ECE4810J System-on-Chip (SoC) Design

Fall 2021

Lab #4 Optimizing Performance through Pipelining

Due: November 4th 11:59pm, 2021

Logistics:

- This lab is a team exercise.
- Please use the discussion board on Piazza for Q&A.
- All reports and code (if available) MUST be submitted to the assignment of Canvas.
- Internet usage is allowed and encouraged.
- No late submission is allowed for this lab.
- All necessary files for this lab have been uploaded to canvas.

1. Overview

This is a lab for exercising the HLS flow on Zynq using Vivado. The goals of this lab are:

- Understand the effect of INLINE directive
- Improve performance using PIPELINE directive
- Distinguish between DATAFLOW directive and Configuration Command functionality

2. Optimizing Performance through Pipelining

2.1 Create a Vivado HLS Project from Command Line

Validate your design using Vivado HLS command line mode. Create a new Vivado HLS project from the command line.

1. Select **Start > Xilinx Design Tools > Vivado HLS 2018.2 Command Prompt**.
2. In the Vivado HLS Command Prompt window, change directory to your lab directory, example: **c:\xup\hls\labs\lab4**. A self-checking program (yuv_filter_test.c) is provided. Using that we can validate the design. A Makefile is also provided. Using the Makefile, the necessary source files can be compiled and the compiled program can be executed. You can examine the contents of these files and the project directory.
3. In the Vivado HLS Command Prompt window, type **make** to compile and execute the program. (You might need to set up the system environment variable for make command)

Vivado HLS 2018.2 Command Prompt

```
C:\xup\hls\labs\lab2>make
gcc -lm   yuv_filter.o yuv_filter_test.o image_aux.o   -o yuv_filter
./yuv_filter
Test passed!

C:\xup\hls\labs\lab2>
```

Note that the source files (yuv_filter.c, yuv_filter_test.c, and image_aux.c) were compiled, then yuv_filter executable program was created, and then it was executed. The program tests the design and outputs Test Passed message. A Vivado HLS tcl script file (pynq_yuv_filter.tcl) is provided and can be used to create a Vivado HLS project.

4. Type **vivado_hls –f pynq_yuv_filter.tcl** in the Vivado HLS Command Prompt window to create the project targeting xc7z020clg400-1 part. The project will be created and Vivado HLS.log file will be generated.
5. Open the **vivado_hls.log** file from your directory (example: *c:\xup\hls\labs\lab4)* using any text editor and observe the following sections:

- Creating directory and project called yuv_filter.prj within it, adding design files to the project, setting solution name as solution1, setting target device, setting desired clock period, and importing the design and testbench files.
- Synthesizing (Generating) the design which involves scheduling and binding of each functions and sub-function.
- Generating RTL of each function and sub-function in SystemC, Verilog, and VHDL languages.

