# Streamline the AMD Vitis™ HLS Workflow with the Performance Pragma

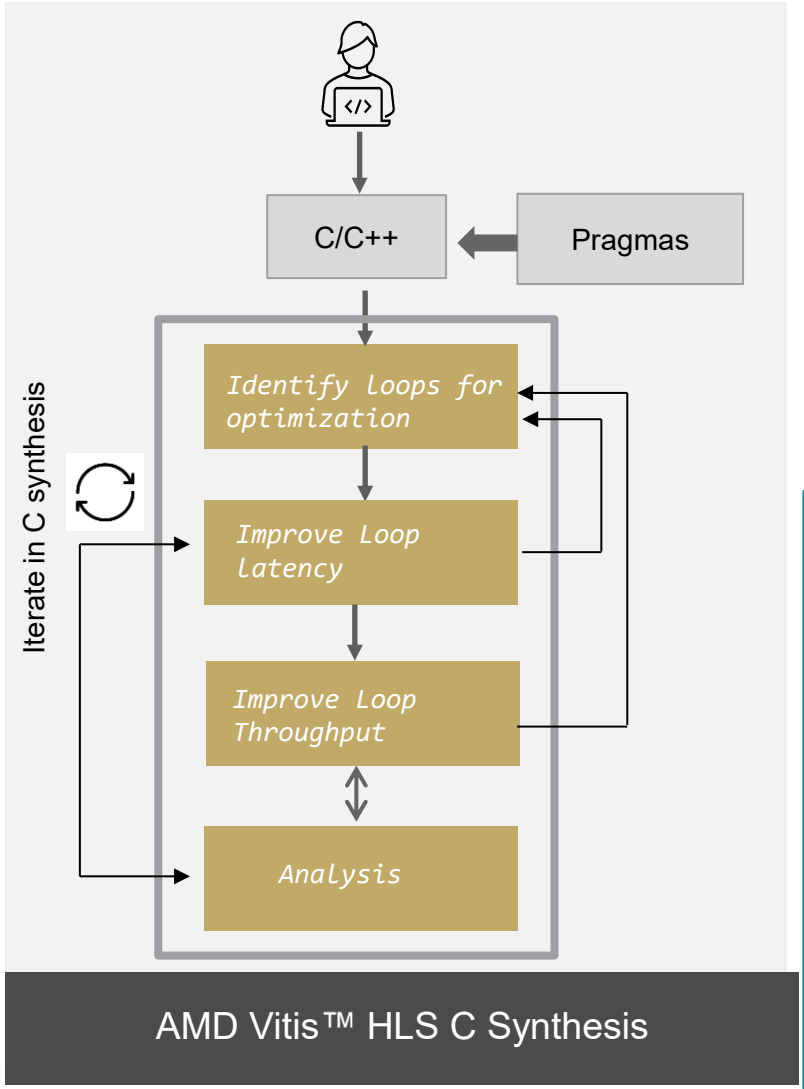
AMD Vitis HLS

AMD
together we advance_

# Agenda

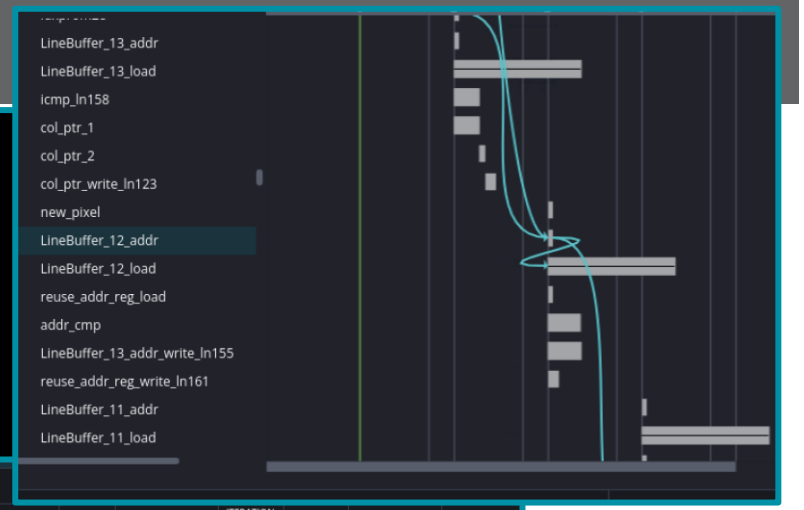
- Traditional HLS Optimization
- Challenges of Traditional HLS Optimization
- Simplifying Optimization with Performance Pragma
- Performance Pragma Methodology
  - Key differences with the traditional methodology
- Performance Pragma in Action: Convolution Design
- IDE Enhancements
- Summary & References
- Q&A

