Published by :
http://www.ijert.org

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 8 Issue 06, June-2019

# Design and Implementation of Reorder Buffer for High Performance Processor

Sruthi R S
M.Tech student
Department of ECE,
ER&DCI Institute of Technology,
Trivandrum

Libin T T
Scientist E
Hardware Design Group
C-DAC, Trivandrum

Kadar A A
Principal Engineer
Department of ECE,
ER&DCI Institute of Technology,
Trivandrum

***Abstract*--High performance processor uses pipelining technique and out-of-order instruction execution to increase the instruction throughput. Pipelining is overlapping of instructions inside the processor to increase overall efficiency of the system. The major limitation of in order instruction issue is that if an instruction is stalled, no later instructions can proceed. Out-of-order instruction execution is adopted to overcome this limitation. After out-of-order execution, it is necessary to reorder the results before commit. This project focuses on the VHDL implementation of a reorder buffer for this purpose. Reorder buffer (ROB) allows the instructions to execute out-of-order but forces the result to commit in order. It is a circular buffer with a tail pointer indicating next free entry and a head pointer indicating the instruction that will commit (/leave ROB) first. ROB holds the results of instructions that have finished execution but are not yet committed. It keeps a track of the order of results it receives and passes the value to commit in program order.**

***Keywords -- Reorder Buffer, Out-of-order execution, superscalar processor, VHDL.***

## I. INTRODUCTION

The conventional processors execute the instruction in program order in which they are fetched from the instruction memory. But if an instruction is stalled in the pipeline, no later instructions can proceed. So to avoid this problem and to increase the performance modern processors execute instructions in out-of-order. To enable out-of-order execution, the instructions are fetched in program order and also retired it in program order. A reorder buffer helps in retiring the instruction in program order. This paper focuses on the VHDL implementation of Reorder buffer. Reorder buffer is a circular buffer with a head pointer and a tail pointer. The head pointer indicates which instruction will leave the ROB first and the tail pointer indicating next free entry. In the superscalar processor, more than one instruction is executed in a single clock cycle.

In superscalar processors, the instructions are fetched from a sequential instruction stream. The CPU checks the data dependencies during the run time. There are different types of data dependencies that may lead to hazards. Such hazards are called Data hazards. These hazards can be eliminated by a technique called register renaming. In register renaming, the destination registers are renamed by temporary register names and thereby eliminates these types of hazards.

The paper is organized as follows. The related paper works in the design of ROB is provided in section II. The basics of Reorder Buffer are given in Section III. The block diagram and flowchart of the proposed ROB are given in section IV and V. Results and discussions are given in section VI. Finally, conclusion is given in Section VII.

## II. RELATEDWORK

Embedded systems based on Field- Programmable Gate Arrays (FPGA) exhibits more performance for new applications. High-performance superscalar processors execute instructions out-of-order and it is necessary to reorder the results after execution. This task is performed by the ROB (Reorder Buffer) that uses usually multiport RAM, but only two-port buffers are available in FPGA. An FPGA friendly ROB (Reorder Buffer) architecture using only 2 ports RAM called a multi-bank ROB architecture was proposed [1]. Bluespec generates synthesizable Verilog RTL from high-level models, verification IP, transactors, and production IP. It provides a high-level of abstraction for hardware design by leveraging atomic transactions. It is the single most powerful tool for managing complex concurrency. The design of the reorder buffer for an out-of-order superscalar processor with a MIPS I ISA using Bluespec was explored [2]. Modern reorder buffers (ROBs) were designed to improve processor performance by allowing instruction execution out of the original program order by exploiting existing instruction level parallelism (ILP). The ROB is an important structure in processor execution engine that supports speculative execution, physical register recycling, and precise exception recovering. ROB is considered as a monolithic circular buffer with incoming instructions enters at the tail pointer after the decoding stage and retiring instructions at the head pointer after the commitment stage. This design is suitable for both embedded and high-performance microprocessors due to the reduction in the area which reduces the power and delay [3].