6. Open the created project (in GUI mode) from the Vivado HLS Command Prompt window, by typing **vivado_hls –p yuv_filter.prj**. The Vivado HLS will open in GUI mode and the project will be opened.

```
****** Vivado(TM) HLS - High-Level Synthesis from C, C++ and SystemC v2018.2 (64-bit)
  **** SW Build 2258646 on Thu Jun 14 20:03:12 MDT 2018
  **** IP Build 2256618 on Thu Jun 14 22:10:49 MDT 2018
   ** Copyright 1986-2018 Xilinx, Inc. All Rights Reserved.

source C:/Xilinx/Vivado/2018.2/scripts/vivado_hls/hls.tcl -notrace
INFO: [HLS 200-10] Running 'C:/Xilinx/Vivado/2018.2/bin/unwrapped/win64.o/vivado_hls.exe'
INFO: [HLS 200-10] For user 'RIHL' on host 'rihl' (Windows NT_amd64 version 6.2) on Wed Aug 01 22:38:07 +0800 2018
INFO: [HLS 200-10] In directory 'C:/xup/hls/labs/lab2'
INFO: [HLS 200-10] Opening and resetting project 'C:/xup/hls/labs/lab2/yuv_filter.prj'.
INFO: [HLS 200-10] Adding design file 'yuv_filter.c' to the project
INFO: [HLS 200-10] Adding test bench file 'image_aux.c' to the project
INFO: [HLS 200-10] Adding test bench file 'yuv_filter_test.c' to the project
INFO: [HLS 200-10] Adding test bench file 'test_data' to the project
INFO: [HLS 200-10] Opening and resetting solution 'C:/xup/hls/labs/lab2/yuv_filter.prj/solution1'.
INFO: [HLS 200-10] Cleaning up the solution database.
INFO: [HLS 200-10] Setting target device to 'xc7z020clg400-1'
INFO: [SYN 201-201] Setting up clock 'default' with a period of 10ns.
INFO: [HLS 200-10] Analyzing design file 'yuv_filter.c' ...
INFO: [HLS 200-111] Finished Linking Time (s): cpu = 00:00:01 ; elapsed = 00:00:05 . Memory (MB): peak = 102.652 ; gain = 45.297
INFO: [HLS 200-111] Finished Checking Pragmas Time (s): cpu = 00:00:01 ; elapsed = 00:00:05 . Memory (MB): peak = 102.652 ; gain = 45.297
INFO: [HLS 200-10] Starting code transformations ...
```

```
24 INFO: [HLS 200-10] Starting code transformations ...
25 INFO: [HLS 200-111] Finished Standard Transforms Time (s): cpu = 00:00:01 ; el
26 INFO: [HLS 200-10] Checking synthesizability ...
27 INFO: [HLS 200-111] Finished Checking Synthesizability Time (s): cpu = 00:00:0
28 INFO: [XFORM 203-602] Inlining function 'yuv_scale' into 'yuv_filter' (yuv_fil
29 INFO: [XFORM 203-401] Performing if-conversion on hyperblock from (yuv_filter.
30 INFO: [XFORM 203-11] Balancing expressions in function 'rgb2yuv' (yuv_filter.c
31 INFO: [HLS 200-111] Finished Pre-synthesis Time (s): cpu = 00:00:02 ; elapsed
32 INFO: [HLS 200-111] Finished Architecture Synthesis Time (s): cpu = 00:00:02 ;
33 INFO: [HLS 200-10] Starting hardware synthesis ...
34 INFO: [HLS 200-10] Synthesizing 'yuv_filter' ...
35 INFO: [HLS 200-10] ----------------------------------------------------------
36 INFO: [HLS 200-42] -- Implementing module 'rgb2yuv'
37 INFO: [HLS 200-10] ----------------------------------------------------------
38 INFO: [SCHED 204-11] Starting scheduling ...
39 WARNING: [SCHED 204-21] Estimated clock period (10.283ns) exceeds the target (
40 WARNING: [SCHED 204-21] The critical path consists of the following:
41     'mul' operation ('tmp_25', yuv_filter.c:57) (3.36 ns)
42     'add' operation ('tmp3', yuv_filter.c:57) (3.02 ns)
43     'add' operation ('tmp_26', yuv_filter.c:57) (3.9 ns)
44 INFO: [SCHED 204-11] Finished scheduling.
45 INFO: [HLS 200-111]  Elapsed time: 15.045 seconds; current allocated memory: 9
46 INFO: [BIND 205-100] Starting micro-architecture generation ...
47 INFO: [BIND 205-101] Performing variable lifetime analysis.
48 INFO: [BIND 205-101] Exploring resource sharing.
49 INFO: [BIND 205-101] Binding ...
50 INFO: [BIND 205-100] Finished micro-architecture generation.
51 INFO: [HLS 200-111]  Elapsed time: 0.105 seconds; current allocated memory: 95
52 INFO: [HLS 200-10] ----------------------------------------------------------
53 INFO: [HLS 200-42] -- Implementing module 'yuv2rgb'
54 INFO: [HLS 200-10] ----------------------------------------------------------
55 INFO: [SCHED 204-11] Starting scheduling ...
56 WARNING: [SCHED 204-21] Estimated clock period (10.8454ns) exceeds the target
57 WARNING: [SCHED 204-21] The critical path consists of the following:
58     'mul' operation ('tmp_12', yuv_filter.c:101) (3.36 ns)
59     'add' operation ('tmp1', yuv_filter.c:101) (3.02 ns)
60     'add' operation ('tmp_14', yuv_filter.c:101) (2.14 ns)
61     'icmp' operation ('icmp9', yuv_filter.c:101) (0.959 ns)
62     'select' operation ('p_phitmp2', yuv_filter.c:101) (0 ns)
63     'select' operation ('G', yuv_filter.c:101) (1.37 ns)
64 INFO: [SCHED 204-11] Finished scheduling.
```

2.2 Analyze the Created Project and Results

Open the source file and note that three functions are used. Look at the results and observe that the latencies are undefined (represented by ?).

1. In Vivado HLS GUI, expand the source folder in the *Explorer* view and double click **yuv_filter.c** to view the content.