# Traditional HLS Optimization




- Optimization driven by manual pragma insertion
- Map system performance goals with pragmas inserted manually
- Top-level/system throughput inferred (not explicitly modeled)

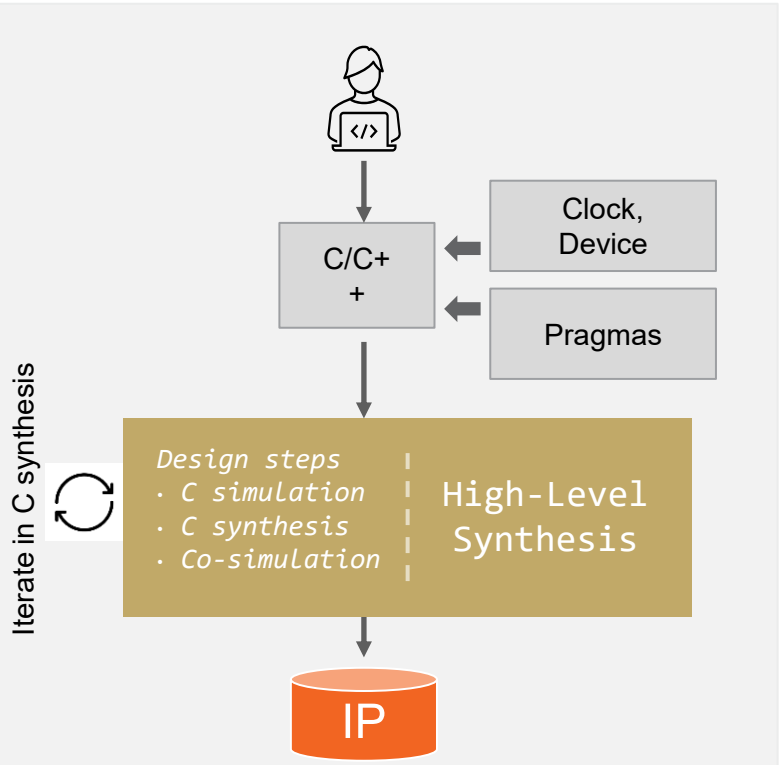
```
void xfv(...)
...
#pragma HLS ARRAY_PARTITION variable=row_ind complete
#pragma HLS ARRAY_PARTITION variable=OutputValues complete
#pragma HLS ARRAY_PARTITION variable=OutputValues1 complete
#pragma HLS ARRAY_PARTITION variable=src_buf complete
#pragma HLS ARRAY_PARTITION variable=src_buf complete
pragma HLS ARRAY_PARTITION variable=buf complete
for (...)
#pragma HLS pipeline
#pragma HLS LOOP_FLATTEN
for (col = 0; col<img_width>>+)
#pragma HLS UNROLL
for (..)
#pragma HLS UNROLL
```




| MODULES & LOOPS | ISSUE | LATENCY(CYCLES) | ITERATION LATENCY | INTERVAL | TRIP COUNT | PIPELINED |
|---|---|---|---|---|---|---|
| Filter2DKernel (5) | | 2,087,211 | | 2,087,068 | | dataflow |
| entry_proc | | 0 | | 0 | | no |
| ReadFromMem (2) | H | 2,074,007 | | 2,074,007 | | no |
| Window2D (1) | | 2,087,067 | | 2,087,067 | | no |
| Filter2D (2) | H | 2,073,858 | | 2,073,858 | | no |
| WriteToMem (1) | H | 2,073,676 | | 2,073,676 | | no |

***Optimization Driven by Manual Pragma Insertion!!!***

# Challenges of Traditional HLS Optimization




AMD Vitis™ HLS compiles C++ to RTL

- In the regular workflow, achieving performance goals **requires expertise in choosing the right combination of pragmas…**

- Pragma-based designs can be **less flexible and sensitive to changes**
  - Vision library color detection uses over 40+ classic pragmas
    - unroll, pipeline, flatten…
  - Any changes to throughput goals could affect these 40+ pragmas...

***Choosing the Right Combination of Pragmas can be a Challenge!!!***

# Simplifying Optimization with Performance Pragma

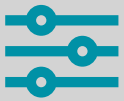
AMD Performance Pragma simplifies HLS optimization

Allows users to define a high-level throughput goal

Shifts manual pragma selection optimization burden to the compiler

Enables Automatic Pragma Generation: No more manual pragma guessing!


Intelligently infers and applies optimizations (pipelining, unrolling, etc.)



