**Design and FPGA Based Implementation of 1-Bit Dynamic Branch Predictor**

Sweety Prachi Chaudhary

*Electronics and Communication Engineering Electronics and Communication Engineering D.C.R.U.S.T, Murthal* Sonipat, India *D.C.R.U.S.T, Murthal* Sonipat, India

[sweetynain28@gmail.com](mailto:sweetynain28@gmail.com) [dahiya.prachi@gmail.com](mailto:dahiya.prachi@gmail.com)

**Abstract:** To improve the performance of the processors, the pipeline technique is used extensively to implement I.L.P (Instruction Level Parallelism). We measure the processor performance through the quantity of Instructions Level Parallelism represented through its design. We can counterstrike the parallelism with the aid of the execution of conditional branch instructions, which might additionally break the flow of the program execution. To overcome the branch problem, numerous ways have been suggested proactive to predict both the direction as well as the address of the executed instructions. In this paper, the FPGA ( Field Programmable Gate Arrays) based implementation of the 1-bit dynamic branch predictor is presented. The 1-bit dynamic branch predictor is developed especially for conditional branch instructions. The pipeline would not detach from the instruction queue when any conditional instructions occur as a 1-bit dynamic branch predictor implements it. This technique improves processor performance by implementing a deduction in the controlling hazard in the pipeline. Modelsim and Quartus tool is used for the design and implementation of the predictor.

**Keywords:** Pipeline, branch prediction,1-bit dynamic branch predictor.

# Introduction

With the onset of continuous growth of superscalar architecture, the industry experts began working to improve the performance of the processor. Many commercial processors are carried out on superscalar architectures, like Pentium(intel), Atlon(AMD), Ultrasparc, PowerPC7405, Alpha(DEC) and others.

The pipeline is an implementation technique to super scale [1] the processor performance and allows better utilization of hardware. The rationale behind using pipelines is to improve processor performance. Without pipelining in a processor, each instruction must wait till the previous instruction has been executed successfully. This should happen before the next instruction begins in a correct sequential manner. When the pipelines are more and more profound, the branch misprediction penalty is increasing due to conditional branches [2].

In the pipeline, the next instruction would be prevented within the instruction stream from executing at some point in its exact cycle. This situation is called the pipeline hazard [3]. Hazards are classify into three types:-

1. Structural hazard [4]: This hazard occurs when any instruction requires some additional units for the execution of the instruction. But in this case, the other units cannot support the execution of the instructions due to the pipelining.
2. Data hazard [5]: This hazard occurs when an instruction depends on the result of previous instructions that are unavailable or inaccurate.
3. Control hazard [5]: This hazard occurs in the pipeline stage when the branches enter the pipeline stage and change the execution order of the program counter of the other instructions.

The superscalar processor has to execute instructions to conquer control hazards. The negative results of control hazards on the effective processor performance decrease as the depth of the pipelines increases. This fact is certainly

one of the problems in present-day processor design. To improve accuracy rate of branch prediction, the best methodology would be to speculate at the most in all likelihood execution paths.

Branch Predictor

Branch outcome Predicted direction

and other information

Program counter Predicted target address

Fig 1: Schematic diagram for the branch prediction[1]

A branch is a type of control logic that allows code to shift from one block to another. The decoding and interpretation of a branch instruction by the processor would be time-consuming. Branch prediction enables the processor to predict the outcome of the branch, and that outcome will be based on some prediction strategy, which is set by the instruction set architecture. The main concept of the branch prediction is to reduce the total number of stalls owing to the branch instructions. When any branch comes in contact with any program, it must be executed before the program knows what instructions to run next. The branch prediction tells the processor if the outcome of the branch has to be taken or not and how to continue executing instructions based on that assumption. In the case when the prediction is correct, computation time is utilized effectively, which enhances the processor performance. In the case where the prediction is incorrect, the performance of the processor does not improve by the branch prediction. However, depending on the prediction strategy, the performance of the processor based on the prediction of the branches.