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 8 Issue 06, June-2019

## III. REORDERBUFFER

A reorder buffer contains opcode, destination register address and PC address of the issued instruction in the program order. It is responsible for storing the result in the order in which they are executed (out-of-order), updating the status of completion flag, determining when these instructions are ready to commit and committing the result in program order. ROB also handles branch misprediction, exception and interrupt. If a branch misprediction is detected, then the ROB allows removing all instructions on the wrong path (i.e., younger than the branch) while keeping older instructions. If an exception occurs, then the exception cause is placed in the ROB and handles the exception when all the previous instructions commit from the ROB. Similarly, if an interrupt occurs, then the ROB immediately handles the interrupt by removing all the instructions in the ROB. As a result, the ROB is really a speculative frame representing the near future state of the program.

## IV. BLOCK DIAGRAM

### A. *ROB Entry*

Each entry in the ROB has several fields. Some fields are shown below:

- Intr_type : Type of the instruction that is issued from the instruction queue
- Dest_addr: Destination register address of the issued instruction.
- result: The data from the data bus after execution.
- br_cond: Whether the branch is correctly predicted or not.
- PC: The PC value of the issued instruction.
- Exception : It indicates whether an exception/interrupt occurred or not in the processor
- exp_cause: It indicates what type of exception occurred, so that the processor can handle the exception.
- cflag: To indicate whether the execution is completed or not. After that the data will be committed.

Additional pointers are required to indicate the entry and exit from the ROB. Head_ptr is used to indicate which instruction will commit/leave the ROB, Tail_ptr is used to indicate where the next instruction will enter the ROB.

### B. *Block Diagram*

Fig 1 shows the block diagram of the proposed ROB.

- Rst: To reset the buffer.
- Clk: To provide clock to the buffer.
- d_in: Instruction issued from the instruction register contains the opcode and the address of the destination register.
- d_DB1, d_DB2, d_DB3: Value of the data from the 3 Data Bus (DB) after execution.

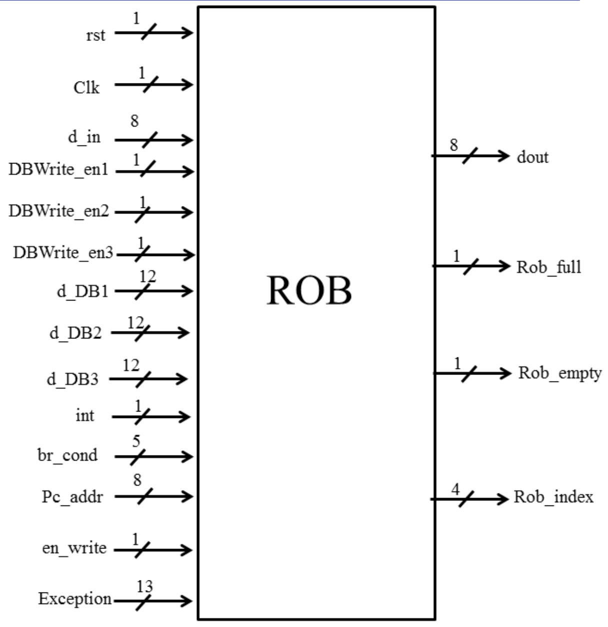

Fig. 1. Block diagram of proposed ROB

- Rob_full: To indicate whether the ROB is full or not.
- Rob_empty: To indicate whether the ROB is empty or not.
- br_cond : To indicate whether the branch is correctly predicted or not
- Pc_addr: PC value of the issued instruction
- Rob_index: To indicate which ROB entry is free.
- en_write: Data enters the ROB when it is high.
- int : To indicate whether there is an interrupt or not
- Exception : To indicate whether there is an exception occurred or not

## V. FLOWCHART

### A. *Write operation*

Fig 2 shows the flow chart of write operation in ROB. Initially there is no instruction in the ROB, so Rob_full is zero and Rob_empty is high. When an instruction is entered Rob_empty becomes low. From the decoded incoming instruction, separate the opcode and destination register address and are written into the fields of ROB. Then the Tail_ptr is incremented by one to indicate that the next instruction will enter into the ROB where Tail_ptr points. Then check for the condition "Head_ptr = Tail_ptr + 1". If the condition is true, then Rob_full becomes high; else check whether any other instruction entered. If the Rob_full is high, then no latter instruction will enter into the ROB. The next data will enter into the ROB only when an instruction commits. In this processor, there are three separate data buses. If the result is entered, then place the result in the corresponding location and set the completion flag to high for indicate the valid result in the ROB. Repeat the process for next instructions entered in the ROB.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 8 Issue 06, June-2019

