Optimization and verification of Bus Interface Unit (BIU) to increase performance of the in-house developed 32-bit RISC processor core

Leema Rachel Mathew[1], Roopashree[2]
[1]*M.Tech, VLSI and Embedded System, SCEM Mangalore*
[2]*Asst Prof Dept of ECE, SCEM Mangalore*

***Abstract* – In this paper, we propose optimization, verification and FPGA prototyping of BIU by analyzing and reducing the number of clock cycles required for the execution of each memory access for instruction and data without changing the existing functionality of the processor so as to enhance its performance.**

***Index Terms* – BIU, Optimization, FPGA prototyping**

I. INTRODUCTION

The Bus present in the architecture is the common pathway through which the information flow takes place between the various components or interfaces to which it is connected. The BIU present in the processor core provides the functionality of generating I/O address to transfer data to and from the memory. The BIU in the processor is responsible for responding to all signals that go to the processor, and generating all signals that go from processor to the other parts of the system.

The processor performance also depends on the speed of the BIU i.e. the time taken by the BIU to transfer the data to and from the external memory. Hence by increasing the speed of the BIU the performance of the processor can be increased. The processor architecture analyzed in this paper is interfaced to two external memories namely instruction memory and data memory. Each memory is accessed through the BIU by the processor. The speed of the access is dependent by the number of clock cycles taken to execute each statement in the BIU code written to implement the desired BIU architecture. The optimization aims at the reduction in the number of clock cycles required to execute each instruction.

After analysis it was found that the BIU of the in-house developed 32-bit RISC processor takes 6 clock cycles for instruction fetch and 13 clock cycles for data fetch. The aim of this paper is to optimize the BIU present in the 32-bit RISC processor is optimized to fetch instruction and data from the memory in one clock cycle rather than the number of clock cycle mentioned before hence improves the performance of the processor. The optimization is done by step wise reducing the number of clock cycle for fetching instruction and data and verifying the functionality.

II. ANALYSIS OF EXISTING BIU

[5]The read and write into the memory is controlled by the signal RdWr_. During the read operation the RdWr_ signal is set high. The As_ and Ds_ signals are asserted to indicate valid address and data are available on the address and data bus. The address bus will have valid address and the data bus will be provided with the valid data by the external memory based on the address sent by the BIU. Once the valid data is obtained from the peripheral then the Dsack_ input signal is set low to terminate the cycle initiated by the BIU. The asynchronous Dsack_ signal is generated by the external decoder unit which is outside the processor. After the Dsack_ signal is asserted the As_ and Ds_ is de-asserted for the next clock cycle and then a new cycle is initiated by the BIU with the next address.

Similarly, during write operation the signal RdWr_ is set low. The As_ and Ds_ signal are asserted. The address bus will have valid address and the data bus will give the data from the processor to write into the memory. Once the data to be written into the memory is written the Dsack_ signal is set for one clock cycle. After the Dsack_ signal is set As_ and Ds_ signals are de-asserted for next clock signal and then a new transaction is generated by the BIU.

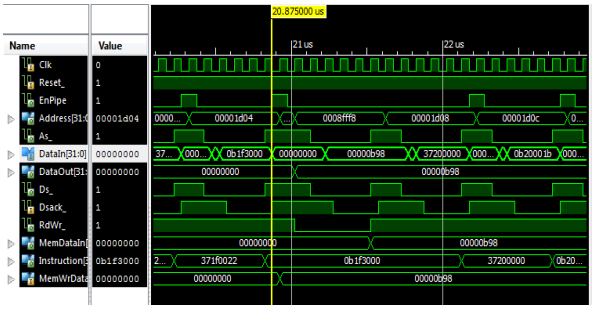

Fig.1. RdWr_ operation in processor through BIU

In this processor for each memory access based on data memory or instruction memory it takes 13 and 6 clock cycles respectively through BIU. The BIU takes 6 cycles to fetch the instruction from external memory and takes 13 cycles to load or store data to external memory. Though the data access from memory is restricted by the access time of the memories but in this case even if the external memory was capable of providing within one cycle access time, the processor BIU used to take the above mentioned number of cycles for instruction and data access. This latency in BIU was due to its internal state machine implementation. So even if the core was provided a clock of xMHz the idle performance of the core would be (x/6)MHz for instruction access and (x/13)MHz for data access.

III. BIU CODE OPTIMIZATION

EnPipe is an internal signal generated by the BIU used by the processor core to move forward the pipeline. Initially the EnPipe was dependent on asynchronous Dsack_ signal. In order to reduce the number of clock cycle the dependency of EnPipe to the Dsack signal had to be removed. So that EnPipe is only dependent on the state machine. Once the EnPipe is made independent of Dsack signal the cycles were reduced by removing and optimizing the states for the state for both instruction fetch access and data fetch access path separately. The following section lists the stepwise approach of optimization.

