Performance Analysis of Branch Prediction Unit for Pipelined Processors

Nikhil Panwar
Student M.Tech VLSI Design
ACS Division, Centre for Development of Advanced Computing (C-DAC), Mohali, 160071, India

Manjit Kaur
Engineer
ACS Division, Centre for Development of Advanced Computing (C-DAC), Mohali, 160071, India

Gurmohan Singh
Senior Engineer
DEC Division, Centre for Development of Advanced Computing (C-DAC), Mohali, 160071, India

ABSTRACT
The branch predictor plays a crucial role in the achievement of effective performance in microprocessors with pipelined architectures. This paper analyzes performance of branch prediction unit for pipelined processors. A memory of 512 bytes is designed for storing instructions. A 32 byte memory is designed for branch target buffer (BTB). This memory is utilized for storing history of the branch instructions. A Finite State Machine (FSM) is designed for branch predictor unit. It consists of four states: strongly taken, weakly taken, weakly not taken and strongly not taken. Prediction is done based on the status of FSM. If the state of FSM is weakly taken or strongly taken, then predictor guesses it as a taken condition else it is assumed to be not taken condition. When the execution of branch instruction is done for the first time the BTB stores the address of current instruction and also the address where it jumps. After this the current status of the FSM is updated accordingly. The program is executed using a branch predictor unit and also without a branch predictor unit. The latency of both the processors with a branch prediction unit and without is branch prediction unit is computed and compared. The simulation results validates that with branch prediction unit latency is decreased.

Keywords
BTB, FSM, ILP, FPGA, Latency, Processor.

1. INTRODUCTION
The performance of microprocessor architectures has doubled in every two to three years. The techniques used for high performance computing are Pipelining and Predictor. Pipelining is highly preferred in high performance embedded processors as it can increase instruction level parallelism. The processor can be broken into different stages while storing each intermediate stage by using pipelining. Pipelining can be applied for the execution of a number of instructions at a particular time. As a result the throughput, which is the number of instructions per second of the processor, is increased [2]. The pipelined instructions need to be examined carefully to understand the effects created by changes in control flow. For an instance four pipelined structures may be required in a pipelined structure namely, instruction fetch (IF), Instruction decode (ID), Execute (EX), and Write back (WB). Each instruction undergoes many stages of execution till the result of fed instructions is known in the process of pipelining. In each preceding stage of pipelining many instructions are being executed simultaneously [3]. When instructions are being fetched a delay occurs before the results of execution, this delay is caused by the conditional branches due to unavailability of the next fetch address and this delay creates ambiguity in case of branch instructions. The instructions are executed sequentially. Due to branch instructions the flow of instruction changes, therefore the fetching unit in the processor should have prior knowledge of the fact that which part of the instruction should be fetched first in order to utilize the pipelining stages contained in the branch instructions. In case of conditional branches two instructions can be followed. If the conditional branch is processed, the fetching of the next instruction is done from the address of the next consecutive instruction which is known as fall through instruction or the instruction is fetched from the target address which is known as target instruction. The branch problem arises since the conditional branch is required to wait for the condition to get resolved and the address of the next instruction is calculated before the next instruction is being fetched. This results in a delay in the processor. Due to these delays the processor performance is degraded. The processing is required to be stopped and the processor needs to wait till the direction of the branch is not discovered. This introduces stalls in the pipeline. The number of stalls is determined by calculating the number of stages in between fetching stage and that stage in which the branch was resolved. The performance problem can be removed by adopting a technique called Branch prediction [4].

Branch predictor helps to predict the path of a branch instruction before actually knowing its behaviour. Flow in the instruction pipeline will be improved using branch predictor. In modern microprocessors with pipelined architectures branch predictors play a crucial role in achieving high performance effectively [4]. Conditional jump instruction is used to implement two-way branching. In case when the conditional jump is considered to be as *not taken* the execution continues along the first branch of the code which comes immediately after the conditional jump. In case when the conditional jump is considered to be *taken* the execution jumps to the location in the memory of the program where the code for the second branch is stored. In pipelined structures clock cycles are shorter as the work required by each stage is not more. The processors are designed with multiple instruction pipelines which allow issuing of multiple instructions in each cycle. The processor should be supplied large number of instructions in order to use the pipelined stages efficiently. The decision to the fact that conditional jump is *taken* or *not taken* cannot be made until calculation has been made on the condition and also until conditional jump has passed execution stage in the instruction pipeline [5]. When branch prediction unit is not present the processor is required to wait for conditional jump instruction to pass through the execution stage before the next instruction is entered into the fetch stage in the pipeline. The branch predictor guesses whether the chances for conditional jump are more for being *taken* or *not taken* and thereby prevents