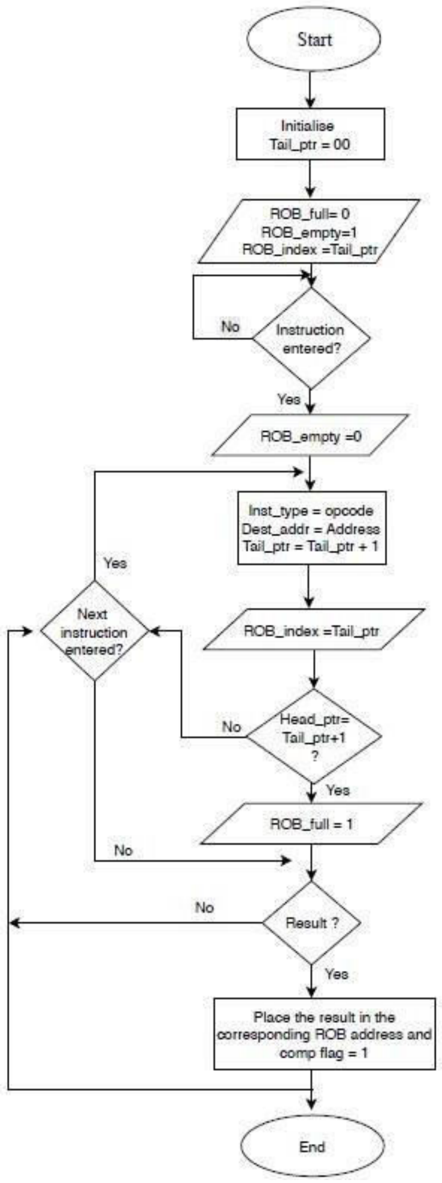

Fig. 2. Flow chart for write operation in ROB

### B. *Read operation*

Fig 3 shows the flowchart of read operation in ROB. First, check for the completion flag. If it is set, then check for the read enable bit for data to the reservation station; else wait for the result for set the completion flag. If the read enable is set then send the data to the reservation station. Then check whether the Head_ptr points to the corresponding ROB address. If it is set, then commit the data and Head_ptr is incremented by one for committing the next data. Then check the condition Tail_ptr = Head_ptr + 1. If it is set indicate there is no instruction, hence ROB_empty becomes high.

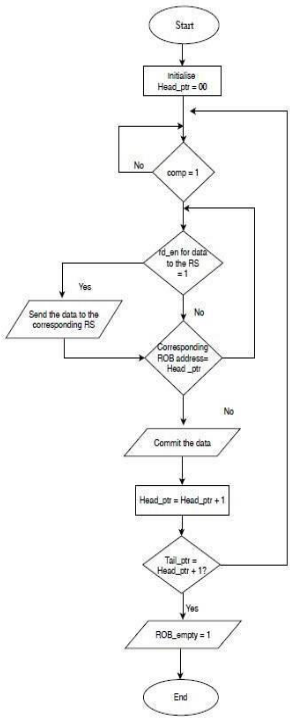

Fig. 3.Flowchart for read operation in ROB

### C. *Branch Instruction*

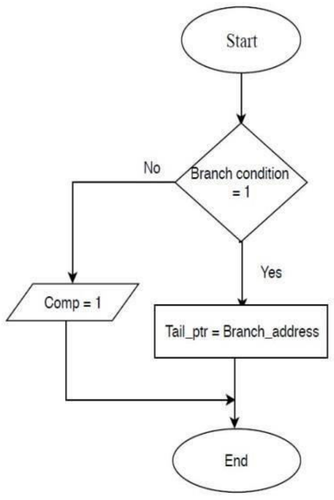

Fig. 4. Flowchart for operation of branch instruction in ROB

Fig 4 shows the flow chart for branch instruction. If the branch condition is high indicates there is a branch misprediction, and then all the instructions in the ROB next to branch instruction is incorrectly executed. So the instructions must be flushed. For that, the Tail_ptr is placed to the ROB entry with branch instruction. If the branch misprediction is true, then set the completion flag to high.