Branch prediction can predicts the direction using static or dynamic prediction. Static branch prediction means that a given branch will always be predicted as taken or not taken without the possibility to change the condition throughout the execution of the program. Whereas, the dynamic branch prediction can change the expected state throughout the implementation of the program.

The accuracy of the branch prediction depends on the prediction of the branches. It has an impact when the predicted outcome is correct with the actual result of that branch. Overall, branch prediction provides an increase in processor performance for conditional branches.

**Contributions:** This paper outlines the concept of branch prediction in the pipeline. Further, an FPGA based 1-bit dynamic branch predictor, which is designed and implemented using a demultiplexer, multiplexer, 1-bit counter, and comparator to enhance the processor performance. The implementation is done using HDL files ,and the simulation and waveform of each block are carried out differently by leveraging Modelsim and Quartus tools.

The remainder of the paper is structured as follows. The background & related work are given in section 2. The architectural exploration of the proposed predictor has been discussed in section 3. The proposed algorithm of the 1-bit dynamic branch predictor is explained in section 4. Section 5 discusses the results extracted from each simulator and is discussed in section 5. Finally, section 6 summarizes the main conclusion and future work of the paper.

## Background and related work

This section will explore various existing techniques for branch prediction. Many researchers have tried to enhance the performance of processors by branch prediction using static as well as dynamic strategies. An original and informative paper on branch prediction schemes was published by Lee and Smith.

Jin et. al. [6] a proposed ahead branch prediction architecture, which predicts the direction by using two predictors. One predictor is located at the front part of the pipeline, which contains four entries of branch target. whereas, the second predictor is located at the back-end of the pipeline, and its prediction is based on the last branch register and previous target register information. The proposed method improves average performance by 2.92% than the branch target buffer based predictor.

Berestizshevsky and Weiss [7] proposed a decoupled branch predictor for dynamic branch prediction. This predictor avoids unnecessary access to the branch history table. This proposed predictor enhances the average speed of up to 8.95%.

Culpepper and Gondree [8] analyze the branch prediction using SVM based learning technique. Various characteristics of the branches and loops are fed into SVM to correctly predict the direction of branches,as this approach is very simple and provides better accuracy results than the perceptron technique. The main advantage of this approach is using a long history length to reduce the misprediction rate.

Ismail [9] described various branch predictors using SPECint95 benchmarks. The performance of each branch predictor is compared in terms of the misprediction rate and branch accuracy rate. Among all predictors, the hybrid is the most promising predictor and provides an improvement over gshare dynamic branch predictor.

Parasanna et al. [10] implemented a bimodal base predictor along with a tagged table indexed with branch history. This predictor improves the accuracy of conditional and indirect branch instructions.

Chieh et. al. [11] proposed the bi-mode branch predictor, which uses two pattern history tables to reduce the effects of aliasing. This predictor uses two-bit counters, and the most significant bit indicates which of the two pattern history tables is to be used. This predictor can help remove aliasing while keeping the aliasing collectively for the two-bit counter tables.

Sweety and Chaudhary [12] implements the always taken and always not-taken static branch prediction schemes. The results in paper shows that the always taken gives better accurcay as compare to another technique. These techniques improve the control hazards problems in the pipeline.

Dual-path instruction processing is proposed by yang et al. [13] using the Branch Prediction Reversal Unit(B.P.R.U). This architecture targets to reduce the pipeline-fill penalty after a misprediction. An 8% improvement is noted over a single path with Gshare predictors.

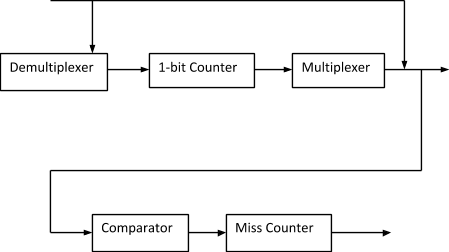
Table 1: Comparison of several branch prediction schemes

|  |  |  |
| --- | --- | --- |
| Predictor | Description of features to exploit branches | Reference |
| Local branch predictor | During prediction, the process will involve delving deeper into the history of the branch to look at what was  the last result and will predict the outcome over this. | [14–16] |
| Global branch predictor | The prediction will be made using the outcomes of the previous branches and not the history of the particular branch. | [17–19] |
| Hybrid branch  predictor | Use combine predictor for local as well as global branch  predictor. | [20–22] |
| Neural branch  predictor | A weighted vector is chosen, which depends on the path  that leads to a branch instead of only the branch address. | [23–25] |

