystems for high throughput packet processing.
- **Performance:** The system has been engineered from the ground up to ensure high throughput, low latency, and minimized resource utilization.

## Programming Protocol-independent Packet Processing

An industry standard, domain specific language, programming protocol-independent packet processors (P4), is used for requirements capture. VNP4 converts the P4 design intent into an AMD FPGA or adaptive SoC design solution and allows programmers to build new data planes by explicitly specifying the header and packet processing requirements. To implement a P4 design, the compiler maps the intended functionality onto a custom data plane architecture of VNP4 RTL engines and software drivers. This mapping chooses appropriate engine types and customizes each of them based on the P4-specified processing. The specialized engines used to achieve this goal include parsing engines, match-action engines, and deparsing engines, each generated according to an application-specific requirement.

The generated RTL is integrated in a packaged IP in the AMD Vivado™ Design Suite where it can immediately be combined with other standard IPs, such as media access controllers, to create a complete device design. The design is then synthesized and a bit-file is generated for the targeted device. Even before synthesis design data is generated, critical design metrics are available, such as required latency and memory resources.

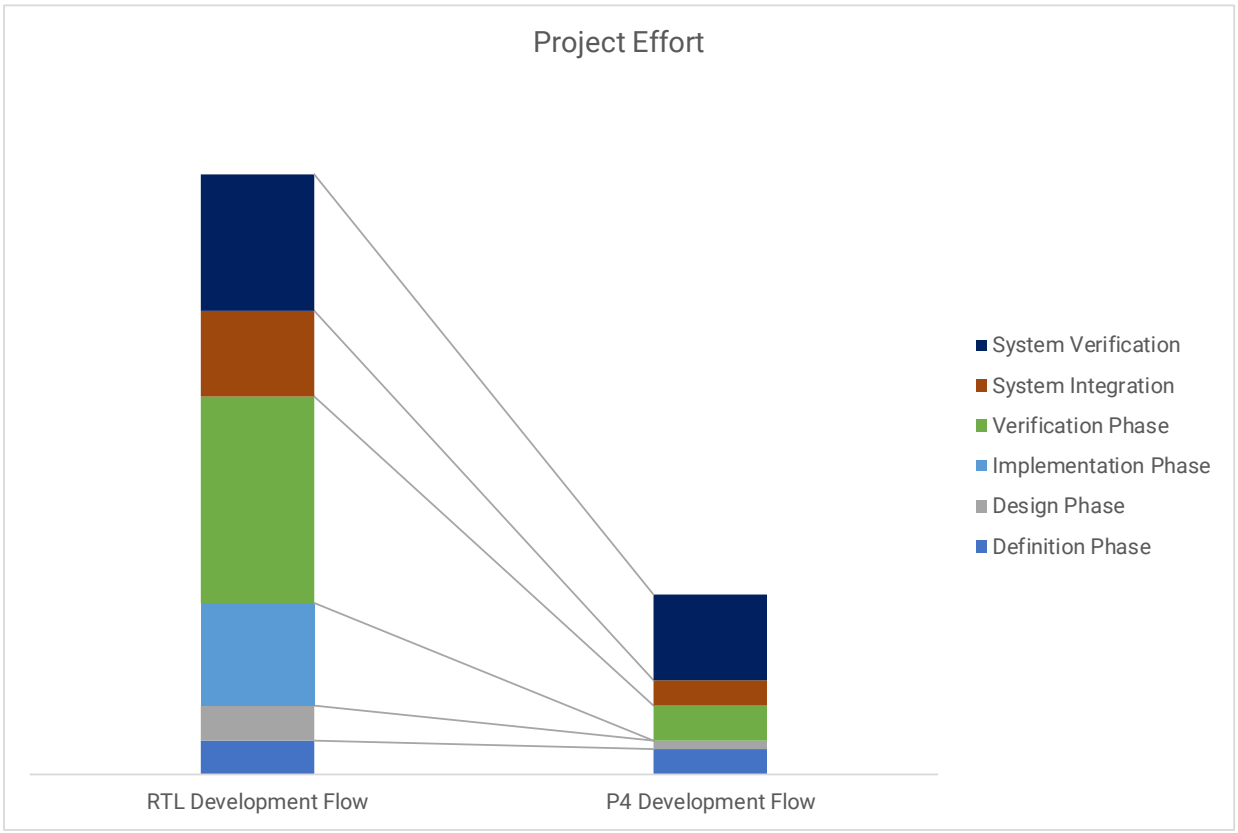
The current AMD solution was designed based upon feedback from hundreds of customers and information gathered from earlier iterations. The three key elements that differentiate this latest generation of the tool are:

- Native support for the $P4_{16}$ language
- Algorithmic content addressable memory technology
- Dedication to efficient resource utilization and robust timing closure

# Productivity

A major advantage of designing with VNP4 is the savings in development time and effort. This applies to the actual generation of the RTL, but also might be more significant during the RTL verification. To highlight where the savings exist, the following figure compares the different phases of a typical RTL development flow against the approach using VNP4.

