23T2 COMP4601 Chapter 4 Lab/Week 6 Hand-in exercises 8/06/2023
(Notes and questions on PP4FPGAs – Chapter 4 exercises)

During Weeks 4 & 5, you are expected to complete Chapter 6 of the 2020.1 Vivado HLS Tutorial and to carry out selected exercises from Ch. 4 of the text. Your answers to indicated exercises should be submitted electronically by **5pm Monday 3 July**. This hand-in is worth 10% of your mark in the course. Your report should be between 5 and 10 pages long. Marks will be deducted for submissions longer than 15 pages.

Completing the exercises below should take 2 – 3 hours after you have completed Chs 2 – 6 of the Vivado HLS Tutorial. The files needed for this lab are contained in the `ch4.zip` zipfile. First you'll take a detailed look at the matrix-vector multiplication code of Figure 4.4, then you'll optimize the DFT code of Figure 4.15.

Matrix-vector multiplication exercises

Record your latency and utilization results for solutions 1 – 5 in two tables that allow these to be compared. Do not cut and paste the synthesis reports for individual solutions into your report.

solution1

> The matrix-vector multiplication code of Figure 4.4 is contained in `matrix_vector_base.c`. This code can be loaded into the project `matrix_vector_proj` using the Tcl script `matrix_vector_proj.tcl` and the Vivado HLS command prompt: `vivado_hls -f matrix_vector_proj.tcl`
>
> After you have created the project, run `vivado_hls -p matrix_vector_proj` from the Vivado HLS command prompt to open the synthesized baseline solution of the project in the Vivado HLS GUI.
>
> Describe the performance and resource utilization before adding any directives. Briefly explain the execution schedule obtained.

solution2

> Create a new solution from ***solution1***.
>
> Unroll the `dot_product_loop` completely by adding a directive to the directive script and run synthesis.
>
> Compare the performance and utilization with *solution1*.
> Explain why you obtain a loop iteration latency of 6.

solution3

Create a new solution from ***solution1****.*

Pipeline the `dot_product_loop` with the default II and run synthesis.

Compare the performance and utilization with *solution2.*
Explain what happened to the loop nest.
Explain why the iteration latency is 3 and the iteration interval is 1.

solution4

Create a new solution from ***solution1***.

Unroll the `data_loop` and run synthesis.

Is this worth doing? Why (not)?

solution5

Create a new solution from ***solution1****.*

Pipeline the `data_loop` and run synthesis.

Briefly explain the scheduling of the loop.
Taking both performance and utilization into account, rank the 5 solutions you have so far in your order of preference and explain your choice.

solution6

Create a new solution from ***solution1***.

Compare the performance and utilization of the manually unrolled code of Figure 4.6 with that of *solution2.* (The code is in the file named `matrix_vector_base_unroll_inner.c`. Copy this file to `matrix_vector_base.c` within your Windows directory and reload the source file in the GUI to confirm that you have copied the code correctly.)
Run synthesis.

Record your latency and utilization results for solutions 7 – 11 in two tables that allow these to be compared.

solution7

Create a new solution from ***solution5***.

In Windows, copy the file `matrix_vector_base_copy.c` to `matrix_vector_base.c` so as to revert back to the code used for *solution1-5*. Reload the source file in the GUI to confirm that you have restored the code correctly.

Add `array_partition` directives to the `M` and `V_In` arrays while pipelining the `data_loop`. The effect should be similar to the effect of, but not the same as, the listing of Figure 4.11. Add the directives **`%HLS ARRAY_PARTITION variable=M cyclic factor=2 dim=2`** and **`%HLS ARRAY_PARTITION variable=V_In cyclic factor=2 dim=1`** to the directives script and run synthesis.

Compare the resulting performance and utilization with that of *solution5*.
Briefly explain the execution schedule.

solution8

Create a new solution from ***solution7***.

Modify the `array_partition` directives to use **`block`** partitioning and run synthesis.

Explain the observed performance in the light of *solution7.*

solution9

Create a new solution from ***solution7***.

Modify the `array_partition` directives to implement **`complete`** partitioning and run synthesis.

Compare *solution5*, *solution7* and *solution9* in terms of performance and utilization. Explain your findings.

solution10

Create a new solution from ***solution9****.*

Modify the pipeline directive to **target an `II=2` and `run synthesis`**.

Compare *solution7*, *solution9* and *solution10.*

solution11

Create a new solution from *solution9* but set the target clock period to 5 ns. Run synthesis.

Explain the loop iteration latency you observe.

Do you think there is any further improvement in performance possible?

DFT exercises

Record your latency and utilization results for solutions 1 – 4 in two tables that allow these to be compared.

solution1

The DFT baseline code of Figure 4.15 is contained in `dft.cpp`[1]. This code can be loaded into the project `dft_proj` using the Tcl script `dft_proj.tcl` and the Vivado HLS command prompt: `vivado_hls -f dft_proj.tcl`

After creating the project, run `vivado_hls -p dft_proj` from the Vivado HLS command prompt to open the synthesized baseline solution of the project in the Vivado HLS GUI.

Describe the performance and resource utilization before adding any directives. Briefly explain the execution schedule.

solution2

Create a new solution from ***solution1***.

Pipeline the inner loop labelled `dft_label0` and run synthesis.

Explain the impact of pipelining `dft_label0` on the performance, execution schedule and utilization.
Which operations limit the iteration interval?

[1] There are some minor differences between the listing of Figure 4.15 and the contents of `dft.cpp`. `IN_TYPE` and `TEMP_TYPE` were set to `float`, and the expression for `w` was altered to allow the use of `ap_fixed` type data.

solution3

Create a new solution from ***solution2***.

Change `IN_TYPE` and `TEMP_TYPE` to be of **type `ap_fixed<16,4>`** and run synthesis.

Describe the performance and utilization of the resulting design in comparison to *solution2*.
Outline the most significant constraints on the performance of this solution.
What considerations have you ignored in changing the program data types?
What could you do to assess the impact of changing the program data types?

solution4

Create a new solution from ***solution3***.

Swap the inner and outer loops of the source code as explained on pages 97-99. Run synthesis.

What do you observe? Why?
Include a copy of your loop interchange code into your report.