## Architectural exploration of the 1-bit dynamic branch predictor

In this section, FPGA based implementation of the 1-bit dynamic branch predictor is described. An FPGA device consists of a logic cell, and each cell interconnects to carry out a specific function. The desired hardware functionalities are typically described in the HDL code that is synthesized and implemented by means of the FPGA device. Due to the programmability of the FPGA devices, custom-designed hardware can be incorporated into the embedded system as well.

Address of the branch



# Prediction

# Miss-Prediction

## Fig 2: Schematic view of the proposed branch prediction

A schematic view of the branch predictor is depicted in fig 2. The proposed predictor contains the combination of different generic blocs, and each block performs its operation on the input instructions. The various generic blocks are demultiplexer, 1-bit counter, multiplexer, comparator, and miss counter. Starting from the demultiplexer is used to change the value of the counter. A counter is used to store the local history and change its value according to the previously executed value. A multiplexer is used to select the prediction and the result of the multiplexer is to be considered as a prediction. A comparator is used to compare the value using the XOR gate between the outcome and the prediction of the branch. In the end, the miss counter determines the number of mispredictions.

## The proposed algorithm of the 1-bit dynamic branch predictor

The proposed algorithm of the 1-bit dynamic branch predictor is implemented in VHDL using only basic units such as counters, multiplexer, comparator, and demultiplexer. The recommended steps are followed as below:

1. *Convert the assembly language into MIPS code*
2. *Consider MIPS code as the test program for the blocks*
3. *Generate all branch decisions using an already executed program*
4. *Use above generated sequence as an input to the HDL architecture*
5. *Simulation result and waveform of each block of HDL is to be implemented*
6. *Find out the number of misprediction and the number of addresses hit.*

The test program is made up of two loops, one inside the other. The outer is defined by j, whereas the inner loop is defined by i.

*int c,*

*int main(){ int i,j;*

*for (i=0; i<1000;i++) # (Outer loop) for (j=0;j<4;j++) # (Inner loop) c++;*

*}*

*return c;*

*}*

The inner loop is made with a counter from 0 to 3, so there will be four iterations of the inner loop every time. This loop will have the following behavior: T, T, T, NT. Indeed the last iteration will be the end of the loop and so, the branch will not be taken. Since the system has a 1-bit counter, the prediction will be the outcome of the previous iteration, hence NT, T, T, T. Finally, there is two mispredictions for the inner loop.

The outer loop is made from 0 to 999. First of all, the behavior of the loop will be T, T, T, T, NT. Now the

counter is initialized on Not Taken, so there will be one misprediction at the beginning and another at the end when the loop is over.

## FPGA based implementation of the 1-bit dynamic branch predictor

This branch predictor stores the recent prediction of each conditional branch and assumes that future behavior will continue the same way and make predictions. In this predictor, the 1-bit value indicates whether the branch is predicted to be taken or not taken.

The prediction is based on the lower bit, and this bit of the program counter indexes the value in 1-bit and gets the prediction. Here, prediction means if the recent branch was taken or not taken. The processor fetches the next instruction from the target address or the sequential address. The 1-bit value to indicate whether the branch is predicted to be taken or not taken. It stores the value in the 1-bit form either in 0 - Prediction was not-taken (NT) (Last time) or 1- Prediction was taken (T) (Last time).

1-bit branch predictor is made up of counters to store the local history, one multiplexer to select the prediction, a comparator to check if there is a misprediction or not, a 1-bit counter to determine the number of misprediction and a demultiplexer to change the value of the counter.

Additionally, this dynamic branch predictor uses the branch history table to store the prediction history of the conditional branches. The lower bit of the branch index the value in the branch history table with 0 and 1. Based on the prediction processor, fetch the next instructions from the target address or continue the sequential instructions.