Offers flexible throughput control via target specification

Tool automatically determines optimal pragma configuration

Represents a new, higher-level way to constrain design throughput

Provides a more intuitive and efficient path to desired performance

# Performance Pragma

Performance Pragma can be applied to a top-level function and/or individual loops

## Top-Level Performance Pragma

- Defines a design-wide throughput goal
- Guides the compiler to optimize the entire design
- Automatically infers and applies loop-level pragmas based on analysis

## Loop-Level Performance Pragma

- Targets specific loops for local control
- Can be automatically inferred based on top-level performance pragma or manually applied
- Enables fine-grain optimization, infers classic pragmas (pipeline, unroll, etc...)

## Benefits

New top-level performance goal representation

Design-wide analysis to validate performance goals

Helps achieve top-level performance pragma targets

Precise control of loop behavior

# Performance Pragma Methodology

Offers a streamlined approach to guide the HLS optimization process to achieve optimal performance

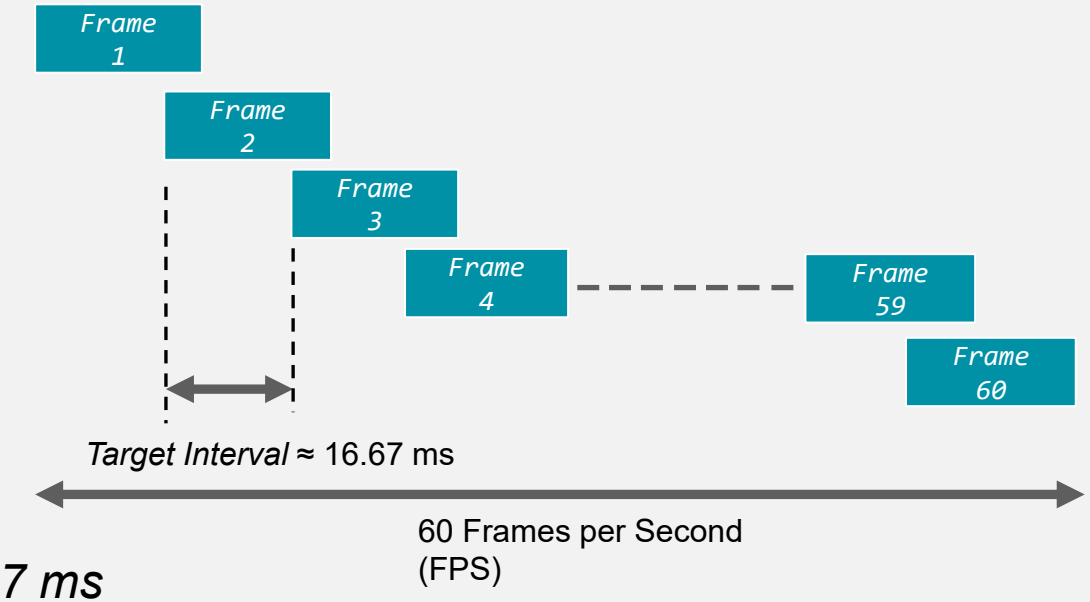
**Step 1** → **Step 2** → **Step 3** → **Step 4**

Consider a video application aiming for a frame rate of 60 frames per second (60 fps)

**Determine the top-level performance target (`target_ti`)**

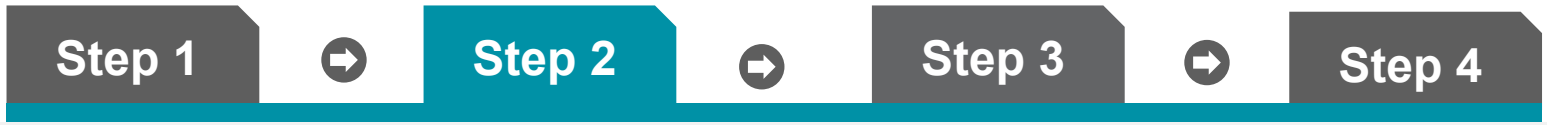
- To achieve the 60 FPS, the top-level function must be ready to process a new frame within 1/60th of a second
- Implies:

*target_ti = 1 / throughput(FPS) = 1 / 60 seconds ≈ 16.67 ms*



# Performance Pragma Methodology

Offers a streamlined approach to guide the HLS optimization process to achieve optimal performance