wastage of cycles. The branch is fetched and speculatively executed which has been guessed to be the most likely to happen. The speculatively executed or partially executed instructions are discarded and the pipeline starts over with the correct branch, incurring a delay if it is later detected that the guess was wrong. The boxes are used to represent the instructions which are independent of each other [9]. The number of stages included in the pipeline structure from the fetch stage to the execution stage is equivalent to the time which is wasted when a branch misprediction occurs. A good branch predictor is required when the pipeline gets longer. When a conditional jump is encountered firstly no information is available for a prediction to take place. A record is kept by the branch predictor whether the branch is taken or not taken.

2. LITERATURE SURVEY

A lot of research work has been carried out to enhance the performance of processor. This section provides an idea about various developments in the past on branch predictors.

J. V. Kumar *et al.* in 2014 presented a low power pipelined 64-bit RISC processor containing a Floating Point Unit based on FPGA [7]. The development of this processor was carried out especially for carrying arithmetic operations on both fixed and floating point numbers, for branch and logical functions. No plush occurs for pipelining in case of occurrence of branch instructions as its implementation was alone by making use of dynamic branch prediction. As a result the flowing instruction was increased and high effective performance was encountered. By using RTL coding the dynamic power could be reduced since it uses clock gating technique. In this paper Double Precision floating point arithmetic operations such as addition, division, multiplication and addition were also implemented.

Priya P. Ravale *et al.* in 2010 designed a branch prediction unit of a microprocessor based on superscalar architecture [8]. In the proposed design rigorous research was carried out using simple scalar tool through simulation of superscalar architectures. The result was focused on areas namely, data dependence, memory latency and control dependency. Outcomes were noted for various benchmarks in the fields of data base, operating systems and mathematics with the use of 'C' language for combinations of different parameters. An optimum model have been developed which gave a consistent performance for all of the above mentioned areas. Control dependence was critical amongst the three areas for achieving better performance. So concentration was kept on the 1-level and 2-level branch prediction scheme in the control dependence. The interfacing of branch prediction unit was done using FPGA with an IP core externally by deploying VLSI technique. The branch predictor unit was evaluated for its performance using FPGA and was verified in order to find a branch predictor unit which was optimum.

Harsh Arora *et al.* in 2013 designed a dynamic branch prediction modeler for RISC architecture [1]. For designing this author has studied that owing to features of RISC architecture, benefits have been taken by computer designers from ILP and deeper pipelines were used by them along with wider issue rates and superscalar techniques. Although a change in the flow of execution of instructions was observed due to the existence of branch instructions. When a branch was encountered then either the pipeline need to be stalled till the execution of branch instructions was going on or a prediction was required to be made regarding the output of the branch which was either the branch should be considered as a *taken one* or *not taken*. By addition of stalls in the pipeline the performance was lost. Technique of branch prediction was deployed in order to reduce this loss of performance. The performance loss was minimized by making prediction regarding the behaviour of branch and issuance of subsequent instructions before actually knowing the outcome of the branch.

3. DESIGN OF BRANCH PREDICTOR UNIT

Figure 1 shows the block diagram branch predictor unit. A branch target buffer (BTB) provides instructions in the predicted path. Pattern history provides the history of the prediction. According to history the selection logic selects whether the prediction is to be made from BTB or normal execution is to be done without any prediction. When the prediction is done from the BTB and conditional branch is there then at the time of execution it is checked whether the condition is true or not. In case of true condition the prediction comes to be true and when condition goes false wrong prediction takes place. Now the value of program counter (PC) is updated again with the next PC value.