## Demultiplexer

The address of the instructions is assigned as an input.

Depending on the value of the input condition, it gives the output as conditional and unconditional branch instructions.

The clock is used to change the input value of the address of the branch.

Table 2: Outlines the results of demultiplexer

|  |  |  |  |
| --- | --- | --- | --- |
| Clock | Address of the branch | The outcome of the branch (Unconditional branch) | Conditional of the branch |
| 0ns | 3h0 | 0 | 1 |
| 10ns | 3h1 | 1 | 0 |
| 20ns | 3h2 | 0 | 1 |
| 30ns | 3h3 | 0 | 1 |
| 40ns | 3h4 | 1 | 0 |
| 50ns | 3h5 | 1 | 0 |

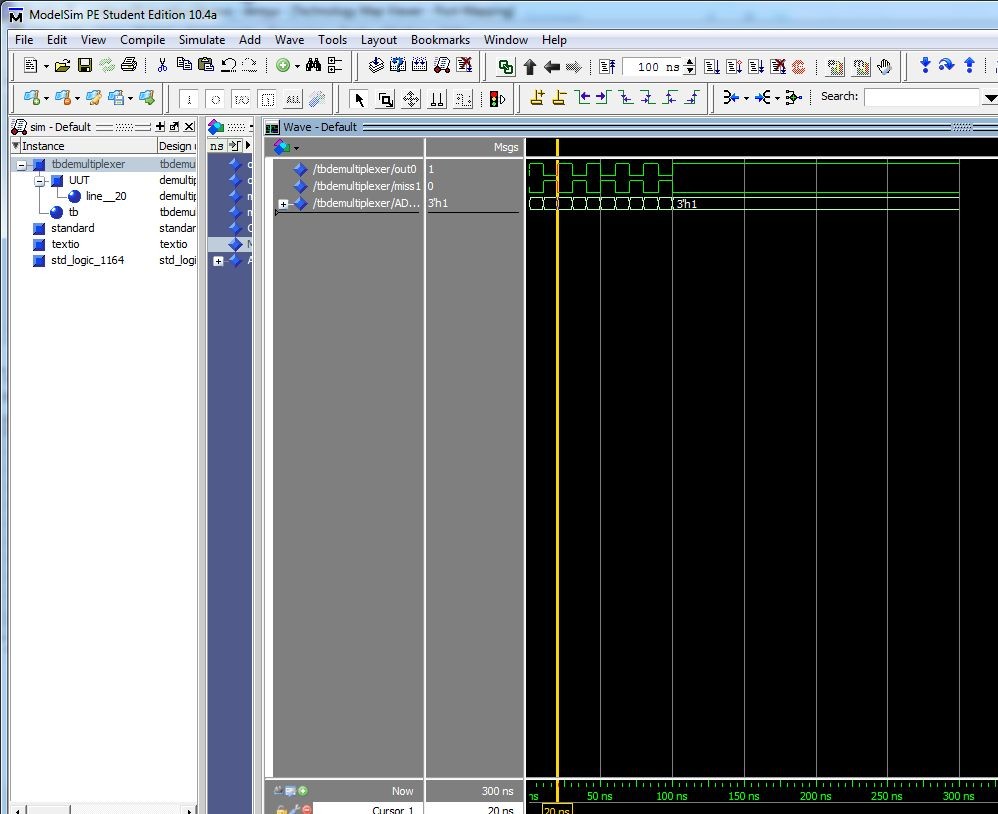


Fig 3: Simulation results of demultiplexer

## Multiplexer

The multiplexer is used to select the prediction.

Depending on the value of the input condition and address of the conditional branch, it gives the output as prediction 0 and prediction 1.

Input A – Unconditional branch, Input B – Conditional branch, Input - address of the branch The clock is used to change the input value of the address of the branch.

Output - determine the prediction of the instructions.