### *D. Exception handling*

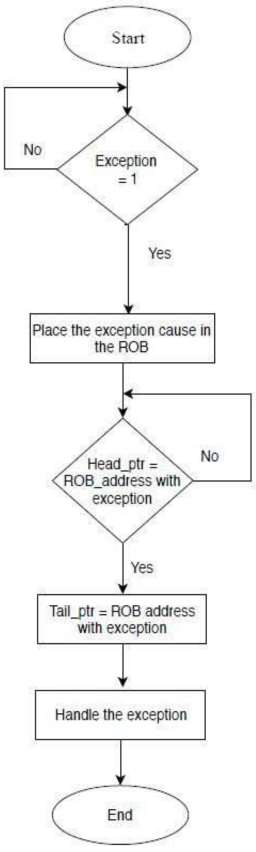
Start
Exception = 1
No
Yes
Place the exception cause in the ROB
Head_ptr = ROB_address with exception
No
Yes
Tail_ptr = ROB address with exception
Handle the exception
End

Fig. 5. Flowchart for exception handling in ROB

Fig 5 shows the flowchart for exception handling in ROB. If the exception is high, then place the exception cause in the field of ROB. Exception cause is placed in the field of ROB to acknowledge the processor about the type of exception. If the Head_ptr points to the ROB entry with exception indicates that all previous instructions are committed. The Tail_ptr is placed to the ROB entry with exception and handles the exception

### *E. Interrupt handling*

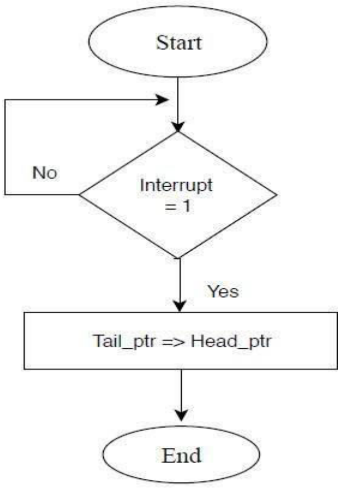

Fig. 6. Flowchart for interrupt handling in ROB

Fig 6 shows the flowchart for interrupt handling. If the interrupt is high then the Tail_ptr is placed to the ROB entry where Head_ptr points and then handle the interrupt