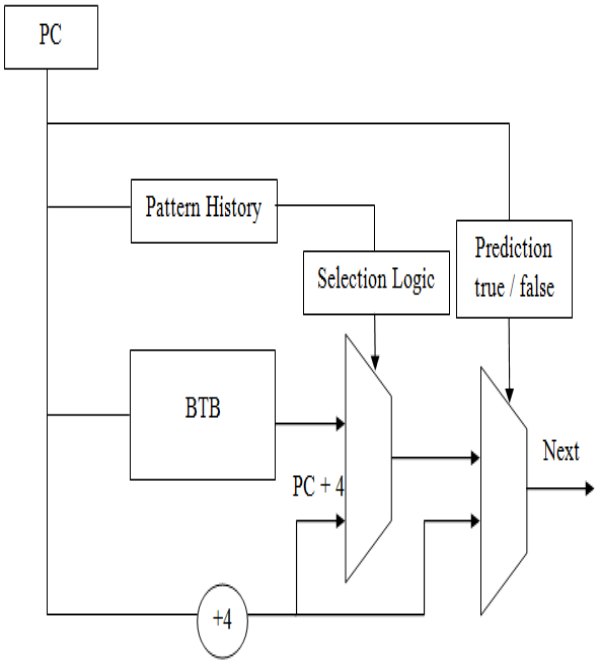

Fig. 1: Block diagram of a branch prediction unit

At the time of misprediction one pipeline cycle is being wasted and the instruction occurring after the branch instruction is firstly fetched and then it is ignored. After execution of branch instruction is done; the fetching is resumed from the address which was resolved.

3.1 BTB (Branch Target Buffer)

This BTB is used to store the address of branch instruction along with the address of branch target. Figure 2 shows branch target buffer in which width of branch instruction address is 16 bits and width of branch target address is also 16 bits and their corresponding depths are 8 bits. Here address 0 is the corresponding to the 0^{th} location and address 7 is corresponding to 7^{th} location.

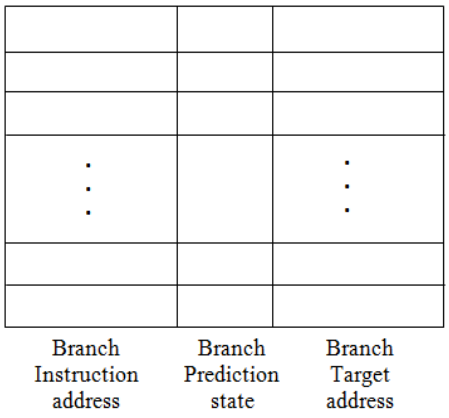

Fig. 2: Branch target buffer

When instruction gets decoded then we get to know whether it is a branch instruction or not. After the result of instruction decoder, branch predictor checks whether the address of branch instruction is already there in the BTB or not. If the address is contained in BTB then the corresponding target address is predicted and gets executed. After this the value of program counter is updated. In case the address is not present in the BTB then at the time of execution of instruction the address of the instruction is stored in the BTB and after execution the target address is stored. And if this instruction is repeated in future the address is already stored and therefore it gets executed thereby saving 2-3 clock cycles required in the execution.

3.2 2- Bit Predictor

Dynamic branch prediction involves prediction which is based on hardware. In this technique a prediction is made based on the direction taken by the branch when it was executed for the last few times. With the utilization of branch history the predictions can be made more accurately. In this method the history for each branch is considered separately and the advantage is taken of the repetitive patterns. A typical branch prediction scheme is shown in figure 3