- The design is implemented in 3 functions: **rgb2yuv**, **yuv_scale** and **yuv2rgb**.
- Each of these filter functions iterates over the entire source image (which has maximum dimensions specified in image_aux.h), requiring a single source pixel to produce a pixel in the result image.
- The scale function simply applies individual scale factors, supplied as top-level arguments to the Y'UV components.
- Notice that most of the variables are of user-defined (typedef) and aggregate (e.g. structure, array) types.
- Also notice that the original source used malloc() to dynamically allocate storage for the internal image buffers. While appropriate for such large data structures in software, malloc() is not synthesizable and is not supported by Vivado HLS.
- A viable workaround is conditionally compiled into the code, leveraging the **SYNTHESIS** macro. Vivado HLS automatically defines the **SYNTHESIS** macro when reading any code. This ensure the original malloc() code is used outside of synthesis but Vivado HLS will use the workaround when synthesizing.

2. Expand the **syn > report** folder in the *Explorer* view and double-click yuv_filter_csynh.rpt entry to open the synthesis report.
3. Each of the loops in this design has variable bounds – the width and height are defined by members of input type image_t. When variables bounds are present on loops the total latency of the loops cannot be determined: this impacts the ability to perform analysis using reports. Hence, **"?"** is reported for various latencies.

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
?	?	?	?	none

2.3 Apply TRIPCOUNT Pragma

Open the source file and uncomment pragma lines, re-synthesize, and observe the resources used as well as estimated latencies. Answer the questions listed in the detailed section of this step.

1. To assist in providing loop-latency estimates, Vivado HLS provides a TRIPCOUNT directive which allows limits on the variables bounds to be specified by the user. In this design, such directives have been embedded in the source code, in the form of #pragma statements.

2. Uncomment the #pragma lines (50, 53, 90, 93, 130, 133) to define the loop bounds and save the file.

3. Synthesize the design by selecting **Solution > Run C Synthesis > Active Solution**. View the synthesis report when the process is completed.

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
841205	51621125	841205	51621125	none

4. Scroll the *Console* window and note that yuv_scale function is automatically inline into the yuv_filter function.

```
INFO: [HLS 200-10] Checking synthesizability ...
INFO: [HLS 200-111] Finished Checking Synthesizability Time (s): cpu = 00:00:01 ; elapsed = 00:00:11 . Memory (MB): peak = 104.156 ; gain = 47.289
INFO: [XFORM 203-602] Inlining function 'yuv_scale' into 'yuv_filter' (yuv_filter.c:24) automatically.
INFO: [XFORM 203-401] Performing if-conversion on hyperblock from (yuv_filter.c:92:33) to (yuv_filter.c:92:27) in function 'yuv2rgb'... converting 7 basic blocks.
INFO: [XFORM 203-11] Balancing expressions in function 'rgb2yuv' (yuv_filter.c:30)...11 expression(s) balanced.
INFO: [HLS 200-111] Finished Pre-synthesis Time (s): cpu = 00:00:01 ; elapsed = 00:00:12 . Memory (MB): peak = 125.930 ; gain = 69.063
```

5. Observe that there are three entries – rgb2yuv.rpt, yuv_filter.rpt, and yuv2rgb.rpt under the syn report folder in the Explorer view. There is no entry for yuv_scale.rpt since the function was inlined into the yuv_filter function. You can access lower level module's report by either traversing down in the top-level report under components (under Utilization Estimates > Details > Component) or from the reports container in the project explorer.

6. Expand the **Summary** of loop latency and note the latency and trip count numbers for the yuv_scale function. Note that the YUV_SCALE_LOOP_Y loop latency is 6x the specified TRIPCOUNT, implying that 6 cycles are used for each of the iteration of the loop.

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
841205	51621125	841205	51621125	none