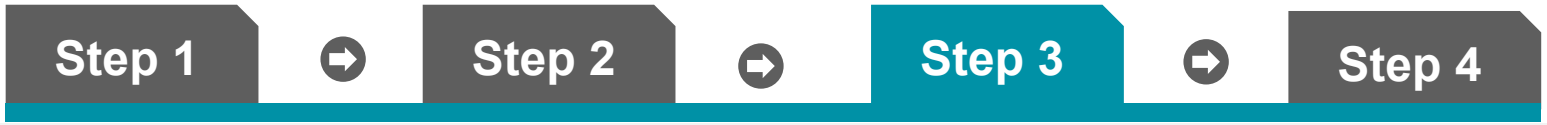



**Re-architect the Code for Dataflow**

- Design necessitates a re-architecture for an explicit load-compute-store (LCS) modeling compatible with the dataflow pragma
- Allows AMD Vitis™ HLS to effectively:
  - Optimize the code
  - Exploit task-level parallelism

# Performance Pragma Methodology

Offers a streamlined approach to guide the HLS optimization process to achieve optimal performance in the user's hardware implementation

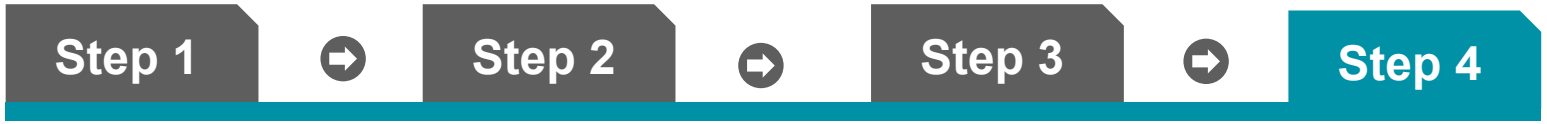



## Run C Simulation and Determine Loop Trip Counts

- That metric is crucial since the performance pragma algorithm requires loop budgeting…
- By default, variable loop bounds are assumed to be "1024," which can be inaccurate…
- For variable loop bounds, provide trip count information via the loop_tripcount max=N pragma…

# Performance Pragma Methodology

Offers a streamlined approach to guide the HLS optimization process to achieve optimal performance in the user's hardware implementation




## Add the Top-Level Performance Target

- Apply the desired performance goal using the top-level performance pragma

  #pragma HLS performance target_ti = 16.67 ms

## Add/Update Local Performance Targets

- Identify performance-critical loops after C synthesis
- Specify loop-level performance targets for these critical loops
- Iteratively refine & re-synthesize until the design meets performance

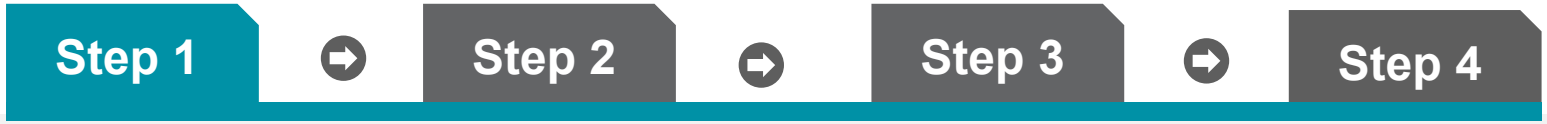
# Performance Pragma Methodology: Key Differences

Comparing the traditional approach (manual pragma insertion) to the top-level performance pragma workflow…

| Top-Level Throughput Constraint | Needs Trip Count for Dynamic Loops | May also need Loop-Level Performance Pragma |
| --- | --- | --- |
| Unlike starting optimization at individual loop level, you define a system-wide performance target first | Provides the tool with crucial information for accurate performance estimation, especially for loops with variable iterations | While the tool automates, you can still fine-tune specific bottlenecks for more granular control |

# Performance Pragma in Action: Convolution Design

# Convolution 2D: Calculate the Performance Target




## Calculate the Performance Target Based on Throughput Goal

- Goal: Convolution function process an HD 140 frames per second @ 300 MHz clock…