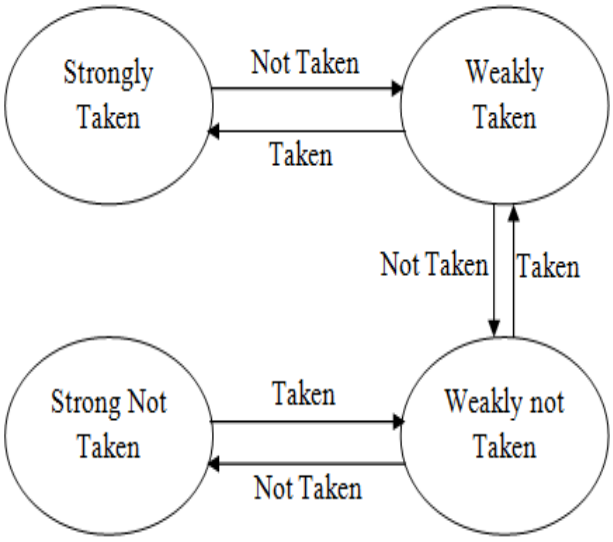

Fig. 3: 2-bit Branch Prediction

In this work, 2-bit dynamic prediction scheme is employed. A 4-state FSM has been used for its implementation. Four states are present namely; *strongly not taken, weakly not taken, weakly taken and strongly taken*. When the execution is taking place, in case of *taken* condition then the state of FSM is *weakly taken*. When execution of the same instruction takes place again then at that time state of FSM changes to *strongly taken* when the condition comes out to be true. In case the condition evaluates to be false then the state changes to *weakly not taken*. When the condition is not taken then the state of FSM is *weakly not taken*. This process continues, for each true condition the state gets incremented while for false condition a decrement in the state is examined. Figure 3 shows how the states change when the execution is taking place. Branch prediction is based on the patterns and current state of the FSM.

To test the designed branch prediction unit, a simple 8-bit RISC processor was designed. To verify the performance of designed branch prediction unit, a program was executed firstly without branch prediction and afterwards with branch prediction.

4. SIMULATION RESULTS

The branch prediction unit was described in Verilog HDL and synthesized using Xilinx Virtex-5 device XC5VLX50T. Simulation results of all the blocks have been carried out using Xilinx ISim simulator.

A processor was designed to perform testing of branch predictor unit. Outputs were examined for varied number of inputs. The same inputs were applied to the processor without branch predictor unit. Figure 4 indicates the simulation waveforms of designed processor with branch predictor unit. An input is applied in hex format and output is taken in accordance with this input value. *Cycle [31:0]* indicates number of clock cycles required in execution of a program. *Cycle [31:0]* bit gets incremented when clock arrives. Now the same input is applied to the processor without branch predictor unit and the value of output is observed. Figure 5 depicts simulation waveforms for the processor without branch predictor unit. The output for both the simulations has been checked. The values of output registers are same for both the simulations but the value of cycle bit is different for both. It can be seen that the values for *reg_r1 [7:0], reg_r2 [7:0], reg_r3 [7:0], reg_r4 [7:0], reg_r5 [7:0]* are same in both the waveforms. In figure 4, the value of *cycle [31:0]* is 470 with branch prediction; while it is 585 without branch predictor unit in figure 5. It has been observed that 115 cycles have been saved with the help of designed branch predictor unit in executing the same program.

The simulation waveforms in figure 6 depict the execution of JZNE instruction. JZNE instruction is a conditional branch instruction. When the decoding process ends we get to know the current state of the instruction whether it is a JZNE instruction. The value in state [7:0] gets updated with 12 which is the value for JZNE instruction. Since it is a branch instruction therefore the branch bit goes high. The execution of JZNE instruction utilizes five clock cycles. The number of clock cycles which are being used is shown in cycle [31:0]. It is noted that its value goes from 12 to 16. When the execution of JZNE instruction takes place again then its target address is predicted by the predictor in advance. As a result of this now only 2 clock cycles are required for the execution of JZNE instruction. Now in the cycle [31:0] the value goes from 20 to 21. Hence three cycles can be saved with the help of branch predictor. At the time of execution, the condition of JZNE instruction is checked. If the condition is true then prediction is correct else the prediction is incorrect. Now *wp* bit goes high indicating that a misprediction has occurred. The value of PC gets updated again with the address of next instruction.

