Project 6: DMA Popcount (V)

E315

Version 2022.0

Autograder Due: Wednesday, October 25th, 2023

Demo Due: Friday, October 27th, 2023

Maximum Group Size: 2

Starter Code: https://github.com/Engr315/P6_DMA_Popcount.git

Goal

The goal of this project is to learn to use a Direct Memory Access (DMA) interface to accelerate data transfers between the CPU and the FPGA.

Popcount

The population count (or popcount) is the number of 1s (ones) in the binary representation of a non-negative integer. For example, 5 (which is 101 in binary) has a population count of 2, while 1 (001 in binary) and 8 (100 in binary) both have a population count of 1. [Link]

This is the same as previous projects.

Getting started

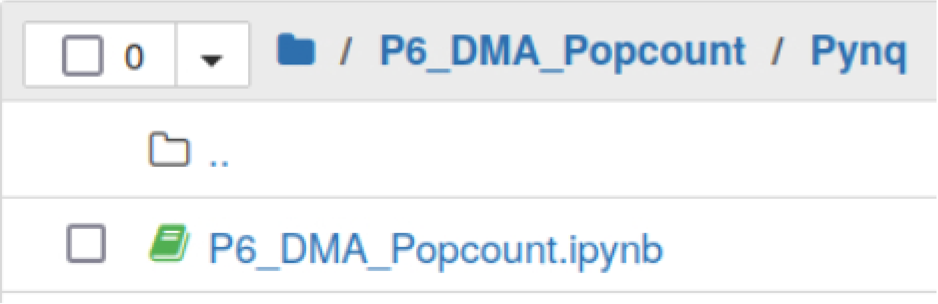
Similar to previous projects, use git to clone the project, and the e315helper script to get it setup.

```
git clone https://github.com/Engr315/P6_DMA_Popcount.git
cd P6_DMA_Popcount
python3 e315helper.py init --ip 192.168.2.99
```

(Your IP address may vary)

Popcount in Python

The Pynq folder contains the Python starter code for the Pynq board.



☐ 0 ▾ 📁 / P6_DMA_Popcount / Pynq

📁 ..

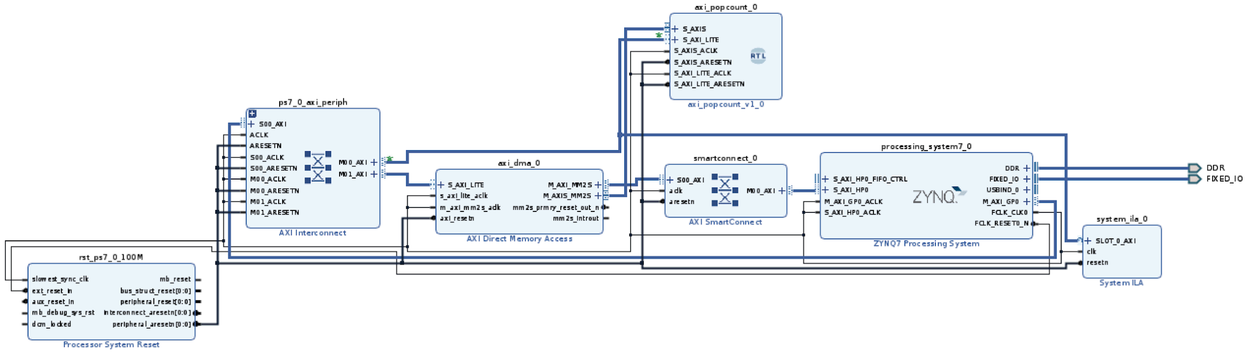
☐ P6_DMA_Popcount.ipynb

`DMA_Popcount.ipynb` contains the Python starter code, which is similar to previous projects. The only difference is the addition of a ‘dma’ version of the hardware popcount.