### Determine the Top-Level Performance Target (target_ti)

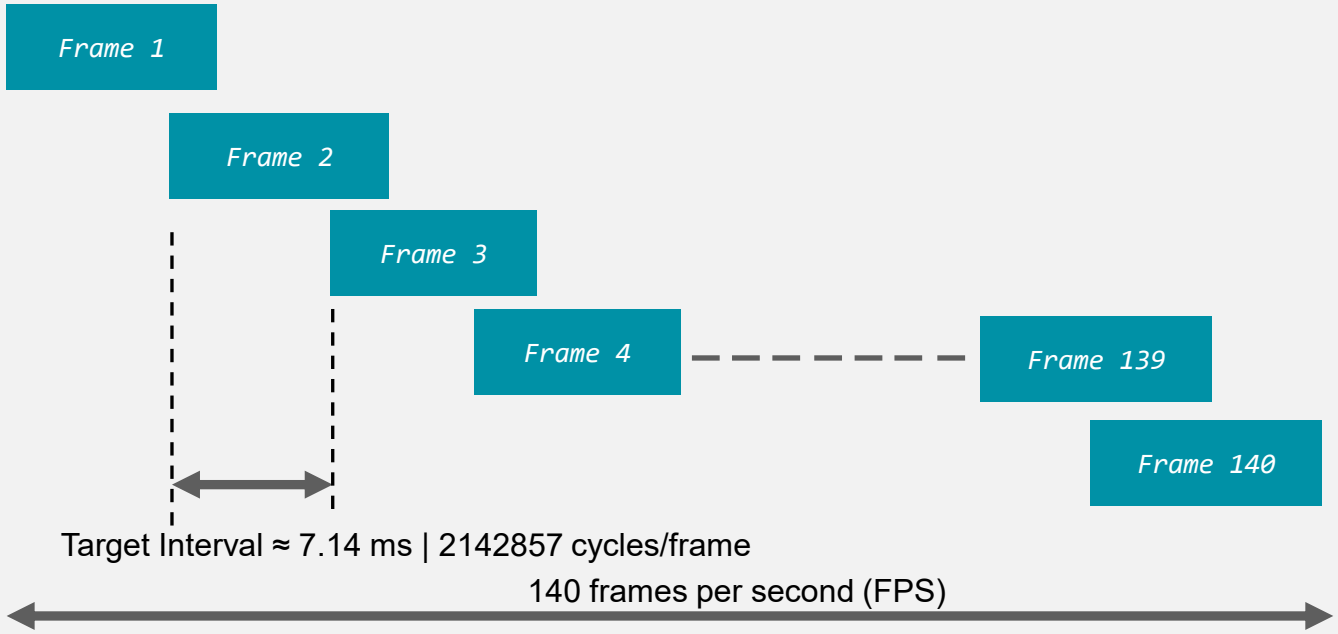
*Target Interval (target_ti)* can be expressed in time (ms) *or* number of cycles:

- Time: 140 frames/sec hence: `target_ti`

  (milliseconds/frame) = 1000 /140 = `7.14` ms
- Cycles: 140 frames/sec at 300 MHz hence:
  - `target_ti` = (kernel freq.) / (throughput) =

    (300 * $10^6$ cycles/second) /140 =

    `2,142,857.14` cycles/frame

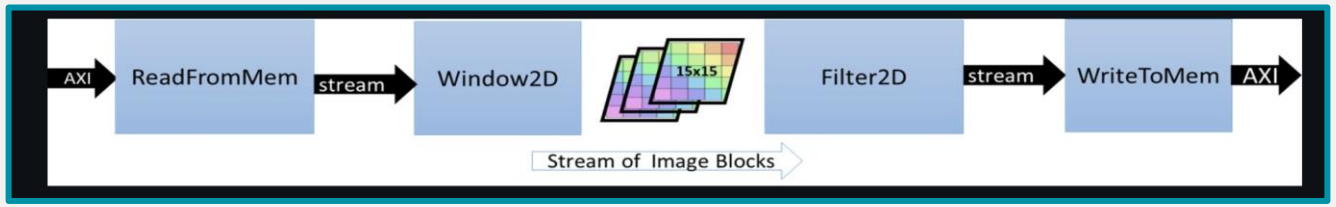


# Convolution 2D: Re-architect the Code for Dataflow

Step 1 → Step 2 → Step 3 → Step 4

## Re-architect the Code for Dataflow

- Design necessitates a re-architecture to make load-compute-store explicit and apply the dataflow pragma




```
void Filter2DKernel(
        const char           coeffs[256],
        float                factor,
        short                bias,
        unsigned short       width,
        unsigned short       height,
        unsigned short       stride,
        const unsigned char  src[MAX_IMAGE_WIDTH*MAX_IMAGE_HEIGHT],
        unsigned char        dst[MAX_IMAGE_WIDTH*MAX_IMAGE_HEIGHT])
  {

#ifdef STEP1
#pragma HLS performance target_ti = 2142857
#endif
#pragma HLS dataflow
```

# Convolution 2D: Run C Simulation with Code Analyzer



**Run C Simulation and Determine Loop Trip Counts**

- Run C simulation with Code Analyzer
- Add trip counts for dynamic variable loops

```
        for (int x = 0; x < width; x++)
        {
#pragma HLS LOOP_TRIPCOUNT max=1920
            // Read a 2D window of pixels
            window w = window_stream.read();
```