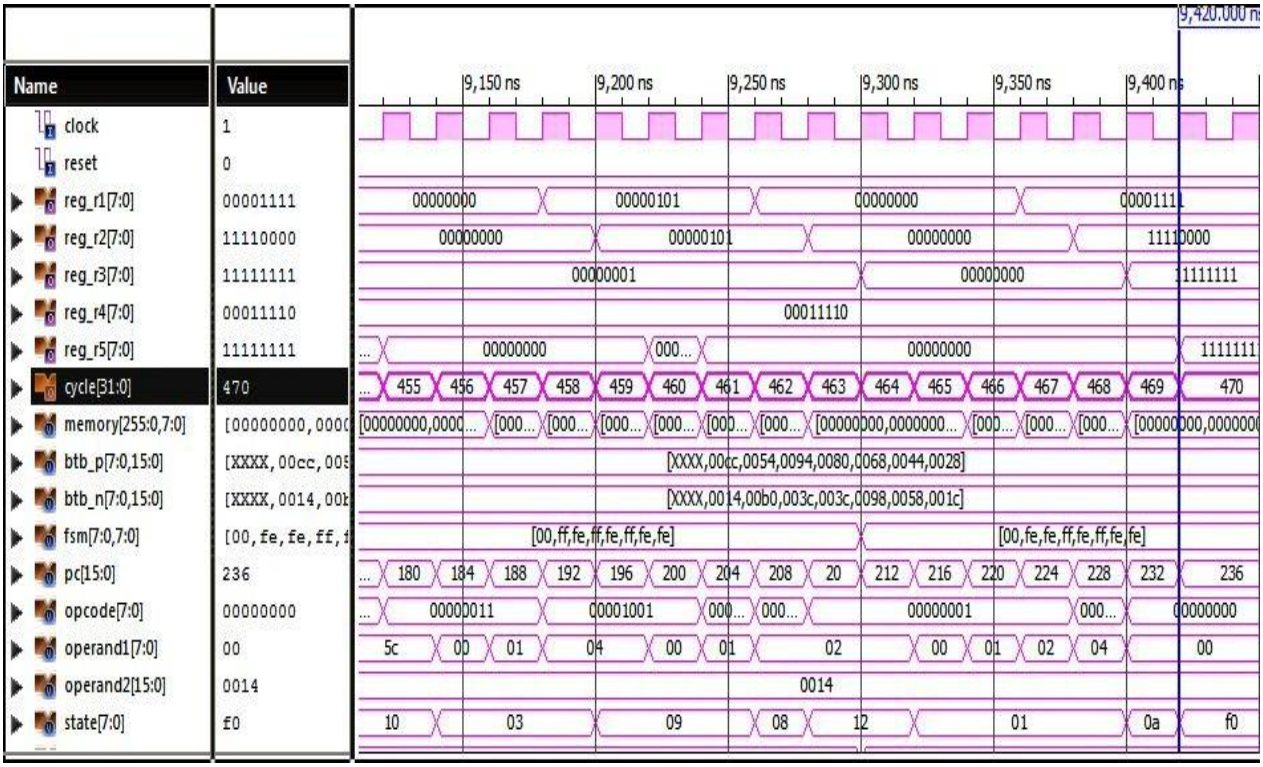

Fig. 4: Simulation waveforms of designed processor with branch predictor unit

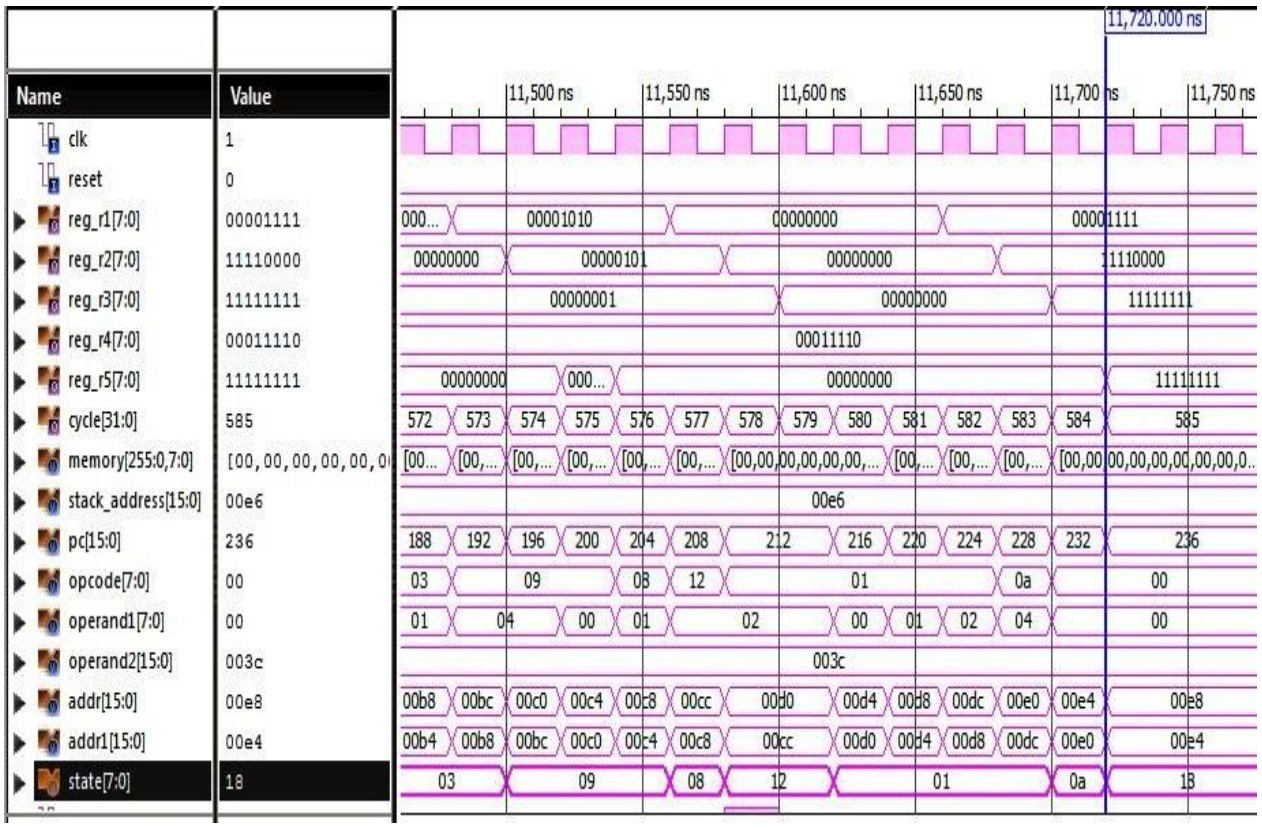

Fig. 5: Simulation waveforms of designed processor without branch predictor unit

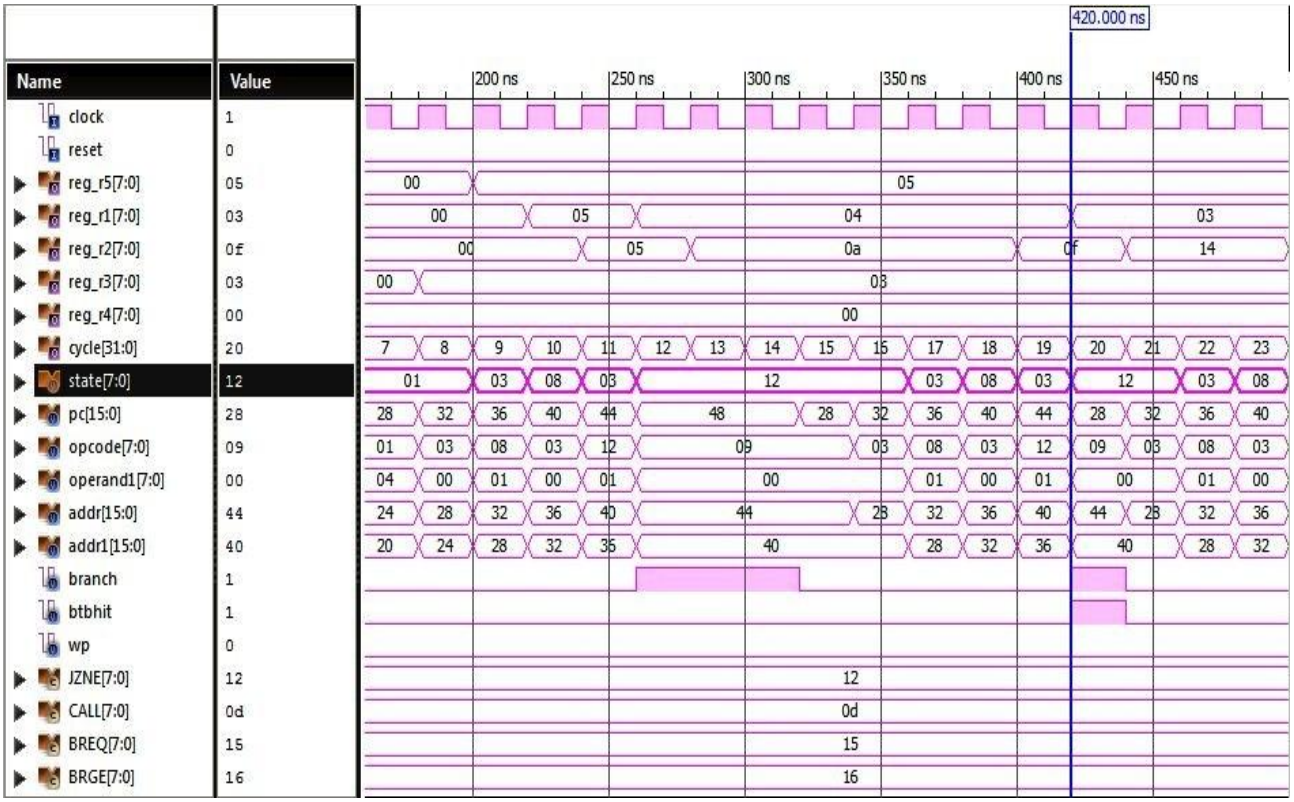

Fig. 6: Simulation waveform of execution of JZNE instruction

By testing different inputs, the latency of different branch instructions has been calculated. Table 1 shows the number of clock cycles saved in executing branch instructions using