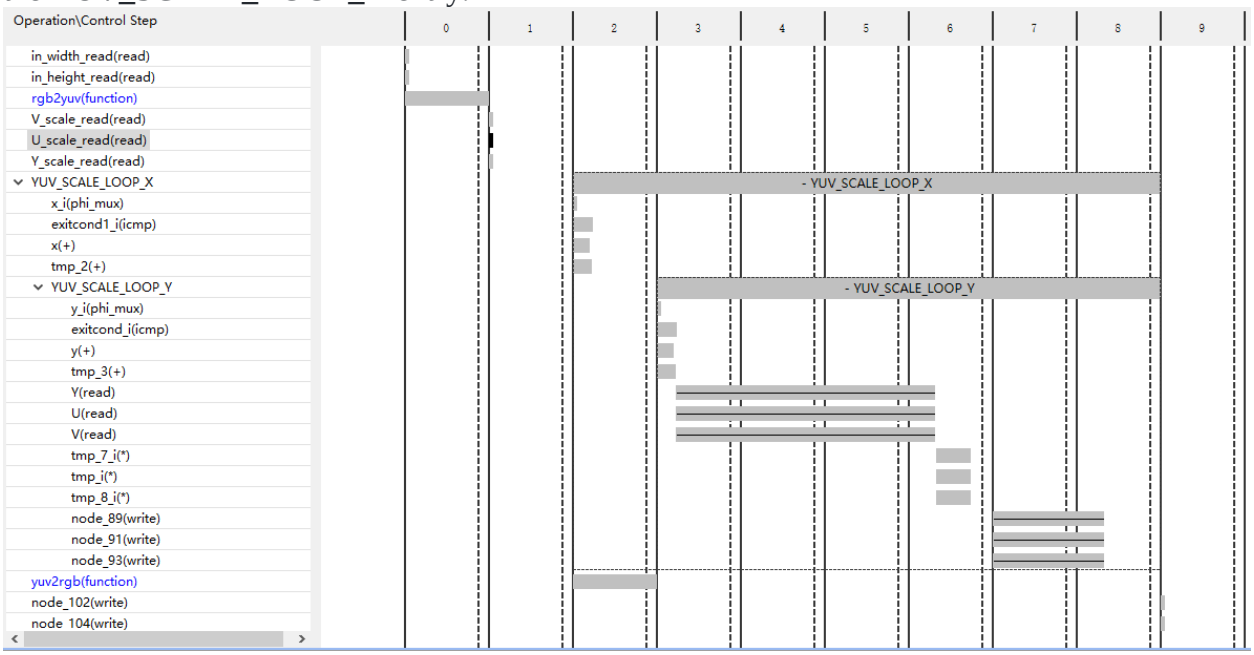
Detail

Instance

Loop

	Latency			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- YUV_SCALE_LOOP_X	240400	14749440	1202 ~ 7682	-	-	200 ~ 1920	no
+ YUV_SCALE_LOOP_Y	1200	7680	6	-	-	200 ~ 1280	no

7. You can verify this by opening an analysis perspective view, expanding the **YUV_SCALE_LOOP_X** entry, and then expanding the **YUV_SCALE_LOOP_Y** entry.



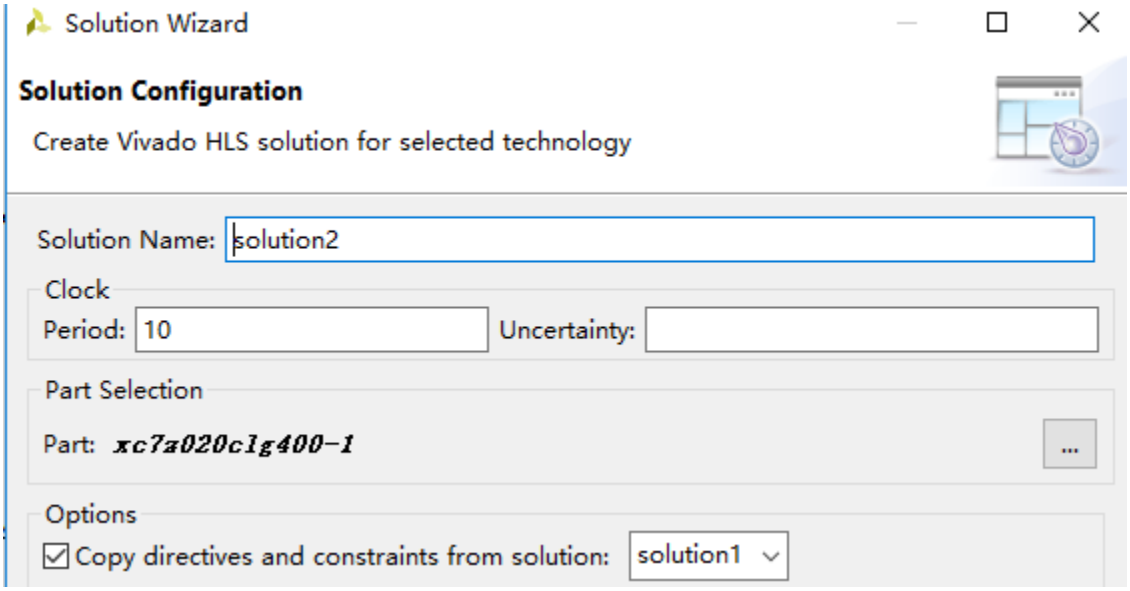
8. In the report tab, expand **Detail > Instance** section of the *Utilization Estimates* and click on the **grp_rgb2yuv_fu_244 (rgb2yuv)** entry to open the report.