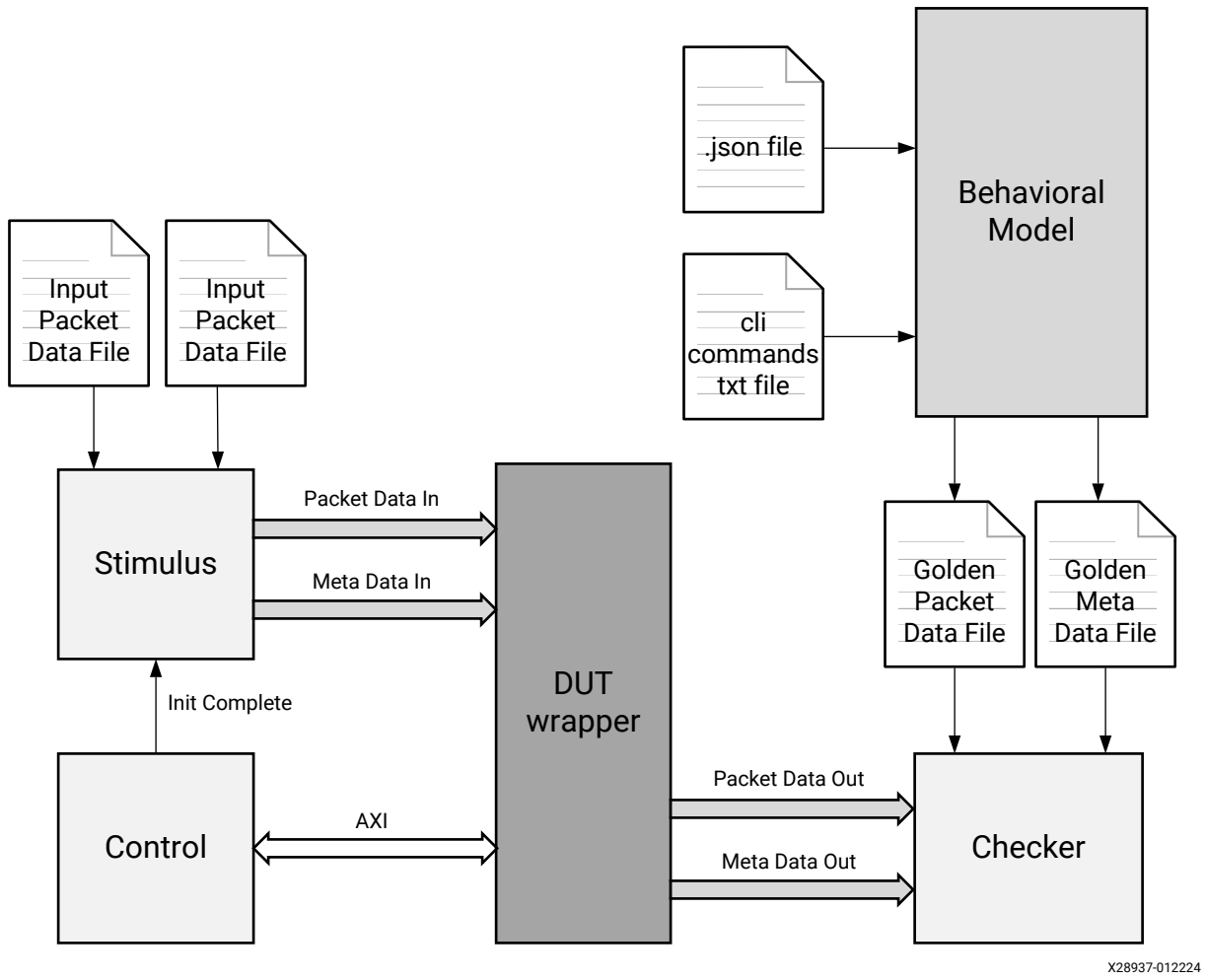
*Figure 2:* **Conceptual Breakdown of Project Effort**




The definition phase includes defining the scope of a project and capturing the important details in a requirements specification document. When it comes to packet processing, the requirements can be specified more efficiently and effectively with the use of P4 code compared to a requirements specification document. The P4 code is concise and less ambiguous, which helps to avoid misinterpretation later in the project. This consequently saves effort and time in those later stages. Many examples within the industry highlight the benefits of P4 as a specification language [4][5][6].

The definition phase also includes test planning, which involves decisions about the design of a test bench, the nature of the stimulus that is needed to test the design, and the nature of the checking mechanisms. VNP4 provides an example design including a SystemVerilog test bench with automated self-checking against the P4 behavioral model, allowing you to focus more on the stimulus side. The verification can be run using the P4 behavioral model, which has much faster runtimes. The P4's higher level of abstraction makes debug work much easier, where the model outputs a detailed log of each step through the P4 program as a packet is processed. RTL simulation is still recommended when integrating into a larger system design.