Table 3: Outcome results of multiplexer

|  |  |
| --- | --- |
| Clock | Prediction |
| 0ns | 0 |
| 10ns | 0 |
| 20ns | 1 |
| 30ns | 0 |
| 40ns | 1 |
| 50ns | 1 |

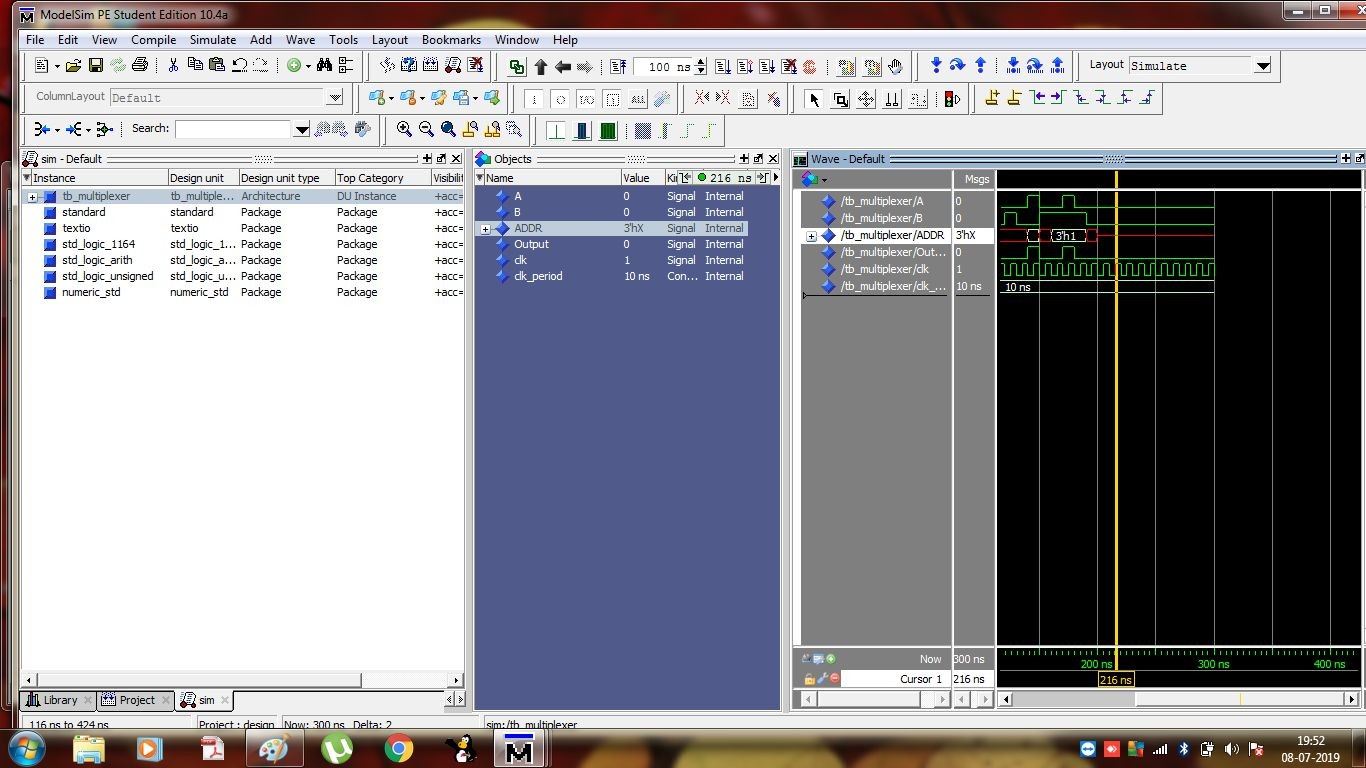


Fig 4: Simulation results of multiplexer

# Comparator

The comparator will compare the outcome and the prediction value of the multiplexer. Depending on the output of the comparator, the result is considered as a miss value.