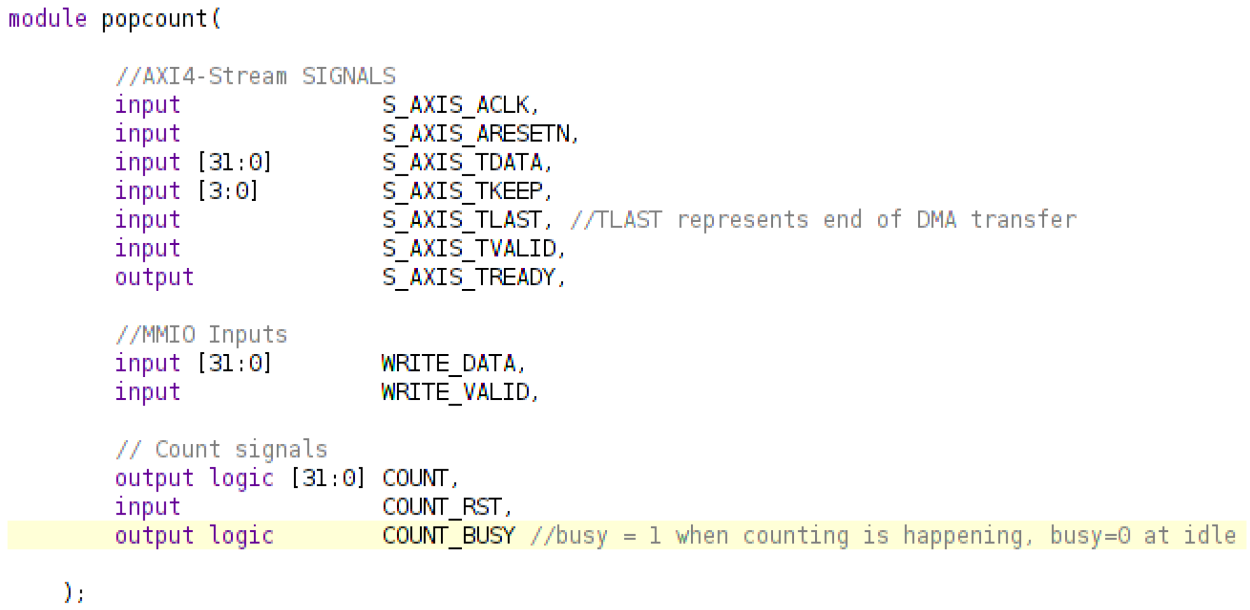
`DMAPopcount.py` is the Python file that you will need to modify. It is the Python interface into the FPGA’s DMA hardware’s block.

Popcount on Vivado

The e315helper script should build a Vivado project for you that looks like the image below.



You will **again** be modifying `popcount.sv`. This time you need to also support values coming over an AXI stream interface.



```
module popcount(

        //AXI4-Stream SIGNALS
        input                   S_AXIS_ACLK,
        input                   S_AXIS_ARESETN,
        input [31:0]            S_AXIS_TDATA,
        input [3:0]             S_AXIS_TKEEP,
        input                   S_AXIS_TLAST, //TLAST represents end of DMA transfer
        input                   S_AXIS_TVALID,
        output                  S_AXIS_TREADY,

        //MMIO Inputs
        input [31:0]            WRITE_DATA,
        input                   WRITE_VALID,

        // Count signals
        output logic [31:0] COUNT,
        input                   COUNT_RST,
        output logic            COUNT_BUSY //busy = 1 when counting is happening, busy=0 at idle

    );
```

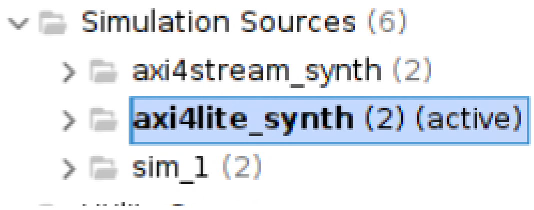
AXI4Stream Popcount Implementation Details

To correctly pass the supplied simulations (detailed below), your module should do the following:

- Correctly respond to MMIO popcount requests of previous projects.
- ***Also*** accept and compute popcount for a stream of values arriving over an incoming AXI stream.

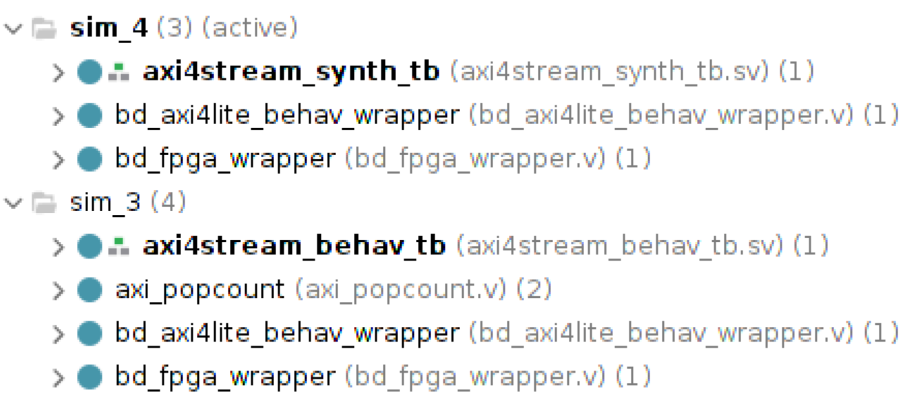
Simulation

There are also two pre-configured simulations to help with debugging.



```
Simulation Sources (6)
    axi4stream_synth (2)
    axi4lite_synth (2) (active)
    sim_1 (2)
```

