0. EXERCISES

1) In Section 2, we discussed the VeeR EL2 configuration that includes a naïve branch predictor that always predicts branches as not taken. However, as we mentioned at the beginning of this lab, this processor includes a more efficient branch predictor called Gshare. Analyse the implementation of this Branch Predictor, both analysing the Verilog code available in module **el2_ifu_bp_ctl** and performing a simulation using RVfpga-Trace with the example used in the Section 2 enabling the Gshare BP.

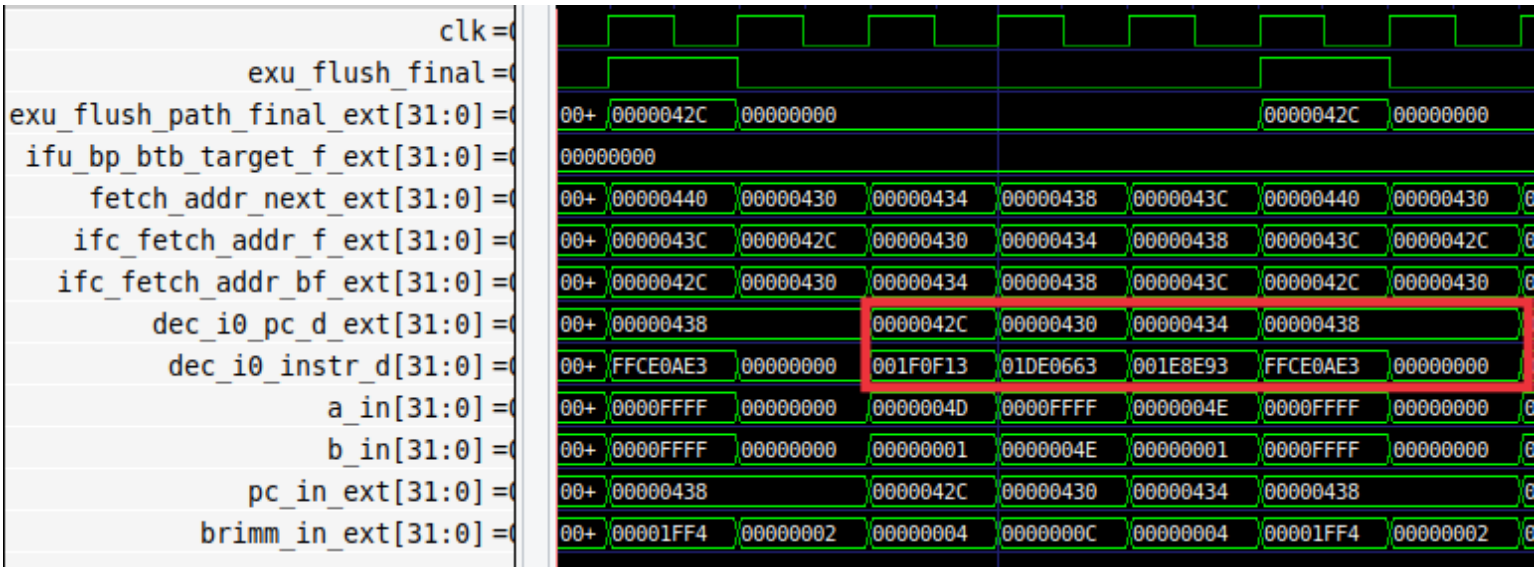
   NOTE: A classic paper published by Scott McFarling in 1993 is called "Combining Branch Predictors" (https://www.hpl.hp.com/techreports/Compaq-DEC/WRL-TN-36.pdf). It describes, in Section 7, the operation of the Gshare branch predictor. You can also search for other documents, such as https://people.engr.ncsu.edu/efg/521/f02/common/lectures/notes/lec16.pdf. We recommend reading them to understand how the Gshare BP works before beginning this section.

Solution not provided for this exercise.

2) In the examples shown in this lab we used the naïve Branch Predictor instead of the Gshare Branch Predictor. Test them enabling the Gshare Branch Predictor and analyse the differences in performance with respect to using the naïve one. Use the following RVfpga tools:
   a. RVfpga-Trace: Remember that you must include the control instruction.
   b. RVfpga-NexysA7: Remember that you must uncomment the configuration of the performance counters in file Test.c and remove the nop instructions in file Test_Assembly.S. If you don't have the physical board, simply skip this and test the program in simulation only.
   c. RVfpga-ViDBo: Remember that you must uncomment the configuration of the performance counters in file Test.c and remove the nop instructions in file Test_Assembly.S.

Naïve BP

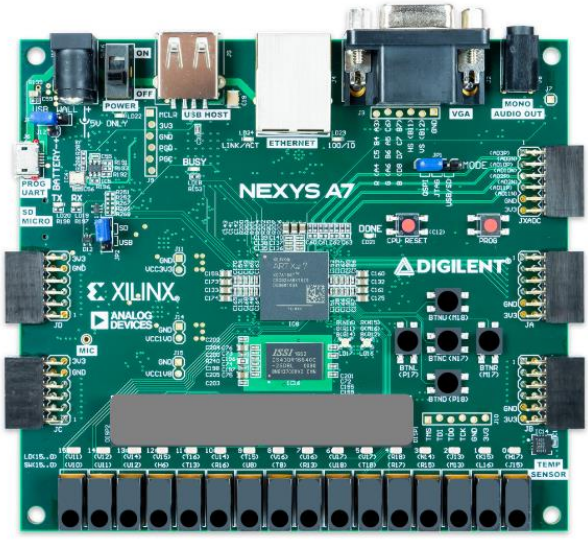
RVfpga-Trace:



RVfpga-ViDBo:

7 SEGMENT DISPLAYS: 0 0 0 0 0 0 0 0

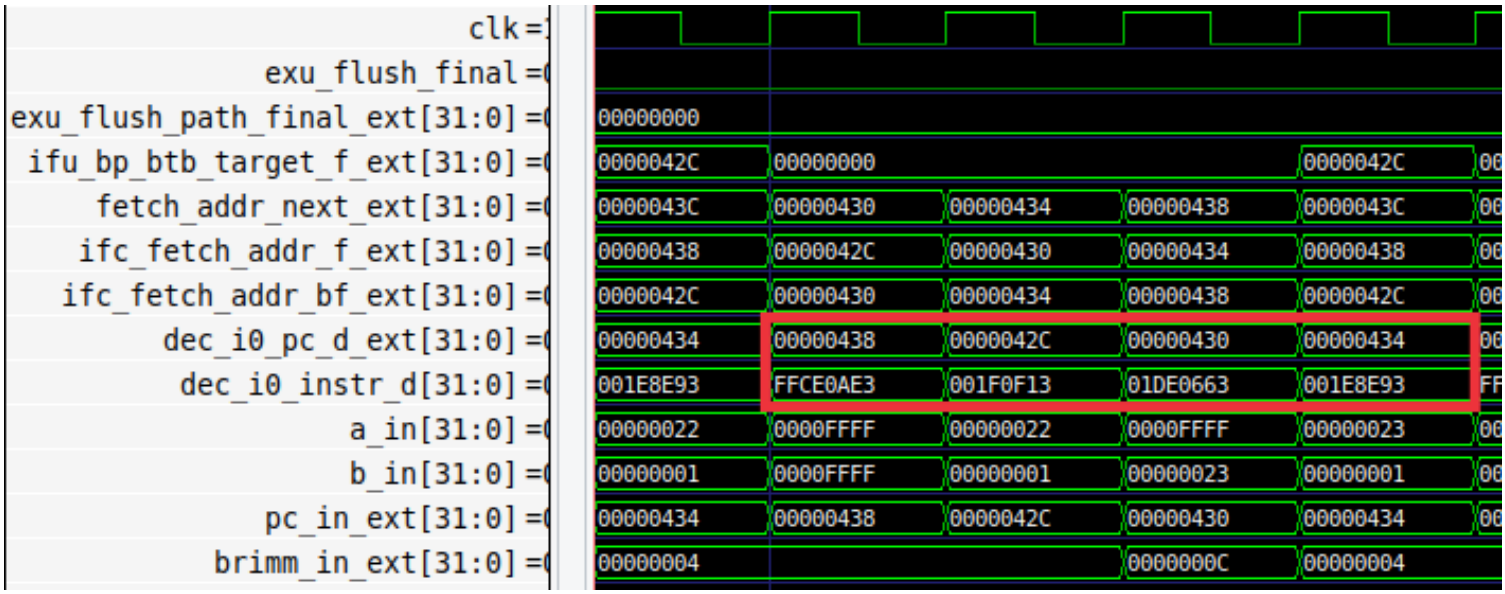
B value: **G value:** **R value:** **Color:**



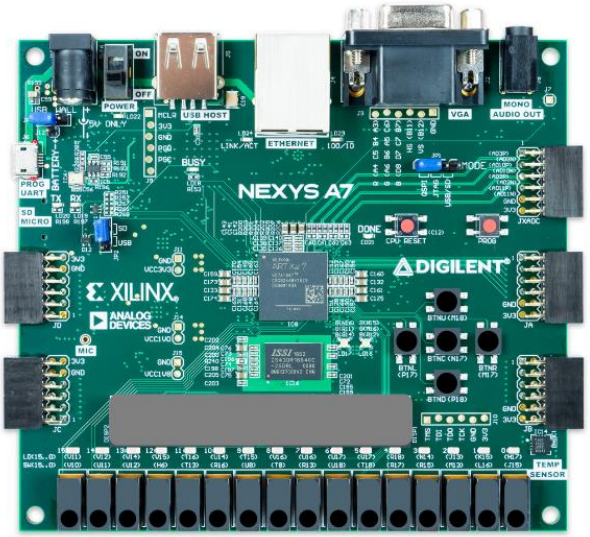
- There are 4 instructions and 65535 iterations. Ideally, this code would need around 262 thousand cycles. However, 1 cycle is lost per iteration due to the wrong prediction of the second branch. Thus, 5 cycles per iteration are spent.

Gshare BP

RVfpga-Trace:



RVfpga-ViDBo:



- There are 4 instructions and 65535 iterations. Now that the Gshare BP has been enabled, the code needs around 264 thousand cycles and IPC=1.

3) In the examples of previous labs we used the naïve Branch Predictor instead of the Gshare Branch Predictor. Test them enabling the Gshare Branch Predictor and analyse the differences in performance with respect to using the naïve one.

Solution not provided for this exercise.

4) Use a Bimodal Branch Predictor (you can implement it yourself or download one from the internet) and compare its performance to the Gshare BP.

Solution not provided for this exercise.