*Figure 3:* **VNP4 Example Design Test Bench**




The design phase, which would otherwise involve the detailed inner workings and interface connection specifications of the RTL modules, can be simplified to a few top-level VNP4 parameters and clocking decisions. The use of standard interfaces (AXI4-Stream and AXI4-Lite) simplifies the connection to other parts of the system. The user metadata structure also provides customization for custom side-band signals that are needed for interconnection by the user application.

One of the biggest savings when using VNP4 is the reduction in RTL and driver coding in the implementation phase. If the functionality can be described in P4 without user externs, then no RTL coding is required. The engineering effort saved in RTL coding is magnified for more complex P4 designs. This savings is further multiplied in cases of changing requirements, scope creep, and new features.

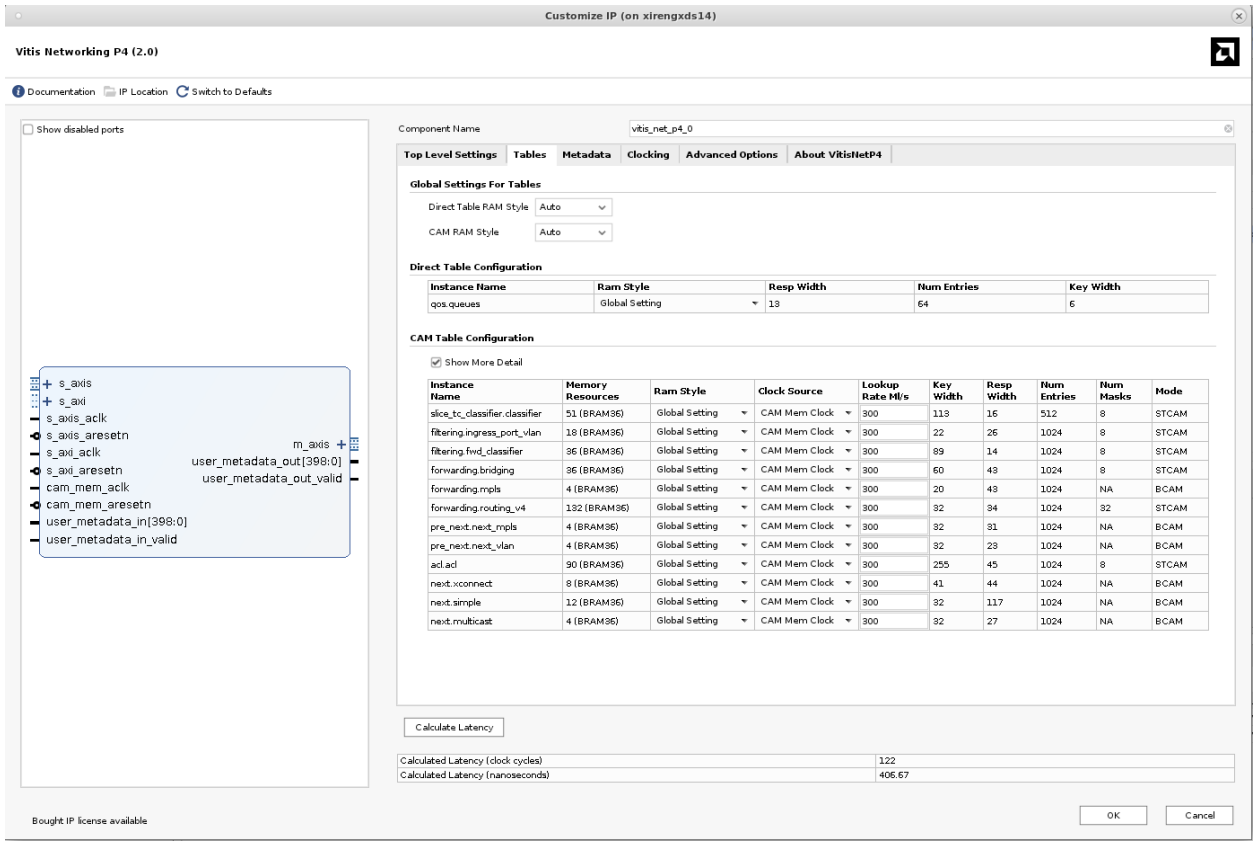
Similarly, the verification phase is also much shorter where there is little or no RTL test bench coding involved. The P4 code can be verified using the behavioral model. The runtime and iteration cycles are faster here compared to an RTL test bench. Detailed log information is provided by the model to indicate how each packet is processed by the P4 code step by step, allowing for easier debug compared to reviewing RTL waveforms.