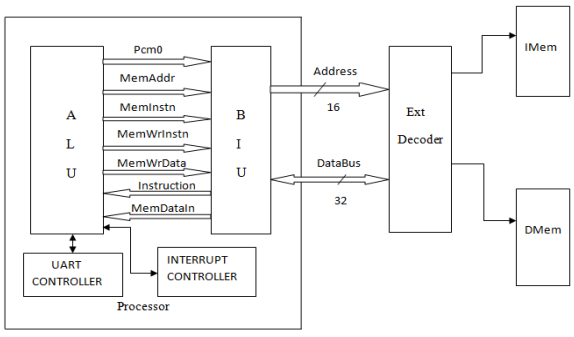

Fig.2. Processor Architecture

A. *Making Dsack synchronous from asynchronous*

The Dsack signal which is an input to the BIU is initially asynchronous. When As_ and Ds_ are set the Dsack_ signal is set. Dsack_ signal remains low till the As_ and Ds_ goes high. This dependency is done by removing double flopping of the asynchronous Dsack_ signal. Initially since Dsack_ was asynchronous so double flopping of the Dsack_ signal was done internally to remove metastability. Making Dsack_ synchronous from asynchronous removed the double flopping requirement and hence enhances each transaction done by BIU by 2 clock cycles. So after this modification the instruction fetch cycles reduced to 4 from 6 cycles and data fetch or store cycles reduced to 11 from 13.

B. *Removing the dependency on Dsack*

After making Dsack from asynchronous to synchronous the number of cycles was found to be 4 and 11 for instruction fetch and data fetch respectively. The state machine was then optimized to remove the dependency on Dsack signal by fixing the number of states required to achieve the cycle requirement so that the state can be further removed to reduce the number of clock cycle required.

C. *Reducing the instruction fetch cycles*

The instruction fetch takes 4 clock cycles after removing the dependency on the Dsack signal. Now the cycles are only dependent on the state machine. The clock cycle was reduced from 4 to 3 by making changes in the state machine. The variation in the functionality is checked and the BIU code is modified as per the required functionality. Each time the reduction in the clock cycle is done the functionality is checked and the required modifications are done in the BIU code to retain the functionality. The instruction fetch cycles were reduced finally to 1 clock cycle by reducing the number of states and analyzing and resolving the dependencies encountered by doing the modification.

D. *Reducing the memory access cycle*

The BIU takes 11 cycles for the memory access. After removing the dependency on the Dsack the number of clock cycles reduced to 8 by removing the dummy states present in the previous BIU design. Similar to instruction fetch cycles the memory access cycle is also reduced one at a time. Each time the clock cycle is reduced the functionality is checked and the modification is done. The memory access cycle is reduced to 1

cycle by the step wise approach and the functionalities were also checked to remove the dependencies.

Fig.2 shows the previous architecture of the processor core and Fig.3 shows the modified architecture for single cycle optimization of the BIU.

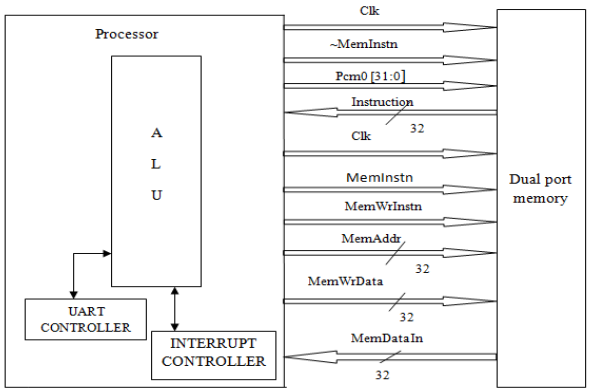

Fig.3. Modified Processor Architecture

IV. BIU CODE STANDALONE VERIFICATION

The standalone verification is done by writing test bench for the code and driving each input and checking if the output signals are generated as per expectation or not. The BIU code inputs and outputs are studied. The way the output changes as the input changes is also studied. Once the relation between the input and the output of the BIU code is studied then the standalone verification is done. The standalone verification was done by writing test bench for the BIU and the required input is given with different values and the outputs are observed. The output responses were checked and the required outputs are obtained.

V. BIU CODE VERIFICATION INTEGRATED WITH THE PROCESSOR CORE

The BIU and the ALU along with the UART controller and the interrupt controller are part of the processor core. The processor core is interfaced to two external memories namely instruction memory and data memory through an external decoder (Ext Decoder). The external decoder decodes address ranges sent out by the processor core and generates the chip selects for the instruction memory and data memory.