# Convolution 2D: Add the Top/Loop-Level Performance Target (1/4)

Step 1 → Step 2 → Step 3 → Step 4

**Add the Top-Level Performance Target**

- Apply the top-level performance pragma using target_ti

```
void Filter2DKernel(
        const char           coeffs[256],
        float                factor,
        short                bias,
        unsigned short       width,
        unsigned short       height,
        unsigned short       stride,
        const unsigned char  src[MAX_IMAGE_WIDTH*MAX_IMAGE_HEIGHT],
        unsigned char        dst[MAX_IMAGE_WIDTH*MAX_IMAGE_HEIGHT])
  {

#ifdef STEP1
#pragma HLS performance target_ti = 2142857
#endif
#pragma HLS dataflow
```

# Convolution 2D: Add the Top/Loop-Level Performance Target (2/4)

Step 1 → Step 2 → Step 3 → Step 4

## Identify Bottleneck Loops (if any)

- Here Window2D does not meet the performance target of 2,142,857.14 cycles/frame

| MODULES & LOOPS | ISSUE | TARGET TI(CYCLES) | ESTIMATED TI(CYCLES) |
|---|---|---|---|
| Filter2DKernel (5) | | 2,142,857 | 45,915,038 |
| entry_proc | | | |
| ReadFromMem (2) | | 2,073,873 | 2,074,007 |
| Window2D (1) | | 532,196,988 | 45,915,037 |
| update_window (2) | | 532,196,985 | 45,915,034 |
| Outline_VITIS_LOOP_141_1 | | 1 | 1 |
| Window2D_Pipeline_VITIS_LOOP_149_3 (1) | | | |
| Filter2D (2) | | 575,394 | 2,073,858 |
| WriteToMem (1) | | 2,073,612 | 2,073,676 |

```
void Window2D(
        unsigned short        width,
        unsigned short        height,
        hls::stream<U8>      &pixel_stream,
        hls::stream<window>  &window_stream)
{
    // Line buffers - used to store [FILTER_V_SIZE-1] enti
    U8 LineBuffer[FILTER_V_SIZE-1][MAX_IMAGE_WIDTH];


    // Sliding window of [FILTER_V_SIZE][FILTER_H_SIZE] pi
    window Window;

    unsigned col_ptr = 0;
    unsigned ramp_up = width*((FILTER_V_SIZE-1)/2)+(FILTER_
    unsigned num_pixels = width*height;
    unsigned num_iterations = num_pixels + ramp_up;
    const unsigned max_iterations = MAX_IMAGE_WIDTH*MAX_IM

    // Iterate until all pixels have been processed
    update_window: for (int n=0; n<num_iterations; n++)
    {
        #pragma HLS LOOP_TRIPCOUNT max=2087047
```

# Convolution 2D: Add the Top/Loop-Level Performance Target (3/4)

Step 1 → Step 2 → Step 3 → Step 4

**Add/Update Local Performance Targets**

- Apply loop-level performance pragmas
  - Specify loop-level performance pragmas for the Window2D function to achieve target_ti

```
    update_window: for (int n=0; n<num_iterations; n++)
    {
        #pragma HLS LOOP_TRIPCOUNT max=2087047

#ifdef STEP2
        MET=no TARGET=2142857, ESTIMATE=45915034 (CYCLES)
        #pragma HLS performance target_ti = 2142857
#endif
```

# Convolution 2D: Add the Top/Loop-Level Performance Target (4/4)

## Target Still Not Reached

| MODULES & LOOPS | ISSUE | TARGET TI(CYCLES) | ESTIMATED TI(CYCLES) | PERFORMANCE CONSTRAINT |
|---|---|---|---|---|
| Filter2DKernel (5) | | 2,142,857 | 45,915,038 | pragma |
| entry_proc | | | | |
| ReadFromMem (2) | | 2,073,873 | 2,074,007 | |
| Window2D (1) | | 532,196,988 | 45,915,037 | |
| update_window (2) | | 2,142,857 | 45,915,034 | pragma |
| Outline_VITIS_LOOP_141_1 | | 1 | 1 | |
| Window2D_Pipeline_VITIS_LOOP_149_3 (1) | | | | |
| VITIS_LOOP_149_3 | | 26 | 13 | inferred |
| Filter2D (2) | | 575,394 | 2,073,858 | |
| WriteToMem (1) | | 2,073,612 | 2,073,676 | |

*C-synth report with top and loop-level Performance pragma*

## Solution

Set a target loop TI = 1 for this shift loop to nudge the compiler to execute the loop in a single cycle

```
        for(int i = 0; i < FILTER_V_SIZE-2; i++) {
#ifdef STEP3
                    #pragma HLS performance target_ti= 1
#endif
            LineBuffer[i][col_ptr] = LineBuffer[i+1][col_ptr];
        }
```