9. Similarly, open the *yuv2rgb* report.

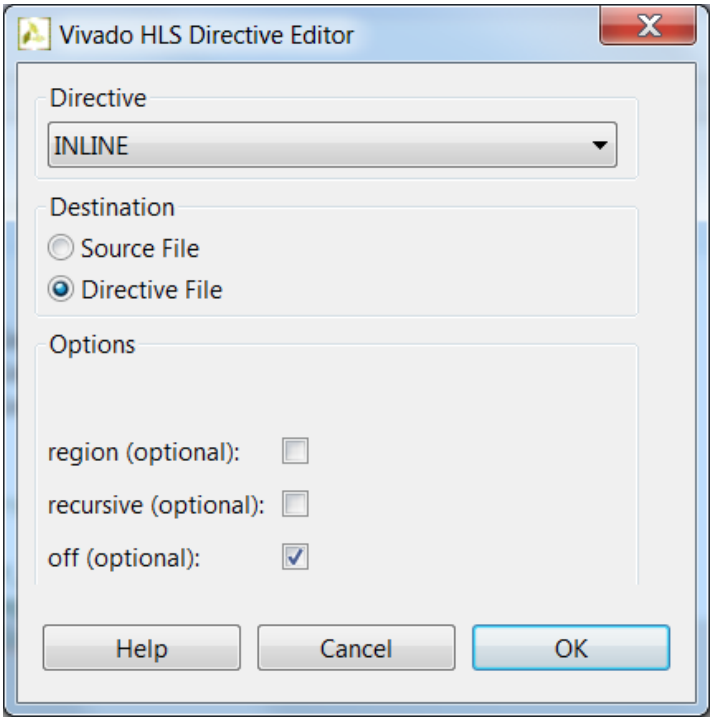
2.4 Turn OFF INLINE and Apply PIPELINE Directive

Create a new solution by copying the previous solution settings. Prevent the automatic INLINE and apply PIPELINE directive. Generate the solution and understand the output.

1. Select **Project > New Solution** or click on the button from the tools bar buttons.
2. A *Solution Configuration* dialog box will appear. Note that the check boxes of *Copy directives and constraints from solution* are checked with *solution1* selected. Click the **Finish** button to create a new solution with the default settings.



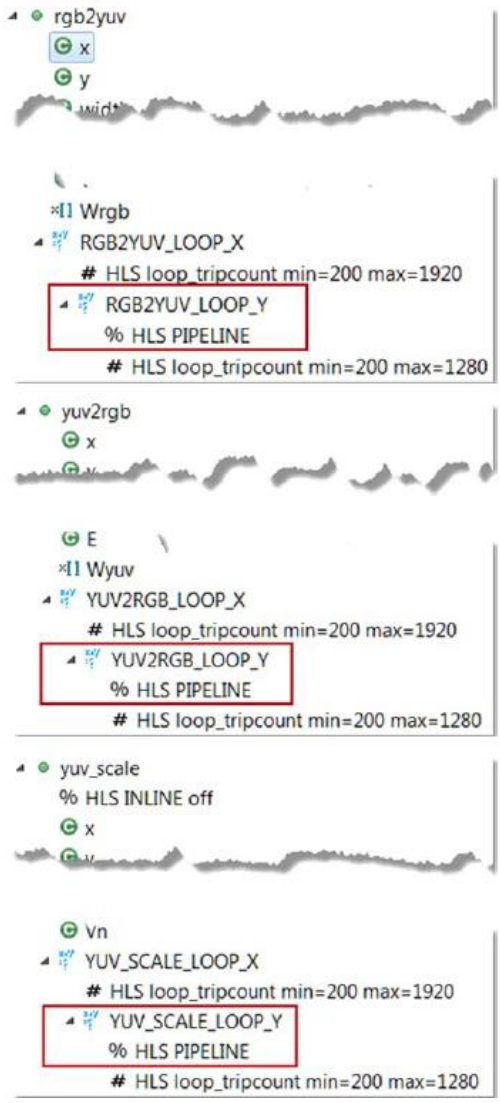
3. Make sure that the **yuv_filter.c** source is opened and visible in the information pane, and click on the **Directive** tab.
4. Select function **yuv_scale** in the directives pane, right-click on it, and select **Insert Directive...**
5. Click on the drop-down button of the *Directive* field. A pop-up menu shows up listing various directives. Select **INLINE** directive.
6. In the *Vivado HLS Directive Editor* dialog box, click on the **off** option to turn off the automatic inlining. Make sure that the *Directive File* is selected as destination. Click **OK**.