The BIU code optimized and verified stand alone was integrated with the processor core in the above mentioned verification setup by replacing the old BIU module with the new one. Reset was then applied to enable the processor to re-execute the instructions loaded in instruction memory from beginning.

VI. SINGLE CYCLE EXECUTION MODIFICATION AND VERIFICATION

The processor initially consists of a separate BIU and External Decoder. The BIU takes both the program memory and data memory address from the processor. The BIU then based on MemInstn signal (Memory Instantiation) signals decides whether to access data memory or program memory and therefore the address is multiplexed internally. A 32 bit address is generated by the BIU and this 32 bit address is given to the External Decoder which is outside the processor. The external decoder gives this address to the program memory or data memory.

Here the program memory and data memory are separate for the processor. The outputs from the memory are then given to the External Decoder. The External decoder gives the data to the BIU which is then given back to the processor core. The data which has to be written into the memory is also sent from the BIU to the memory through the External decoder.

The loading, storing and fetching the instruction from the memory is done through separate BIU code and the External Decoder. Instead of having separate BIU code this code can be integrated to the processor core so that the memory access can be done directly by the processor based on the requirement. The signals from the BIU to the instruction memory and data memory are identified. These signals are to be mapped from processor directly to the memory.

VII. FPGA PROTOTYPING

FPGA prototyping of the processor core with the newly modified BIU was done at every stage of the modification done in a step wise approach as mentioned before to test the working of the modified BIU in the processor core on FPGA. The instruction memory and the data memory as mentioned in the verification section of the integrated processor core section were replaced with BRAMs available on FPGA [4].

Xilinx Virtex5 FPGA based prototyping platform was used to validate the processor core with the modified BIU. Oscillators of 20MHz frequency available on the prototyping platform board was used as input to the PLL used in the FPGA prototyping setup to generate frequencies of 100MHz and 1.84MHz used as processor core

frequency and UART controller frequency respectively.

VIII. SIMULATION RESULTS

Fig.4 shows the simulation result obtained after making the Dsack signal synchronous from asynchronous.

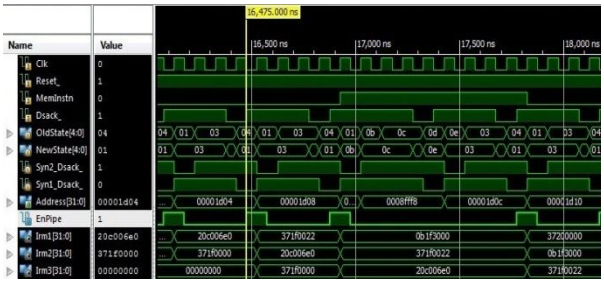

Fig.4. Simulation result in which Dsack is synchronous

Fig. 5 shows the simulation result obtained by removing the separate BIU and External Decoder.

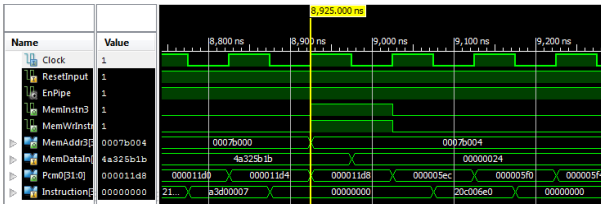

Fig.5. Simulation result without BIU and External Decoder

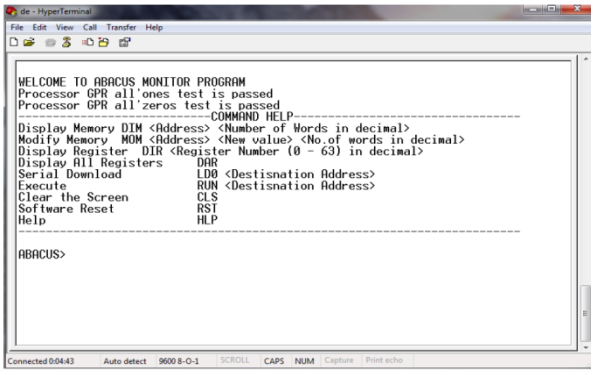

Fig.6. Debug code brought up after FPGA prototyping.

IX. CONCLUSION

The BIU plays an important role in the processor architecture. The performance of processor was increased by optimizing the BIU and verifying the optimized code.

REFERENCES

[1] John L Hennessey, David Patterson, Ed.4, *Computer Organisation and Architecture*, McGraw-Hill, 2012

[2] Steve Furber, Ed.2, ARM *processor Architecture*, Addison Wesley, 2000

[3] Verilog Language Reference Manual

[4] Block RAMs on FPGA – Xilinx app note

[5] In-house developed processor user manual

[6] Xilinx virtex5 FPGA manual

[7] QuestaSim user manual