



# Lab2 Intro

## Improving Performance

# Objectives

- ▶ After completing this lab, you will be able to:
  - Add directives to your design
  - Observe the effect of PIPELINE-ing functions
  - Improve the performance using various directives

# The Design

- ▶ The design consists of YUV filter typically used in video processing. The design consists of three functions – rgb2yuv, yuv\_scale, and yuv2rgb
  - Each of these functions iterates over the entire source image, requiring a single source pixel to produce a pixel in the result image
  - The scale function simply applies individual scale factors, supplied through top-level arguments

```
// The top-level function
void yuv_filter (
    image_t *in,
    image_t *out,
    yuv_scale_t Y_scale,
    yuv_scale_t U_scale,
    yuv_scale_t V_scale
)
{
    // Internal image buffers
    #ifndef __SYNTHESIS__
        image_t *yuv = (image_t *)malloc(sizeof(image_t));
        image_t *scale = (image_t *)malloc(sizeof(image_t));
    #else // Workaround malloc() calls w/o changing rest of code
        image_t _yuv;
        image_t _scale;
        image_t *yuv = &_yuv;
        image_t *scale = &_scale;
    #endif

    rgb2yuv  (in, yuv);
    yuv_scale (    yuv, scale, Y_scale, U_scale, V_scale);
    yuv2rgb  (    scale, out);
}
```

# Procedure

- ▶ Create a Vitis HLS project by executing script from the terminal
- ▶ Open the created project in Vitis HLS GUI and analyze
- ▶ Apply TRIPCOUNT directive using PRAGMA
- ▶ Apply PIPELINE directive, generate solution, and analyze output
- ▶ Apply DATAFLOW directive to improve performance
- ▶ Export and Implement the design

# Summary

- ▶ In this lab you learned that even though this design could not be pipelined at the top-level, a strategy of pipelining the individual loops and then using dataflow optimization to make the functions operate in parallel was able to achieve the same high throughput, processing one pixel per clock.
- ▶ When DATAFLOW directive is applied, the default memory buffers (of ping-pong type) are automatically inserted between the functions. Using the fact that the design used only sequential (streaming) data accesses allowed the costly memory buffers associated with dataflow optimization to be replaced with simple 2 element FIFOs using the Dataflow command configuration



# Thank You

## Disclaimer and Attribution

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's products are as set forth in a signed agreement between the parties or in AMD's Standard Terms and Conditions of Sale. GD-18

© Copyright 2021 Advanced Micro Devices, Inc. All rights reserved. Xilinx, the Xilinx logo, AMD, the AMD Arrow logo, Alveo, Artix, Kintex, Kria, Spartan, Versal, Vitis, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