designed branch predictor. Figure 7 shows the latency for different branch instructions. From the figure 7, it is clear that for the execution of *JZNE* instruction 5 clock cycles are required in processor without branch predictor unit while it takes 2 cycles for its execution in processor having branch predictor unit. So, with every prediction 3 clock cycles are being saved by JZNE instruction. A similar phenomenon is observed for other branch instructions as well.

Table 1: Latency and save clock cycle by predictor

Instruction	No. of clock cycle		Number of clock cycles saved
	With branch predictor	Without branch predictor	
CALL	2	4	2
JZNE	2	5	3
ICALL	2	3	1
JZ	2	5	3
RJMP	2	3	1
JMP	2	4	2
BREQ	2	5	3
BRNE	2	5	3
RET	3	3	0
BRGE	2	5	3
BRLE	2	5	3

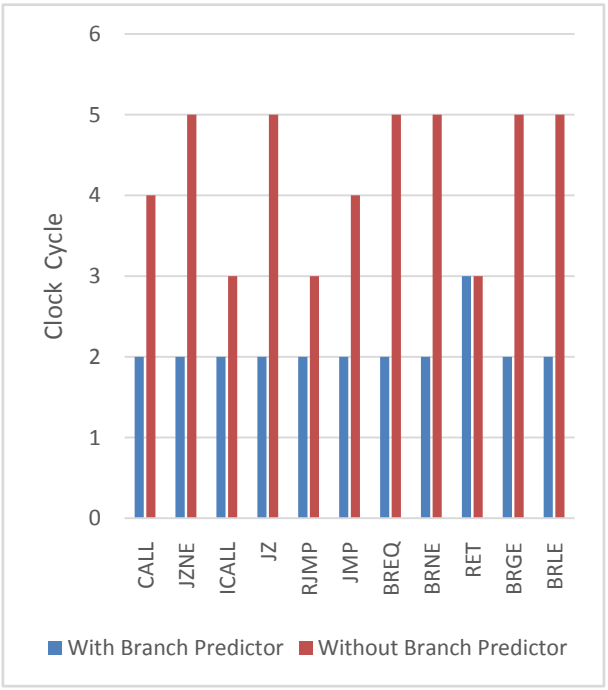

Fig 7: Latency for different branch instructions

The design has been synthesized in Xilinx Synthesis Tool (XST). Table 2 and table 3 show the device utilization summary and the timing summary of the designed processor with and without branch prediction unit respectively.