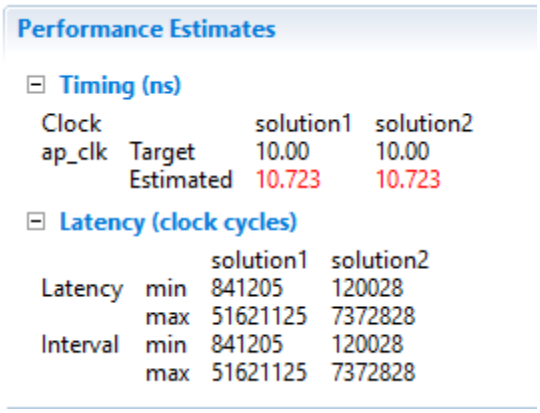
- When an object (function or loop) is pipelined, all the loops below it, down through the hierarchy, will be automatically unrolled.
- In order for a loop to be unrolled it must have fixed bounds: all the loops in this design have variable bounds, defined by an input argument variable to the top-level function.
- Note that the TRIPCOUNT directive on the loops only influences reporting, it does not set bounds for synthesis.
- Neither the top-level function nor any of the sub-functions are pipelined in this example.
- The pipeline directive must be applied to the inner-most loop in each function – the innermost loops have no variable-bounded loops inside which are required to be unrolled and the outer loop will simply keep the inner loop fed with data.

7. Expand the **yuv_scale** in the *Directives* tab, right-click on **YUV_SCALE_LOOP_Y** object and select **insert directives …**, and select **PIPELINE** as the directive.
8. Leave *II* (Initiation Interval) blank as Vivado HLS will try for an II=1, one new input every clock cycle.
9. Click **OK**.

10. Similarly, apply the **PIPELINE** directive to **YUV2RGB_LOOP_Y** and **RGB2YUV_LOOP_Y** objects. At this point, the *Directive* tab should look like as follows.



11. Click on the **Synthesis** button.
12. When the synthesis is completed, select **Project > Compare Reports…** to compare the two solutions.
13. Select *Solution1* and *Solution2* from the **Available Reports**, and click on the **Add>>** button.
14. Observe that the latency reduced.



Performance Estimates

Timing (ns)

Clock		solution1	solution2
ap_clk	Target	10.00	10.00
	Estimated	10.723	10.723

Latency (clock cycles)

		solution1	solution2
Latency	min	841205	120028
	max	51621125	7372828
Interval	min	841205	120028
	max	51621125	7372828

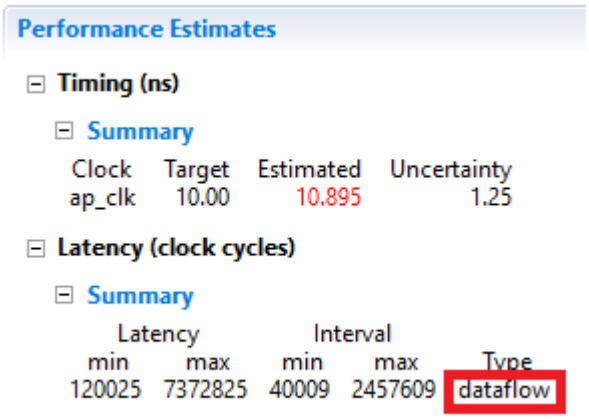
In Solution1, the total loop latency of the inner-most loop was loop_body_latency x loop iteration count, whereas in Solution2 the new total loop latency of the inner-most loop is loop_body_latency + loop iteration count.

15. Scroll down in the comparison report to view the resources utilization. Observe that the FFs, LUTs, and DSP48E utilization increased whereas BRAM remained same.

2.5 Apply DATAFLOW Directive and Configuration Command

Create a new solution by copying the previous solution (Solution2) settings. Apply DATAFLOW directive. Generate the solution and understand the output.

1. Select **Project > New Solution** or click on button from the tools bar.
2. A *Solution Configuration* dialog box will appear. Click the **Finish** button (with copy from Solution2 selected).
3. Close all inactive solution windows by selecting **Project > Close Inactive Solution Tabs**.
4. Make sure that the **yuv_filter.c** source is opened in the information pane and select the *Directive* tab.
5. Select function yuv_filter in the *Directive* pane, right-click on it and select **Insert Directive...**
6. A pop-up menu shows up listing various directives. Select **DATAFLOW** directive and click **OK**.
7. Click on the **Synthesis** button.
8. When the synthesis is completed, the synthesis report is automatically opened.
9. Observe additional information, **Dataflow** Type, in the *Performance Estimates* section is mentioned.