Both flows require a system integration stage in either RTL or IP integrator. However, timing closure can be a significantly lower risk with the P4 flow. In the context of a hardware debug iteration cycle, this becomes even more pronounced. The P4 code can be quickly simplified (for example, reduced table size) to generate test bitstreams with a quicker, more reliable turnaround time, before later switching back to the full P4 functionality.

## Ease of Use

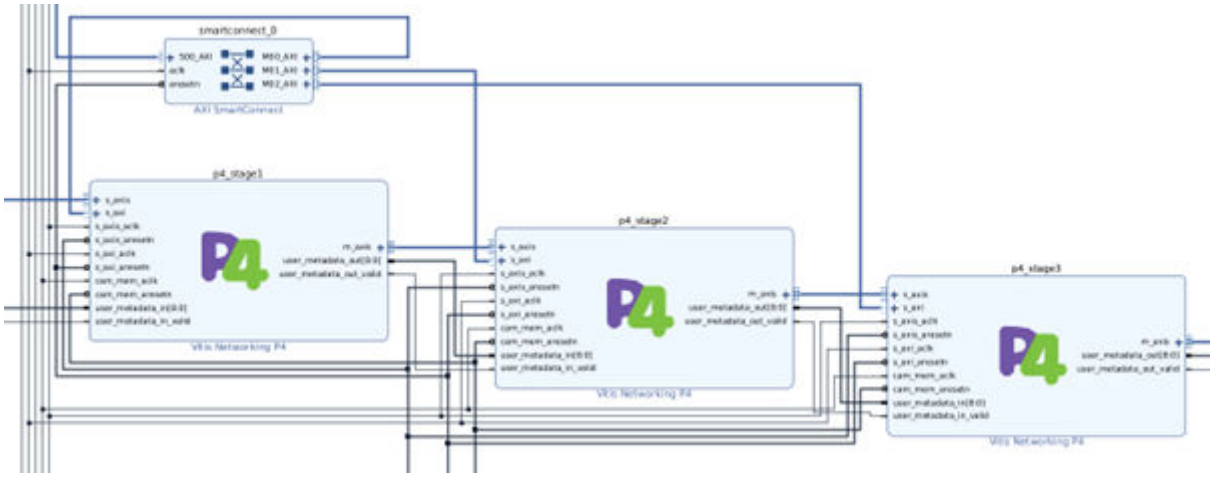
Technology parameters can be customized via a graphical user interface (GUI), which provides visual feedback, such as memory utilization for tables. The GUI displays the specific features and allowable parameter values tailored to the P4 program.

*Figure 4:* **VNP4 Customization GUI**



IP integrator can be used for connectivity between IPs, because VNP4 uses standard interfaces (AXI4-Stream and AXI4-Lite). These IPs can easily be connected in IP integrator, along with straightforward address mapping for AXI4-Lite.

*Figure 5:* **VNP4 within Vivado IP Integrator**




Typical configurations of the IP are supplied in the form of Example Designs. This allows you to see IP features operating in simulation and synthesis.

### Robust Designs

The use of correct-by-design code generation reduces the effort required for verification. Once the P4 code functionality has been verified using the P4 behavioral model, you can have confidence that the design intent in the form of P4 is properly converted into a working design ready for deployment in the FPGA or adaptive SoC.

## Rapid Prototyping and Time to Market

Getting your product to market is faster with the accelerated design cycle. Once the P4 is created, detailed information on the latency of the design and the memory requirements of the system are available. This aids in high-level design decisions such as device selection. Different options can be trialed. After initial decisions are made, the design can then be synthesized and brought through the back-end flow in the Vivado tools to assess the feasibility of the design in terms of resource allocation and timing closure. The P4 code can be verified much quicker than an RTL alternative approach, which provides earlier confidence in the design sizing information from the Vivado tools.

## Features

P4 provides extensive features to differentiate your product. These include options in user metadata and user externs.

### User Metadata

User metadata, inferred in the P4 code, provides side-band signaling alongside the packet data at the input to VNP4. It can be processed within the parser and the match-action pipeline, then it is output again alongside the packet data. One common use provides ingress and egress port number information to indicate where each packet is coming from and going to.

Targeting an FPGA or adaptive SoC means that VNP4 has the benefit of being able to scale the side-band signaling to any width to suit each application. The user metadata structure can be broken down into further struct definitions for convenient grouping of fields. It can be defined to align with other standard architectures (such as Portable NIC Architecture), but it is not restricted to any one of these definitions.

### User Externs

User externs provide an interface between the match-action control block and your own RTL module that resides outside of the VNP4. You can implement whatever function is useful for your design, such as a custom checksum calculation or a hash function.

## Migration

VNP4 allows for easy migration in several ways. Migrating to a different line rate is straightforward (for example, 100 Gb/s to 200 Gb/s), by scaling clock frequencies and bus widths in the GUI. No changes are required to the P4 code, and confidence can be maintained that the original implementation of the requirements remains faithful to the design intent. This can be very effective if developing a family of products with each family member targeting a specific line rate. The same P4 code generates the packet processing RTL in each family member, saving time.