## Voila!! Meets performance!

# Convolution 2D: Results

**Goal**

2,142,857.14 cycles/frame
or 140 frames per second

**Achieved**

2,087,651 cycles/frame or
~144 frames per second

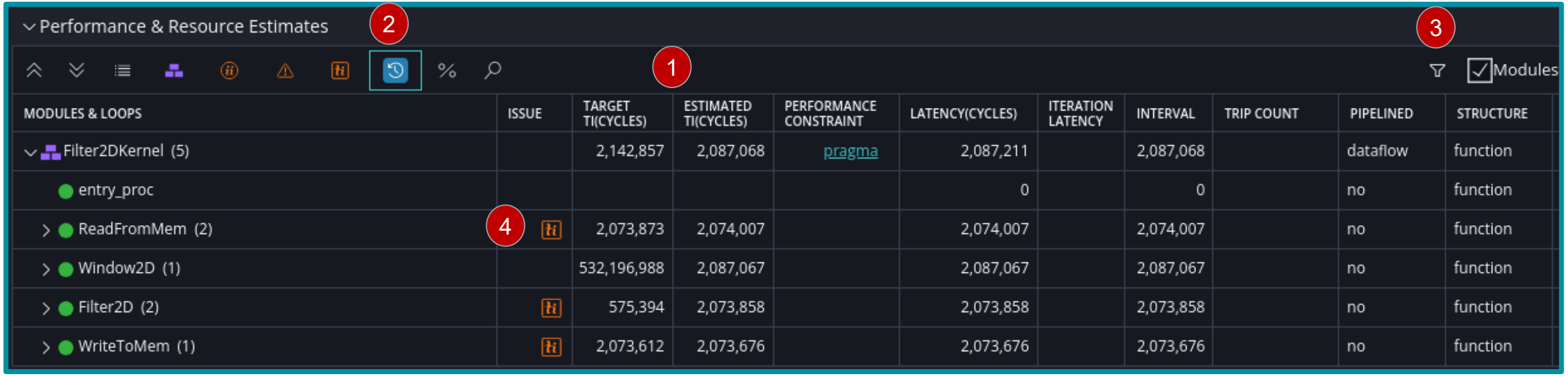
| Category | Using Classic Pragmas | Using Performance Pragma |
|---|---|---|
| Target Interval | 2,087,651 | 2,087,651 |
| Optimizations Pragmas in the Design | 8 | 3 (2.6X fewer pragmas) |

# Limitations of Performance Pragma in v2025.1

| Features | Behavior/Limitation |
|---|---|
| ap_cint | The tool exits with an explicit warning message |
| ap_(u)int / ap_(u)fixed | • No performance models are inaccurate |
| Big constant arrays of ap_int / ap_fixed | • Using them with macro NON_C99STRING will result in a compilation error |
| std::complex<ap_fixed> | Lead to a compiler error on Windows |
| HLS IP blocks (FFT, FIR, ...), hls_math.h, ap_wait, hls::vector, ap_axis/ap_axiu | • No performance models are inaccurate |
| hls::stream_of_blocks, hls::task, hls::split / hls::merge, hls::print, hls::half, ap_utils.h, hls_fpo.h, ap_float, RTL Blackboxes, OpenCL,<br>hls::burst_maxi, hls::fence, hls::directio | The tool exits with an explicit warning message |
| Deprecated HLS pragmas | Assertion failure |
| Function pipeline with sub loop(s) | Assertion failure |