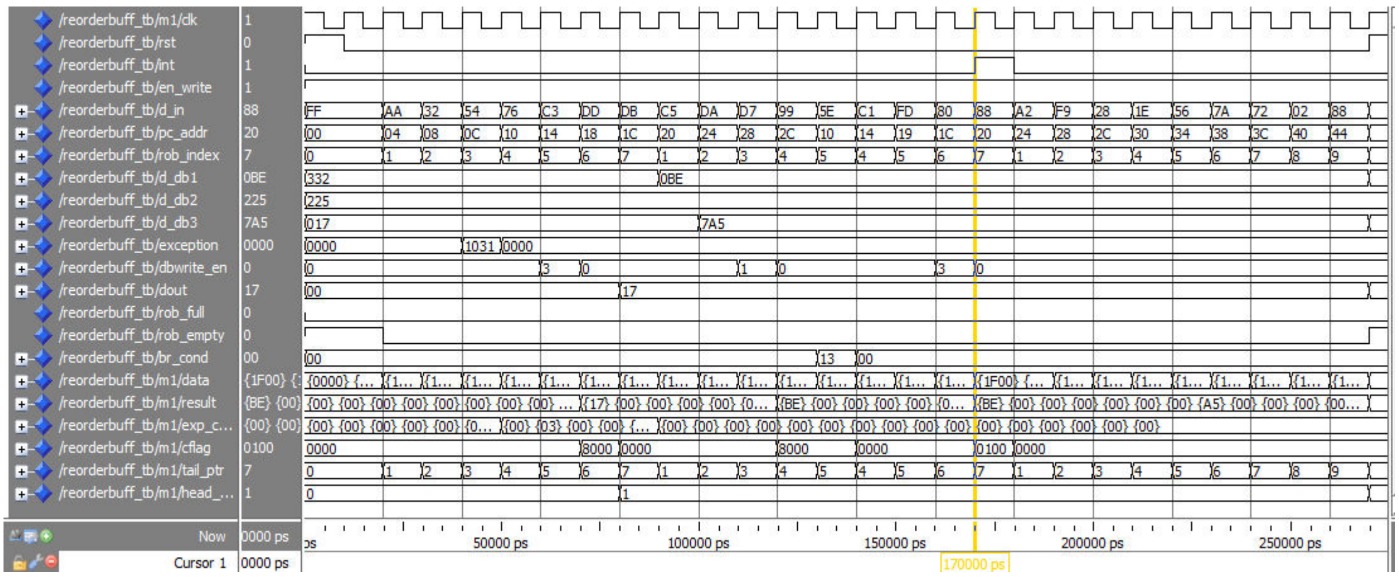

Fig. 7. Simulation waveform of the proposed Reorder Buffer

## VI. RESULTS AND DISCUSSIONS

The designed ROB is simulated using Questa Sim. Fig 7 shows all the signals in the proposed ROB. When the "rst" is high, all the signals become low. "d_in" is the 8 bit input data where last 3 bit represents the opcode and 2 bit represents the destination register address." PC_addr" represents the program counter address value which represents the address of the next instruction to be executed. "data", "result", "exp_cause" and "cflag" are the arrays created to store the decoded input data, result, the exception cause and the valid completion flag. "br_cond" represents whether there is a branch misprediction occurred or not. "exception" represents whether there is exception occurred or not, what is the cause of exception and to which ROB address the exception occurred. "int" represents whether interrupt occurred or not. When this becomes high the processor will suspend all the current execution and handles the interrupt. Head_ptr and Tail_ptr are two pointers which indicate the instruction that is to be committing first and the next instruction entry in the ROB respectively. "Rob_full" and "Rob_empty" are the output signals which indicate the processor that ROB is fully occupied and the ROB is free respectively. "Rob_index" is the output signal which indicates the processor for the next free entry in ROB. The designed ROB is synthesized using Xilinx ISE and successfully implemented in FPGA.

## VII. CONCLUSION

Processors execute instructions in out-of-order to increase the performance and the instruction throughput. A Reorder buffer is used to commit the result in program order in out-of- order processors. This paper focuses on the VHDL implementation of Reorder buffer. The basis of reorder buffer and the algorithm related with the design of reorder buffer have been studied. The VHDL coding of Reorder buffer for instruction entry in superscalar processor has been completed and verified using test bench. Also coded and verified the Reorder buffer for branch instruction. The designed ROB handles the exception and interrupt. Then the ROB is synthesized successfully. After synthesis the ROB is implemented where translation, mapping and place and route are successfully completed for the specified timing constraint.

## VIII. REFERENCES

[1] Mathieu Rosiere, Jean-louDesbarbieux, Nathalie Drach, Franck Wajsburt ”*An Out-of-Order Superscalar Processor on FPGA: The ReOrderBuffer Design* ”,Design, Automation & Test in Europe Conference & Exhibition (DATE), 2012.

[2] Nirav Dave, ”*Designing a Reorder Buffer in Bluespec*” ,Second ACM and IEEE International Conference on Formal Methods and Models for Co-Designl,Vol. 3, Issue 3, 2004

[3] Jos R. GarcaOrdaz, Marco A. Ramrez Salinas, Luis A. Villa Vargas, Hern Molina Lozano, and CuauhtmocPeredoMacas, ”*A Reorder Buffer Design for High Performance Processors*”, Computacin y Sistemas, Vol. 16 , Issue 1, 2012.

[4] S. Weiss, J. E. Smith, ”*Instruction Issue Logic in Pipelined Supercomputers*”,IEEETransactions on Computers,Volume 33 ,Issue 11, November 1984

[5] R M Tomasulo, ”*An Efficient Algorithm for Exploiting Multiple Arithmetic Units*” ,IBM Journal of Research and Development,Volume 11 ,Issue 1,January 1967

[6] Andrew Waterman, KrsteAsanovicSiFive Inc., ”*The RISC-V Instruction Set Manual*”,RISC-V User-Level ISA V2.2,Volume I, May 2017