

## 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.

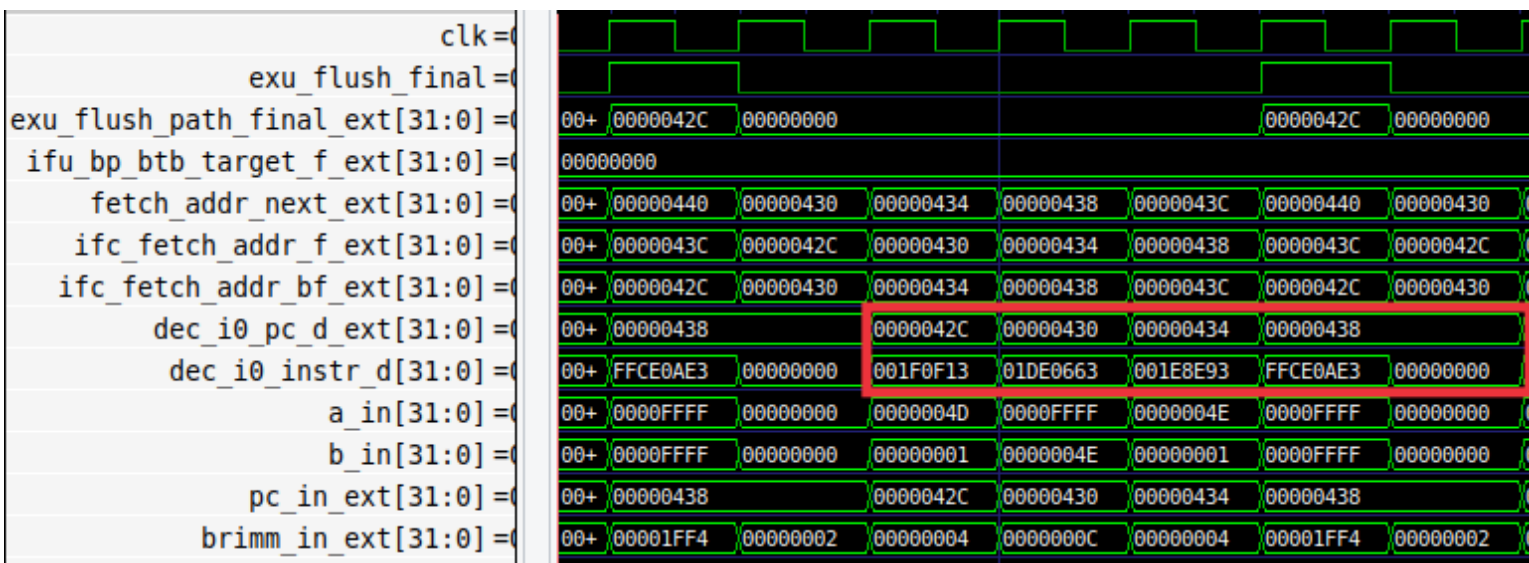
**NOTE:** A classic paper published by Scott McFarling in 1993 is called “Combining Branch Predictors” (<https://www.hpl.hp.com/techreports/Compaq-DEC/WRL-TN-36.pdf>). It describes, in Section 7, the operation of the Gshare branch predictor. You can also search for other documents, such as <https://people.engr.ncsu.edu/efg/521/f02/common/lectures/notes/lec16.pdf>. 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-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:



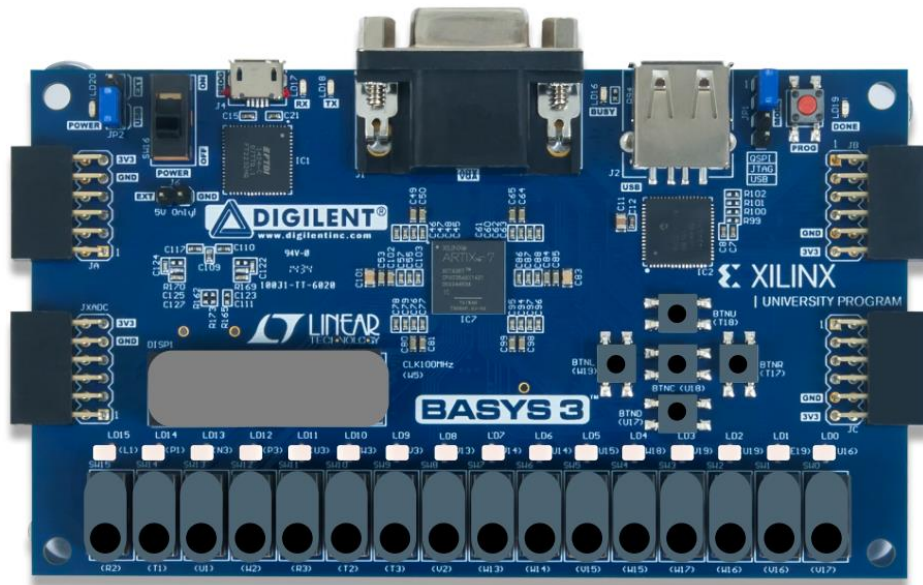
## RVfpga-ViDBo:

```

Cycles = 329684
Instructions = 262748
BrCom = 131154
BrMis = 65601
  
```

Disconnect Clear UART output

7 SEGMENT DISPLAYS: 0 0 0 0



- 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

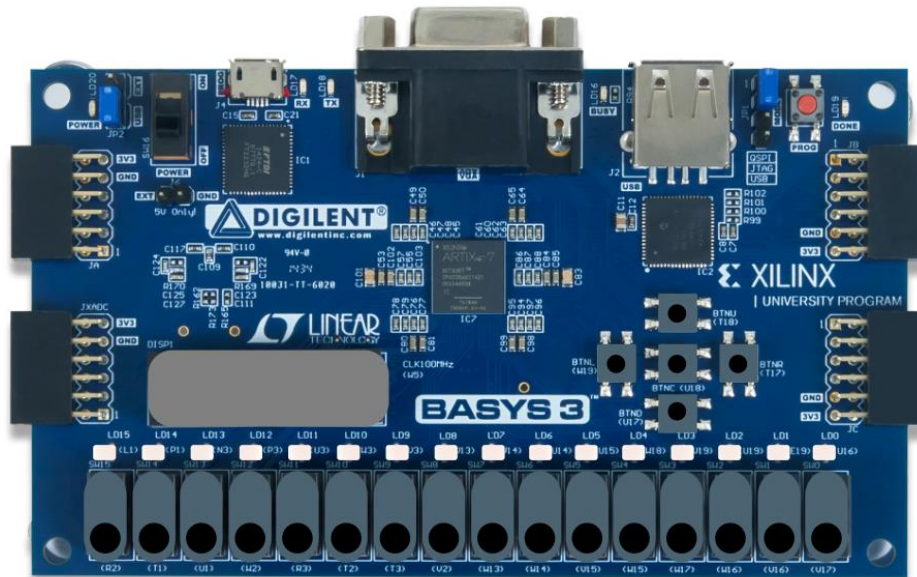
## RVfpga-Trace:

clk=					
exu_flush_final=0					
exu_flush_path_final_ext[31:0]=0	00000000				
ifu_bp_btb_target_f_ext[31:0]=0	0000042C	00000000	0000042C	00000000	00000000
fetch_addr_next_ext[31:0]=0	0000043C	00000430	00000434	00000438	0000043C
ifc_fetch_addr_f_ext[31:0]=0	00000438	0000042C	00000430	00000434	00000438
ifc_fetch_addr_bf_ext[31:0]=0	0000042C	00000430	00000434	00000438	0000042C
dec_i0_pc_d_ext[31:0]=0	00000434	00000438	0000042C	00000430	00000434
dec_i0_instr_d[31:0]=0	001E8E93	FFCE0AE3	001F0F13	01DE0663	001E8E93
a_in[31:0]=0	00000022	0000FFFF	00000022	0000FFFF	00000023
b_in[31:0]=0	00000001	0000FFFF	00000001	00000023	00000001
pc_in_ext[31:0]=0	00000434	00000438	0000042C	00000430	00000434
brimm_in_ext[31:0]=0	00000004		0000000C	00000004	

### RVfpga-ViDBo:

Cycles = 264156
Instructions = 262763
BrCom = 131157
BrMis = 19
<input type="button" value="Disconnect"/> <input type="button" value="Clear UART output"/>

7 SEGMENT DISPLAYS: 0 0 0 0



- 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.