Prototyping with smaller table implementations before moving to a larger number of table entries is also trivial to enable even more rapid prototyping through to hardware implementation. This can include starting off with on-chip SRAM before later increasing the size of the same P4 table to target off-chip DRAM. In a pure RTL design flow, this can be time consuming and introduce new risks. Any impact to the latency or performance of one table can have consequential impact on other parts of the P4 pipeline, for the whole design to remain in sync. However, VNP4 automatically manages all these latency and alignment challenges. More extensive changes in design are also enabled in cases where evolving functionality and requirements are supported through small P4 updates.

## Expansion

Designs can be deployed with multiple instances in parallel and multiple instances in series to support capabilities such as multi-level parsing and multi-data-pipeline systems. The flexible user metadata structure allows information to be passed between P4 instances to support this. There are several reasons to do this:

- To achieve higher total line rate processing or total packet rate

- To optimize more complex parts of the design that do not require high data rates or increased packet rates
- To create a more modular system, restricting the size of each P4 program. Modularity is also a re-use benefit.

*Figure 6:* **Multi-stage Ingress and Egress Pipelines**

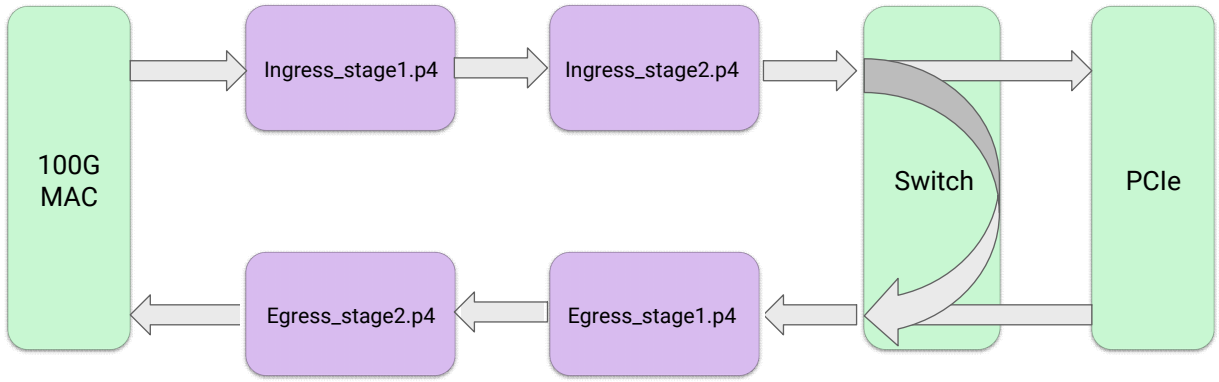
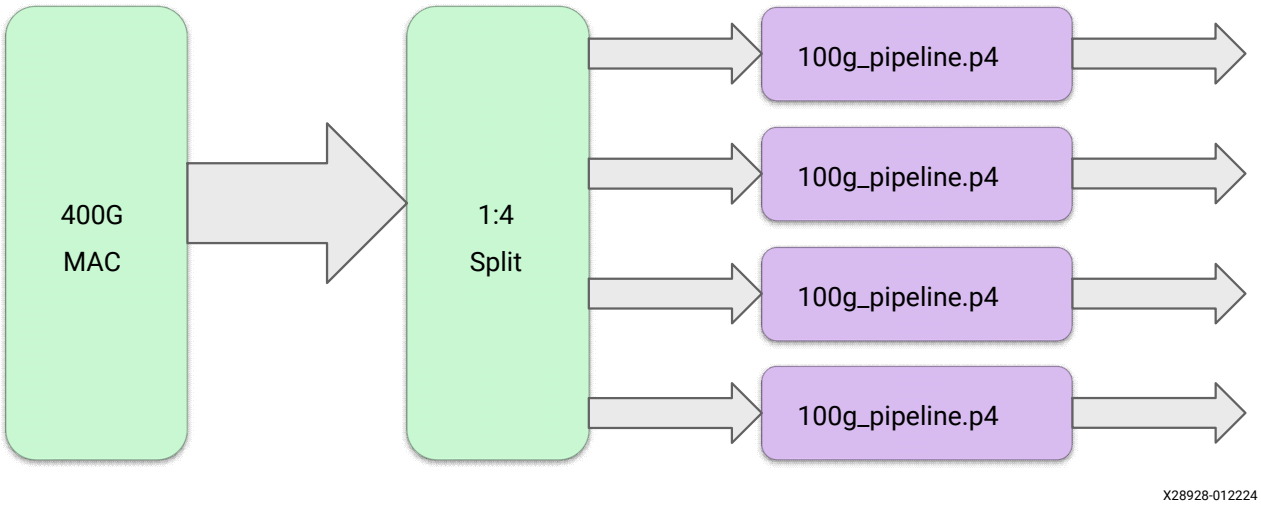



*Figure 7:* **4x100G Pipelines**




# Domain Specificity

This high-level abstraction solution is domain specific, allowing you to take advantage of abstraction without sacrificing performance. The number of degrees of freedom in the packet processing design space is orders of magnitude less than in a general-purpose data processing solution. This means the solution can be optimized for a more efficient implementation while maintaining the flexibility to implement arbitrary packet processing functions.