Table 2: Device utilization summary of the design

Parameter	With Branch Predictor	Without Branch Predictor	Overhead
Number of Slice Registers	4699	4279	8.93 %
Number of Slice LUTs	15331	11385	25.73 %
Number of fully used LUT-FF pairs	877	436	50.28 %
Number of unique control sets	19	14	26.31 %

Table 3: Timing summary of the design

Parameter	Timing summary	
	With Branch Predictor	Without Branch Predictor
Minimum period (ns)	8.359	8.133
Maximum Frequency(MHz)	119.632	122.960
Levels of Logic	17	16

The table 2 shows the device utilization summary of processor with and without branch prediction unit. The overheads to design a processor with branch prediction unit using Virtex 5 (XC5VLX50T) are 8.93%, 25.73%, 50.28 % and 26.31 % for the resources i.e. slice registers, slice LUTs, fully used LUT-FF Pairs and Unique control sets respectively. Table 3 shows the timing summary of the processor with and without branch prediction unit. The maximum frequency at which a processor can be operated with and without branch predictor are 199.632MHz and 122.960MHz respectively. It can be seen that the minimum period is more and the latency of branching instruction is less for the design with branch prediction unit. Therefore it is concluded that the design of a processor with branch prediction unit is more efficient.

5. FPGA IMPLEMENTATION OF THE DESIGN

The Branch prediction unit has been implemented using Xilinx Virtex-5 device XC5VLX50T. This board is having 8 input ports and each port is of 1-bit. 8 LEDs are used for representing an output bit. In the implementation of branch predictor unit these LEDs indicates the values of the registers. Two inputs are used as select lines for checking the values of different registers. The numbers of clock cycles which are used in execution of the program are displayed on the LCD.

Fig. 8: FPGA implementation

6. CONCLUSION

The 2-bit branch predictor was designed using finite state machine (FSM) comprising of four stages- strongly taken, weakly taken, strongly not taken and weakly not taken. Prediction was done based on the status of FSM and branch history. To test the designed branch predictor unit, a simple 8-bit RISC processor was implemented on a processor which has been designed. To verify the performance of designed branch predictor unit, a program was executed firstly without branch prediction and afterwards with branch prediction. The overheads to design a processor with branch prediction unit using Virtex-5 (XC5VLX50T) are 8.93%, 25.73%, 50.28 % and 26.31 % for the resources i.e. slice registers, slice LUTs, fully used LUT-FF Pairs and Unique control sets respectively. The latency of execution of program for both cases was computed and compared. A significant reduction in the latency was observed from the simulation results with branch prediction unit. The maximum frequency at which a processor can be operated with and without branch predictor are 199.632MHz and 122.960MHz respectively. Therefore it is concluded that the design of a processor with branch prediction unit is more efficient.

7. REFERENCES

[1] H. Arora, S. Kotecha, R. Samyal, "Dynamic Branch Prediction Modeller for RISC Architecture," *in Proc. International Conf Machine Intelligence and Research Advancement (ICMIRA)*, pp.397-401, Dec. 21-23,2013.

[2] D. A. Patterson and J. L. Hennessy, Computer Organization and Design: The Hardware Software Interface. Burlington. 4th ed. USA: Morgan Kaufmann, 2010.

[3] D. Orozco, "TIDeFlow: A Parallel Execution Model for High Performance Computing Programs," *in Proc. International Conf Parallel Architectures and Compilation Techniques (PACT)*, pp.211-216, Oct. 10-14, 2011.

[4] C. Egan, "Dynamic Branch Prediction in High Performance Super Scalar Processors," *Ph.D. thesis, University of Hertfordshire*, August 2000.

[5] Dezso Sima, Terence Fountain and Peter Kacsuk, Advanced Computer Architectures: A Design Space Approach. *1st edition Pearson Education Inc.*, 1997.

[6] Tse-Yu Yeh, "Two-Level Adaptive Branch Prediction and Instruction Fetch Mechanisms for High Performance

Superscalar Processors," *PhD thesis, University of Michigan.*, 1993.

[7] J.V Kumar, B Nagaraju, C Swapna, T Ramanjappa, "Design and development of FPGA based low power pipelined 64-Bit RISC processor with double precision floating point unit," *in Proc. International Conf Communications and Signal Processing (ICCSP).*, pp.1054-1058, Apr. 3-5,2014.

[8] P.P Ravale, S. S Apte, "Design of a branch prediction unit of a microprocessor based on superscalar architecture using VLSI," *in Proc. 2nd International Conf Computer Engineering and Technology (ICCET), vol.3, pp.355-360, Apr. 16-18, 2010.*

[9] J.W Kwak, C.S Jhon, "High-performance embedded branch predictor by combining branch direction history and global branch history," *in Proc. IET Computers & Digital Techniques*, vol.2, no.2, pp.142-154, March 2008.