Input – Outcome of the demultiplexer Input – Address of the branch

Input – Prediction of the multiplexer Output – Address of the updated branch

Output (miss) - XOR of outcome and prediction value

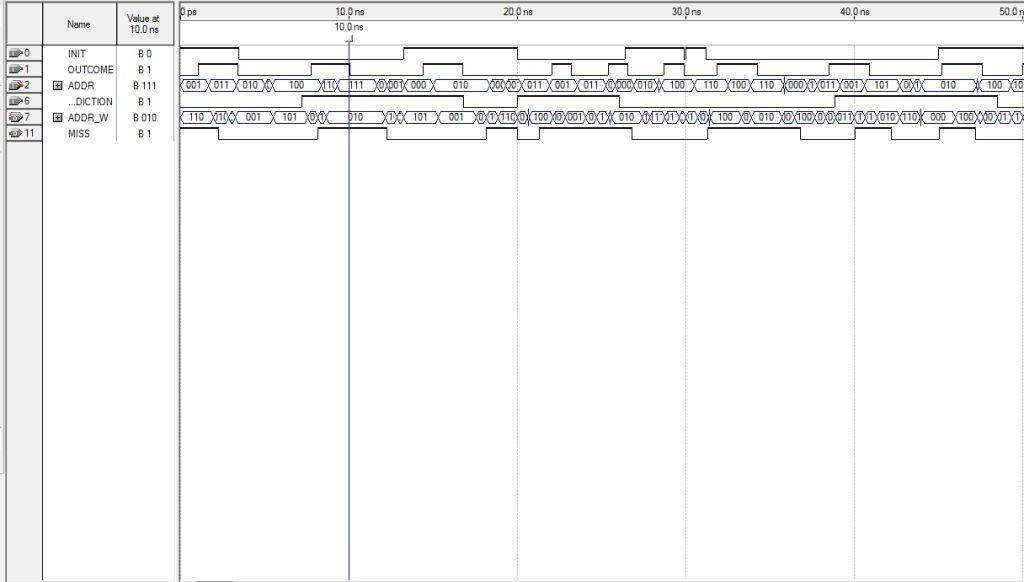


Fig 5: Simulation results of comparator

## 1-bit Counter

Counter stores the local history and determine the number of mispredictions. The output of the comparator is considered as an input for the 1-bit counter.

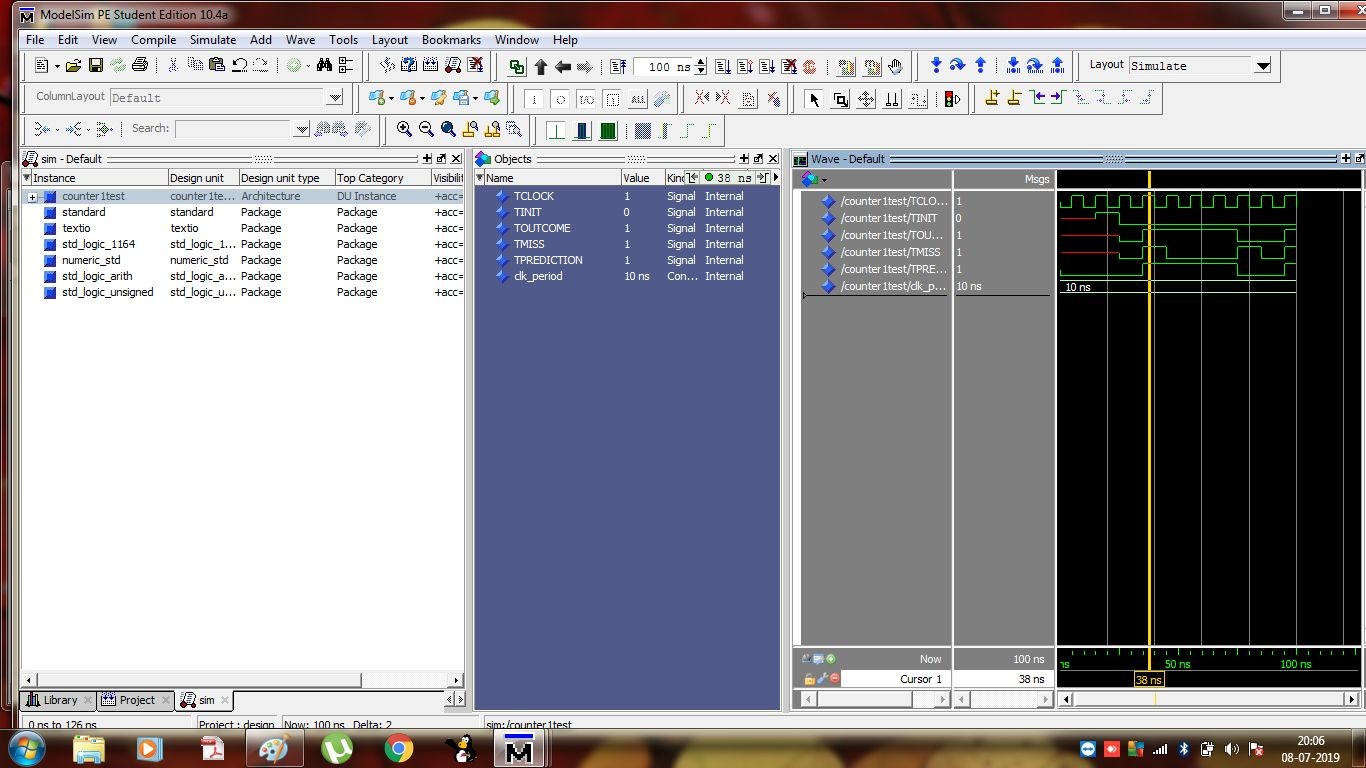
The outcome of the 1-bit counter is that it mis-predicts the first and last branch of the instructions.