# Performance

The pipelined nature of the P4 designs allows VNP4 to process one packet every clock cycle. The only exception to this is for HBM/DDR binary CAM (BCAM) table look-ups, where the DRAM bandwidth might become a limiting factor. All elements of the design can scale in complexity without reducing the performance. This includes a complex header parsing tree, many different table look-ups and actions, and many packet editing operations. VNP4 does not set limitations on these complexities, for example:

- There is no fixed limit to the number of parsing states or header extracts
- There is no fixed limit to the number of headers that can be modified, removed, or inserted
- There is no fixed limit to the number of tables in the match-action block
- There is no fixed limit on the size of the user metadata

All elements can scale up to a large value without impacting the performance. Ultimately, designs reach a natural limit in terms of the device resource utilization (for example, if all block RAMs and UltraRAMs are exhausted). The performance is also not impacted by how deep the parsing goes into a packet.

The packet bus width and the clock frequency can be chosen to achieve the desired performance. The packet rate can also be configured to allow for further optimizations. Some common examples are shown in the following table.

*Table 1:* **Examples of Parameter Settings for Different Throughputs**

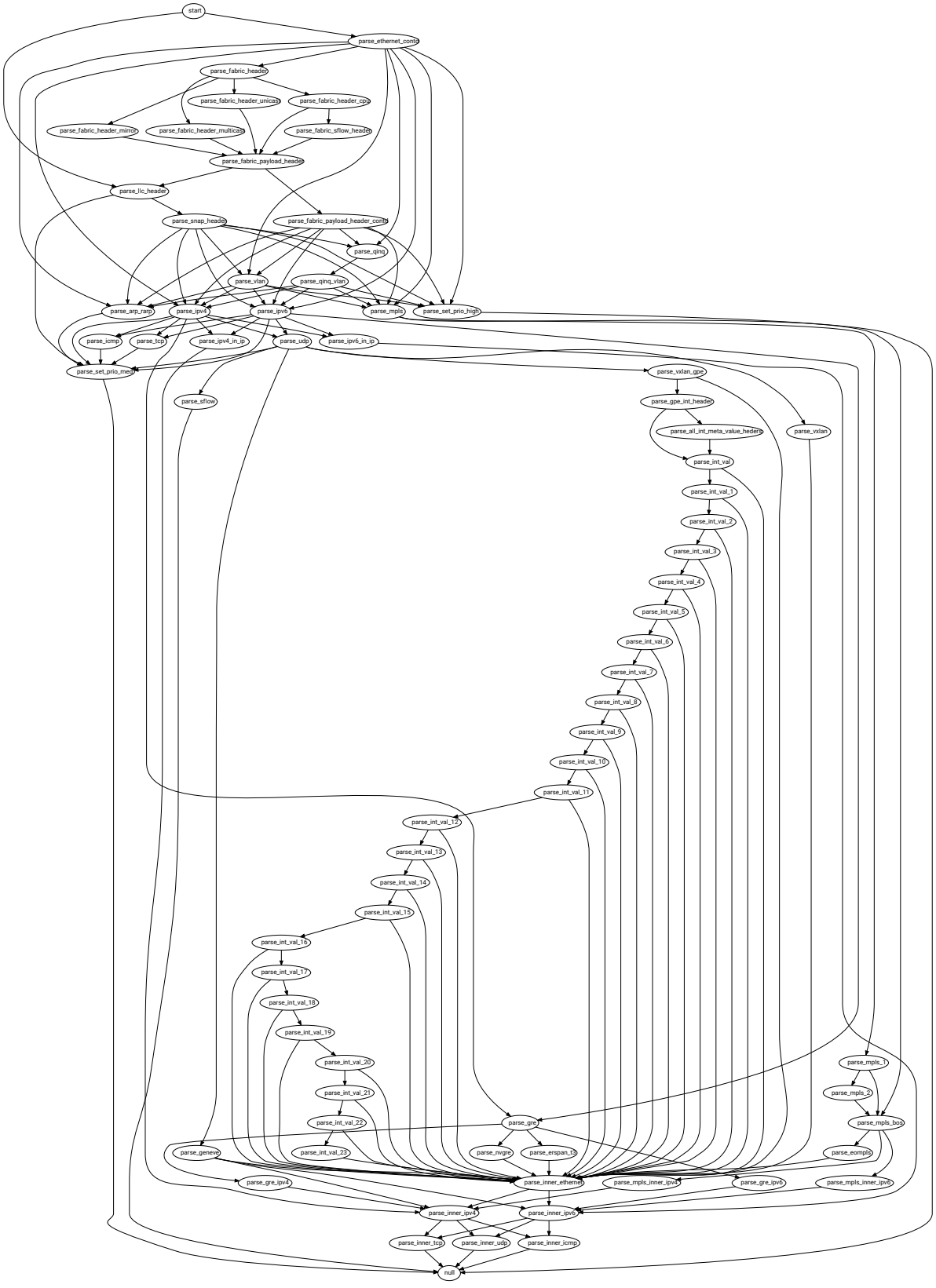
| Throughput (Gb/s) | Packet Rate Mp/s | Data Bus Width (Bytes) | Clock Frequency (MHz) |
|---|---|---|---|
| 200 | 300 | 128 | 336 |
| 100 | 150 | 64 | 300 |
| 50 | 75 | 32 | 300 |
| 10 | 15 | 4 | 312.5 |