Performance estimate after DATAFLOW directive applied

- The Dataflow pipeline throughput indicates the number of clocks cycles between each set of inputs reads. If this throughput value is less than the design latency it indicates the design can start processing new inputs before the currents input data are output.
- While the overall latencies haven't changed significantly, the dataflow throughput is showing that the design can achieve close to the theoretical limit (1920x1280 = 2457600) of processing one pixel every clock cycle.

10. Scrolling down into the *Utilization Estimates* section, observe that the number of BRAMs required has doubled. This is due to the default ping-pong buffering in dataflow.

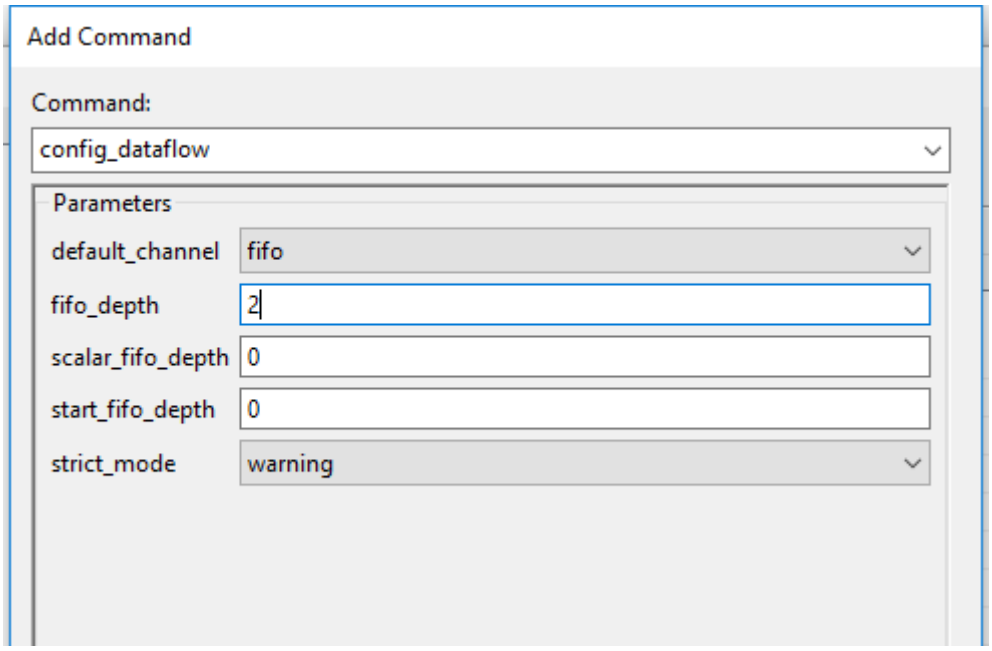
- When **DATAFLOW** optimization is performed, memory buffers are automatically inserted between the functions to ensure the next function can begin operation before the previous function has finished. The default memory buffers are ping-pong buffers sized to fully accommodate the largest producer or consumer array.
- Vivado HLS allows the memory buffers to be the default **ping-pong** buffers or **FIFOs**. Since this design has data accesses which are fully sequential, FIFOs can be used. Another advantage to using FIFOs is that the size of the FIFOs can be directly controlled (not possible in ping-pong buffers where random accesses are allowed).

11. The memory buffers type can be selected using Vivado HLS Configuration command.

Apply Dataflow configuration command, generate the solution, and observe the improved resources utilization.

1. Select **Solution > Solution Settings…** to access the configuration command settings.

2. In the *Configuration Settings* dialog box, select **General** and click the **Add…** button.
3. Select **config_dataflow** as the command using the drop-down button and **fifo** as the default_channel. Enter **2** as the fifo_depth. Click OK.



Selecting Dataflow configuration command and FIFO as buffer

4. Click **OK** again.
5. Click on the **Synthesis** button.
6. When the synthesis is completed, the synthesis report is automatically opened.
7. Note that the performance parameter has not changed; however, resource estimates show that the design is not using any BRAM and other resources (FF, LUT) usage has also reduced.

2.6 **Export and Implement the Design in Vivado HLS**

In Vivado HLS, export the design, selecting VHDL as a language, and run the implementation by selecting Evaluate option.