Fig 6: Simulation results of the 1-bit counter

1. **Conclusion and future work**

This work presents the implementation of branch predictors for general-purpose processors. The 1-bit dynamic branch predictor is implemented in Verilog HDL. This predictor is made up of different blocks such as a multiplexer, demultiplexer, comparator, and 1-bit counter. The simulation results of all the blocks have been carried out using a Modelsim simulator. The testing of the branch predictor is done by using different inputs of VHDL files. The outputs were examined by the various numbers of inputs.

Hence, the proposed FPGA based dynamic branch architecture leads to becoming more competitive then static branch predictors. With 1-bit prediction, if we mispredict once about the branch, we change our mind instantly about the next prediction. This might not be a good thing.

In the future, this work can be further extended with a 2-bit dynamic branch predictor as this prediction will predict the prediction accuracy with 2- bit.

**Reference**

1. Sweety, P.chaudhary: Branch Prediction Techniques Used in Pipeline Processors : A Review. Int. J. Pure Appl. Math. 119, 2843–2851 (2018).

2. Starke, R.A., Carminati, A., de Oliveira, R.S.: Evaluation of a low overhead predication system for a deterministic VLIW architecture targeting real-time applications. Microprocess. Microsyst. 49, 1–8 (2017). https://doi.org/10.1016/j.micpro.2016.11.017.

3. Habibizad Navin, A., Lahouti, E., Lotfi Anhar, M., Mirnia, M.K.: A new method to prevent control hazard in pipeline processor by using an auxiliary processing unit. 2010 2nd Int. Conf. Adv. Comput. Control. 596–599 (2010). https://doi.org/10.1109/ICACC.2010.5487130.

4. Changela, A.: Hazard Detection and Data Forwarding Scheme for 5-stage Pipeline Structure of RISC Processor. Int. J. Innov. Res. Comput. Commun. Eng. 4, 11693–1169 (2016). https://doi.org/10.15680/IJIRCCE.2016.

5. Pandey, A.: Study of data hazard and control hazard resolution techniques in a simulated five stage pipelined RISC processor. IEEE. 2, 1–4 (2016).

6. Jin, W., Shi, F., Song, Q., Zhang, Y.: A novel architecture for ahead branch prediction. Front. Comput. Sci. 7, 914–923 (2013). https://doi.org/10.1007/s11704-013-2260-x.

7. Berestizshevsky, K., Weiss, S.: Decoupled Branch Predictor for Embedded DSP. IEEE Embed. Syst. Lett. 10, 57–60 (2018). https://doi.org/10.1109/LES.2017.2755062.