***Note*:** Higher clock frequencies and packet rates can also be achieved; a trade-off is then needed between function complexity and timing closure.

## Resource Utilization

Very complex parsers can be supported in VNP4, while still operating at 200 Gb/s line rate and 300 million packets per second. To illustrate this point, a `consolidated switch` P4 example was taken and ported to the VNP4 XSA target, with the match-action section removed to focus on the parser. This example has 130,000 unique paths through the P4 parser (including error conditions), and it uses 31k LUTs. The example showcases the level of complexity in terms of parsing that can be enabled by VNP4. While a robust example, it is not the limit of the parsing complexity that can be supported in VNP4.

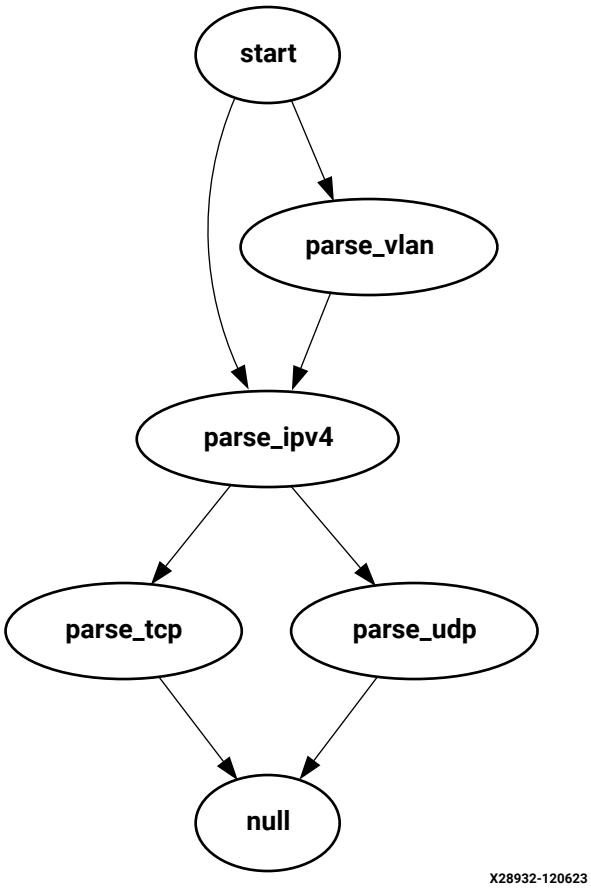
*Figure 8:* **Parse Graph for** `consolidated_switch_xsa.p4`




X28925-120523

For comparison, the parse graph of the less complicated `FiveTuple` example design available within the tool.

*Figure 9:* **Parse Graph for `FiveTuple.p4`**




Typically, the CAMs have a much larger utilization than the packet parsing and editing functions, along with other parts of the logic design outside VNP4, therefore designers can focus their efforts on system level trade-offs such as table entry numbers. Some other examples of resource utilization are given in the following section.

# P4 Examples

## P4 Language Tutorials

The P4 language tutorials [2] provide a set of 12 P4 programs that were created independently from VNP4. They target the BMv2 simple switch (v1model architecture). With a few updates, the programs can be modified to retarget the VNP4 pipeline architecture (XSA). The following table provides a summary of the designs along with device resource utilization numbers for those that are currently supported in VNP4. These designs were configured for a 100 Gb/s setup. The last

column demonstrates that for the larger designs, it is typically the P4 tables that dominate the logic utilization, rather than the packet parsing and editing. To put these utilization numbers in context, a 10G Ethernet MAC and PCS/PMA IP use approximately 10k LUTs [7]. Three of these P4 programs include language or extern features that are not yet supported in VNP4 (for example, the register extern).