Axi4lite_synth is an MMIO-only simulation. Axi5stream_synth is the DMA-enabled simulation. By default, axi4lite_synth is used. To use axi4stream_synth, right click on it and set “Mark Active”. The simulations should NOT require modifications. However, by default your popcount block does nothing, thus the simulations will initially fail.



```
v sim_4 (3) (active)
    > axi4stream_synth_tb (axi4stream_synth_tb.sv) (1)
    > bd_axi4lite_behav_wrapper (bd_axi4lite_behav_wrapper.v) (1)
    > bd_fpga_wrapper (bd_fpga_wrapper.v) (1)
v sim_3 (4)
    > axi4stream_behav_tb (axi4stream_behav_tb.sv) (1)
    > axi_popcount (axi_popcount.v) (2)
    > bd_axi4lite_behav_wrapper (bd_axi4lite_behav_wrapper.v) (1)
    > bd_fpga_wrapper (bd_fpga_wrapper.v) (1)
```

Pynq Python Documentation

https://pynq.readthedocs.io/en/v2.6.1/

In particular, we find this helpful:

MMIO

https://pynq.readthedocs.io/en/v2.6.1/pynq_libraries/mmio.html

DMA

https://pynq.readthedocs.io/en/v2.6.1/pynq_libraries/dma.html

Assignment Description

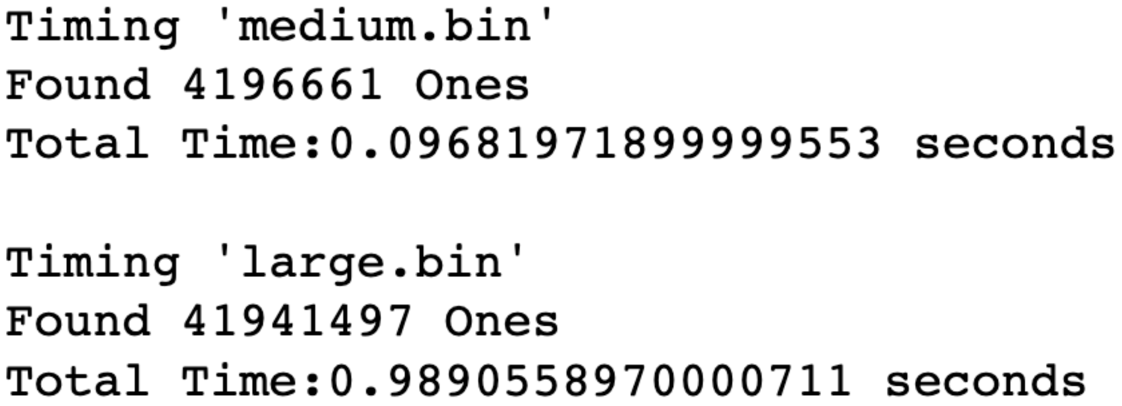
As noted above, the software-only implementation of popcount is quite slow. It requires almost 50 seconds to process the "large" 10M file.

Your task is to use DMA interface on the FPGA to accelerate this to UNDER 1 second (not 10).

For this to work, you will need to:

- Implement the popcount module in Vivado.
- Generate a bitstream and use the e315helper to copy it over to the Pynq board
- Update MyHardwarePopcount.py to load and utilize your hardware module
- Correctly count the number of 1’s in the ‘medium.bin’ file in under 1 second.
- Demonstrate the hardware-accelerated implementation to the TA

For reference, the instructor’s implementation runs the large input file in 0.98 seconds.



```
Timing 'medium.bin'
Found 4196661 Ones
Total Time:0.09681971899999553 seconds

Timing 'large.bin'
Found 41941497 Ones
Total Time:0.9890558970000711 seconds
```

Evaluation

Autograder (50%)

For this project, you will submit your `popcount.sv` implementation to the autograder. It will award you points based on both correctness and performance, with the bulk of the points being allocated for performance.

Demonstration (50%)

You will also need to demonstrate your working project to the TA in lab or office hours. If this is not possible, please make alternative arrangements with the TA.