# IDE Enhancements

# Redesigned Table for Synthesis Report



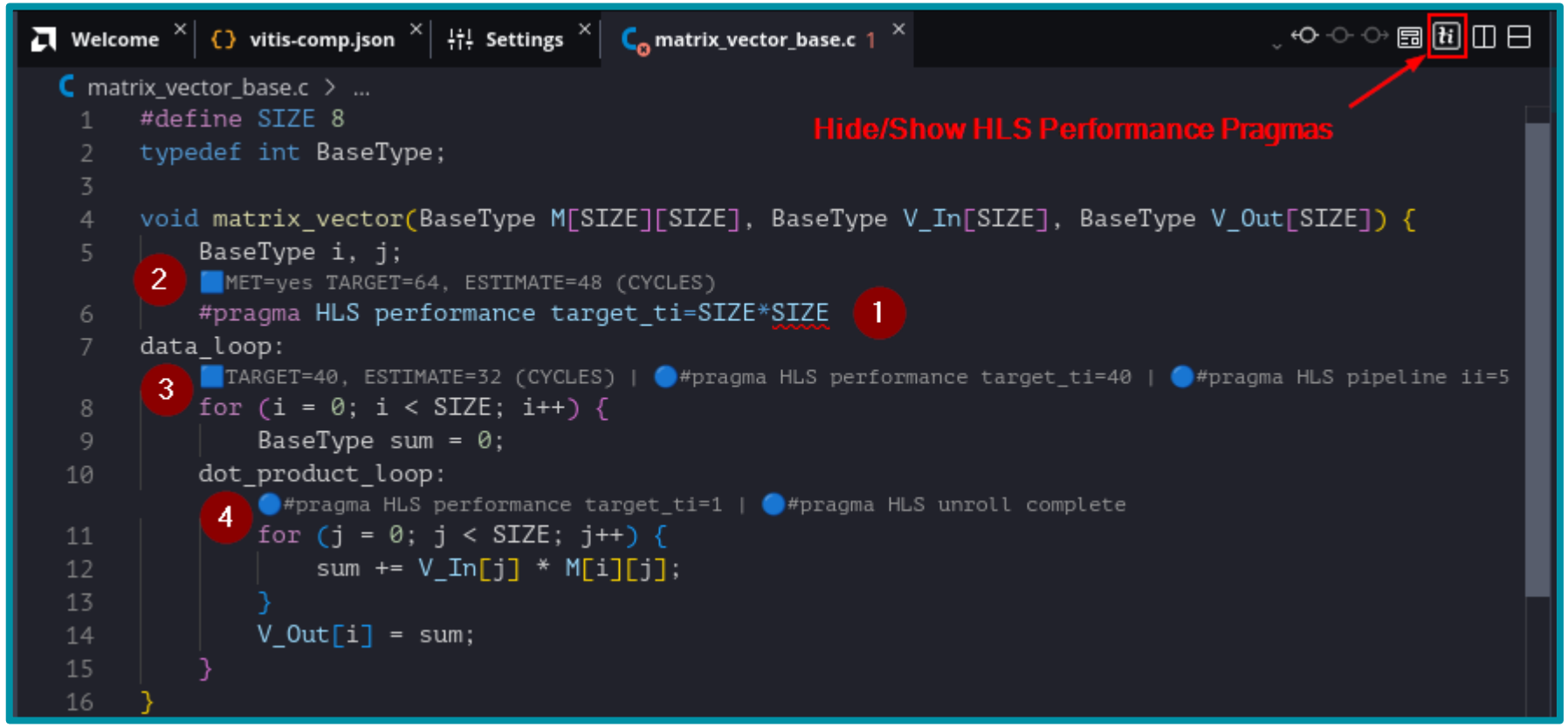

| MODULES & LOOPS | ISSUE | TARGET TI(CYCLES) | ESTIMATED TI(CYCLES) | PERFORMANCE CONSTRAINT | LATENCY(CYCLES) | ITERATION LATENCY | INTERVAL | TRIP COUNT | PIPELINED | STRUCTURE |
|---|---|---|---|---|---|---|---|---|---|---|
| Filter2DKernel (5) | | 2,142,857 | 2,087,068 | pragma | 2,087,211 | | 2,087,068 | | dataflow | function |
| entry_proc | | | | | 0 | | 0 | | no | function |
| ReadFromMem (2) | 4 | 2,073,873 | 2,074,007 | | 2,074,007 | | 2,074,007 | | no | function |
| Window2D (1) | | 532,196,988 | 2,087,067 | | 2,087,067 | | 2,087,067 | | no | function |
| Filter2D (2) | | 575,394 | 2,073,858 | | 2,073,858 | | 2,073,858 | | no | function |
| WriteToMem (1) | | 2,073,612 | 2,073,676 | | 2,073,676 | | 2,073,676 | | no | function |

1. Updated table in response to new performance pragma
2. Richer report visualization: Instantly toggle cycles/ns for clearer insights
3. Greater filtering capability
4. Icons and colorization for csynth_design results condensation

*Quickly perform detailed performance and resource analysis with new table*

# Pragma Overlays in Source Editor

- (1) - HLS performance pragma (top-level function)
- (2) – (4) – Inferred pragmas from (1)




*Review optimizations made by the performance pragma…*

# Summary

Performance Pragma simplifies complex HLS optimization by enabling users to define a high-level throughput goal, shifting the optimization burden to the compiler for automatic pragma generation and application of key transformations, offering flexible throughput control and a more efficient path to desired hardware performance.

# Q&A

# References

1. Vitis™ HLS Introductory Examples
2. Vitis HLS Doc
3. Vitis HLS Product Page - Vitis HLS
4. Vitis HLS tutorial

# Disclaimers

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.


AMD