***Note*:** Utilization numbers are based on 100 Gb/s setup, where TDATA_NUM_BYTES = 64 and the PKT_RATE = 150.

*Table 2:* **P4 Design Examples**

| P4 Program Name | Supported with VNP4 | LUTs Total | Flip-Flops | Block RAMs | UltraRAMs | Latency (Cycles) | Tables as % of LUTs |
|---|---|---|---|---|---|---|---|
| Basic Forwarding | Yes | 28339 | 43929 | 138 | 0 | 53 | 91% |
| Basic Tunnel | Yes | 30993 | 48291 | 146 | 0 | 54 | 88% |
| P4 Runtime | Yes | 35825 | 56894 | 158 | 32 | 83 | 77% |
| Explicit Congestion Notification | Yes | 28346 | 44070 | 138 | 0 | 53 | 90% |
| Multi-hop Route Inspection | Yes | 29715 | 46448 | 138 | 0 | 60 | 86% |
| Source Routing | Yes | 2675 | 5472 | 2 | 0 | 30 | 0% |
| Calculator | Yes | 2382 | 5208 | 3 | 0 | 24 | 5% |
| Load Balancing | No | | | | | | |
| Quality of Service | Yes | 28341 | 44000 | 138 | 0 | 53 | 90% |
| Firewall | Support for register Extern underway | | | | | | |
| Link Monitoring | Support for register Extern underway | | | | | | |
| MultiCast | Yes | 3154 | 5554 | 10 | 0 | 37 | 69% |

**Notes:**

1. Testing and analysis by AMD as of 11/24/23, using AMD Vivado™ Design Suite 2023.2 and an AMD Virtex™ UltraScale+™ device (xcvu37p-fsvh2892-2L-e), with out-of-context synthesis and implementation, and utilization numbers from a post-place utilization report. Actual results can vary. VIV-009.

## VNP4 Example Designs

The following table provides a summary of the device resource utilization numbers for the example designs that are released with VNP4. These example designs are primarily to showcase the various features of the P4 language that are supported in VNP4, rather than complete applications, but the resource numbers still highlight the efficiency of implementing various features.

***Note*:** Utilization numbers are based on 100 Gb/s setup, where $T_{DATA\_NUM\_BYTES}$ = 64 and the PKT_RATE = 150.

*Table 3:* **VNP4 Example Designs**

| P4 Program Name | LUTs (Total) | Flip-Flops | Block RAMs | UltraRAMs | Latency (Cycles) | Tables as % of LUTs |
|---|---|---|---|---|---|---|
| Echo | 3106 | 6784 | 2 | 0 | 26 | 0% |
| FiveTuple | 8807 | 15702 | 6 | 16 | 49 | 33% |
| FiveTuple_tinycam | 8605 | 15215 | 6 | 4 | 30 | 38% |
| Forward | 65251 | 85615 | 250 | 0 | 68 | 93% |
| Forward_tinycam | 11923 | 18680 | 2 | 0 | 32 | 65% |
| Calculator | 2542 | 5149 | 2 | 0 | 26 | 5% |
| Advanced Calculator | 2969 | 5814 | 3 | 0 | 59 | 4% |

**Notes:**

1. Testing and analysis by AMD as of 11/24/23, using AMD Vivado™ Design Suite 2023.2 and an AMD Virtex™ UltraScale+™ device (xcvu37p-fsvh2892-2L-e), with out-of-context synthesis and implementation, and utilization numbers from a post-place utilization report. Actual results can vary. VIV-009.

# Conclusion

Designing for high-speed packet processing in programmable logic can be challenging. VNP4 simplifies the process by using higher-level abstraction with the P4 language, without compromising on efficiency of resource utilization. For further information, download the Vivado Design Suite and select install Vitis Networking P4. Evaluation licenses are available from the product page, along with further documentation. You can also learn more about the P4 language at p4.org.

# References

These documents provide supplemental material useful with this guide:


1. *P4 Language Specification* https://p4.org/p4-spec/docs/p4-16-working-draft.html
2. *P4 Language Tutorials* https://github.com/p4lang/tutorials
3. GitHub, *Consolidated switch repository (API, SAI, and Nettlink)* p4lang/switch
4. Parveen Patel, Google, P4 Workshop 2023 Keynote: *P4 HAL for Network Virtualization* YouTube
5. Nick McKeown Fireside Chat, *2023 P4 Workshop* (YouTube)
6. Keynote: *The Power of Fully-Specified Data Planes*, Rob Sherwood (Intel), YouTube
7. Resource Utilization for 10G/25G Ethernet Subsystem v4.1
8. *Vitis Networking P4 User Guide* (UG1308)


# Revision History

The following table shows the revision history for this document.

| Section | Revision Summary |
|---|---|
| **1/24/2024 Version 1.0** | |
| Initial release. | N/A |

# Please Read: Important Legal Notices