1. In Vivado HLS, select **Solution > Export RTL** or click on the button on tools bar to open the dialog box so the desired implementation can be run. An Export RTL Dialog box will open.

2. Click on the drop-down button of the **Evaluate Generated RTL** field and select **VHDL** as the language and click on the **Vivado synthesis, place and route** check box underneath.

3. Click **OK** and the implementation run will begin. You can observe the progress in the Vivado HLS Console window. When the run is completed the implementation report will be displayed in the information pane.

4. Close Vivado HLS by selecting **File > Exit**.

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.

References:

1. Arty Z7 Reference Manual https://digilent.com/reference/programmable-logic/arty-z7/reference-manual
2. https://xilinx-wiki.atlassian.net/wiki/spaces/A/overview
3. https://pynq.readthedocs.io/en/v2.4/pynq_overlays/partial_reconfiguration.html
4. https://www.hackster.io/90432/programming-python-on-zynq-fpga-ec4712
5. https://github.com/Xilinx/PYNQ-HelloWorld

Acknowledgement:

- Sergiu Masanu, Mircea Stan (U of Virginia)
- Andreas Gerstlauer (U of Austin)
- Digilent
- Xilinx PYNQ project
- Xilinx University Program

Deliverables:

- Group Deliverables (Compile everything as a single pdf report besides the code):
  - Based on **part 2**, please submit
    - In **2.3**, Step 3, Answer the following question pertaining to yuv_filter function.
      Estimated clock period:
      Worst case latency:
      Number of DSP48E used:
      Number of BRAMs used:
      Number of FFs used:
      Number of LUTs used:
    - In **2.3**, Step 8, Answer the following question pertaining to rgb2yuv function.
      Estimated clock period:
      Worst case latency:
      Number of DSP48E used:
      Number of BRAMs used:
      Number of FFs used:
      Number of LUTs used:
    - In **2.3**, Step 9, Answer the following question pertaining to yuv2rgb function.
      Estimated clock period:
      Worst case latency:
      Number of DSP48E used:
      Number of BRAMs used:
      Number of FFs used:
      Number of LUTs used:
    - In 2.4, Step 15, please include a screenshot of the resources utilization.
    - In 2.5, Step 10, please include a screenshot of the Utilization Estimates.
    - In 2.5, Step 7 (the last step), please include a screenshot of synthesis report
    - In 2.6, please include a screenshot of your implementation report.
  - Answer the following questions (based on your understanding, feel free to use the internet):
    - Does the pipelining approach employed in this lab apply to all designs? If not, why?

- In general, how do you identify the performance bottleneck? (You can answer at a high level, based on this lab)

- Individual Deliverables
  - Complete the peer-evaluation form appended in the end of this document

Grading Policy

Factor	Percentage
Part 2.3	30%
Part 2.4	10%
Part 2.5	20%
Part 2.6	20%
Questions	20%

ECE4810J System-on-Chip (SoC) Design

Fall 2021

Lab #4 Peer Evaluation Form

Each team member is required to provide a peer evaluation for the team effort of the lab. The score of the peer evaluation should be integers ranging between 0 to 5, inclusively, with 5 indicating the biggest contribution. A score should be given to each team member including yourself according the team member's contribution based on your observation. A brief description of specific contribution of each team member should also be provided. Note this form needs to be finished without discussing with your teammate, if all forms are found the same, then you have to justify how each member makes exactly the same contribution (through demonstration to the instructor).

Name	Level of contributions (0 – 5)	Description of contributions
(yourself)		
Team member 1		
Team member 2		
Team member 3		

Your lab grade is calculated based on the following:
Individual_Average = (sum of all team member's marks) / (size of the student team)
Group_Average = sum of Individual_Average of all team members / size of student team
Individual_Difference = Individual_Average / Group_Average – 1.0
Using the calculated Individual_Difference, we find a factor from the following lookup table.

Individual_Difference	Factor	Individual_Difference	Factor
>=0 and < +10%	1.0	> -10% and < 0	1.0
>= +10% and < +20%	1.1	> -20% and <= -10%	0.9
>= +20% and < +30%	1.2	> -30% and <= -20%	0.8
>= +30%	1.3	<= -30%	0.7

Your final lab grade is calculated by :
Final grade of lab = points for team effort * factor

Note that the final grade of the lab will not exceed the maximum points assigned to the project. If the peer-evaluation form is not submitted, then your individual_difference will be based on the available data (without yours) only, and your final score will be:
Final grade of lab = points for team effort * factor * 95%