8. Architecture, E.C., Farrens, P.M., Culpepper, B.J., Gondree, M.: SVMs for Improved Branch Prediction Machine Learning in Branch Prediction. (2014).

9. Ismail, N.A.: Evaluation of dynamic branch predictors for modern ILP processors. Microprocess. Microsyst. 26, 215–231 (2002). https://doi.org/10.1016/S0141-9331(02)00023-6.

10. Parasanna, A.N.S.: A Study on Improving Branch Prediction Accuracy in the context of Conditional Branches. 4, 654–662 (2017).

11. Lee, C.-C.L.C.-C., Chen, I.-C.K., Mudge, T.N.: The bi-mode branch predictor. In: Proceedings of 30th Annual International Symposium on Microarchitecture. pp. 4–13 (1997). https://doi.org/10.1109/MICRO.1997.645792.

12. Sweety; P. chaudhary: Implemented Static Branch Prediction Schemes for the Parallelism Processors. In: International Conference on Machine Learning, Big Data, Cloud and Parallel Computing(IEEE Xplore). pp. 79–83 (2019).

13. Dai, H., Yan, C., Gong, B., Yang, J.: Explore prediction for instruction level redundant execution in fault tolerant microprocessors. J. Syst. Archit. 70, 70–81 (2016). https://doi.org/10.1016/j.sysarc.2016.04.004.

14. Ho, C.Y., Fong, A.S.S.: Combining Local and Global History Hashing in Perceptron Branch Prediction. 6th IEEE/ACIS Int. Conf. Comput. Inf. Sci. (ICIS 2007). 54–59 (2007). https://doi.org/10.1109/ICIS.2007.81.

15. Hida, I., Ikebe, M., Asai, T., Motomura, M.: A 2-clock-cycle Naïve Bayes classifier for dynamic branch prediction in pipelined RISC microprocessors. 2016 IEEE Asia Pacific Conf. Circuits Syst. APCCAS 2016. 297–300 (2017). https://doi.org/10.1109/APCCAS.2016.7803958.

16. Mühlbauer, F., Schröder, L., Schölzel, M.: On Hardware-based Fault-Handling in Dynamically Scheduled Processors. In: Design and Diagnostics of Electronic Circuits \& Systems (DDECS), 2017 IEEE 20th International Symposium on. pp. 201–206 (2017).

17. Parihar, R.: Branch Prediction Techniques and Optimizations. Univ. Rochester, NY, USA. (2015).

18. Skadron, K., Martonosi, M., Clark, D.W.: A Taxonomy of Branch Mispredictions , and Alloyed Prediction as a Robust Solution to Wrong-History Mispredictions. Proc. 2000 Int. Conf. Parallel Archit. Compil. Tech. (Cat. No.PR00622). 1–8 (2000).

19. Parihar, R.: Branch Prediction Techniques and Optimizations. Univ. Rochester, NY, USA.

20. Su, X., Wu, H., Yang, Q.: An Efficient WCET-Aware Hybrid Global Branch Prediction Approach. Proc. - 2016 IEEE 22nd Int. Conf. Embed. Real-Time Comput. Syst. Appl. RTCSA 2016. 195–201 (2016). https://doi.org/10.1109/RTCSA.2016.46.

21. Otiv, S., Garikipati, K., Patnaik, M., Kamakoti, V.: H-Pattern : A Hybrid Pattern Based Dynamic Branch Predictor with Performance Based Adaptation. 2–5.

22. Dirk Grunwald , Donald Lindsay, B.Z.: static methods in hybrid branch prediction. (1997).

23. Mao, Y., Zhou, H., Gui, X.: Exploring Deep Neural Networks for Branch Prediction. 1–7.

24. Jiménez, D.A.: An optimized scaled neural branch predictor. Proc. - IEEE Int. Conf. Comput. Des. VLSI Comput. Process. 113–118 (2011). https://doi.org/10.1109/ICCD.2011.6081385.

25. Steven, G., Anguera, R., Egan, C., Steven, F., L., V.: Dynamic Branch Prediction using Neural Networks. Dsd’01. 178–185 (2001).