Functional Verification of Enhanced RISC Processor

SHANKER NILANGI[1]

[1]Assistant Professor, Dept of ECE, Bheemanna Khandre Institute of Technology, Bhalki, Karnataka, India
s.nilangi@gmail.com[1]

SOWMYA L[2]

[2]Assistant Professor, Dept of TE, M.V. Jayaraman College of Engineering, Bangalore, Karnataka, India
sowmyalm@gmail.com[2]

ABSTRACT- This paper presents design and verification of a 32-bit enhanced RISC processor core having floating point computations integrated within the core, has been designed to reduce the cost and complexity. The designed 3 stage pipelined 32-bit RISC processor is based on the ARM7 processor architecture with single precision floating point multiplier, floating point adder/subtractor for floating point operations and 32 x 32 booths multiplier added to the integer core of ARM7. The binary representation of the floating point numbers employed in the design eliminates the need for floating point registers and uses same set of registers thereby reducing the complexity, area and cost. Mask based data reversal barrel shifter performs parallel flag computations during shift or rotate and has least worst case delay of 0.94 ns compared to other barrel shifters. The hardware of the 32-bit RISC processor core has been modeled in Verilog HDL, simulated in VCS. Verification of a complex design such as 32-bit RISC is one of the major challenges as it consumes more time. In this work, a verification environment is being developed to verify the design RISC processor core.

Keywords: ARM7, RISC, Functional Verification

1. Introduction

The market for embedded systems is huge and is growing rapidly compared to the general purpose computers. Low power embedded CPU market which forms the largest market for processors is being dominated by the RISC processors. This is due to the architectures of RISC, improvements in the semiconductors and embedded platforms of RISC. A small size, simple, good code density and low cost ARM RISC processor can be integrated with ease in ASIC. The modular nature of the ARM architecture provides flexibility to add features onto the ARM core such as on chip memory, MMU, FPU and so on for building application specific processors based on ARM. ARM7 32 bit processor belonging to the family of ARM processors used in application specific integrated circuits has a simple structure and allows faster execution of programs. The instruction set supports conditional execution of instructions which allows straightforward programming of assembly code making them ideal for high level language compilers. The ARM7 integer core architecture is simple and ideal for designing and is used in most of the embedded applications.

2. Design of 32-bit Enhanced RISC Processor core Architecture

Among the ARM processors, ARM7 architecture has been found to be simple and ideal for designing. The ARM7 owes its success to the combination of low power, low cost and high performance and is used in embedded based applications like mobiles, PDAs, iPods, pagers, personal audio, etc. The designed RISC processor architecture is based on the ARM7 core.

2.1 *ARM7 Architecture*

The ARM7 has a 32-bit load store architecture where most of the operations are performed by loading the value into registers from memory and storing the result from registers to memory. Since it is 32-bit architecture, it has 32-bit data bus for transferring data and 32-bit address bus for transferring address. Instructions are also of 32-bits. The architecture has two principal units which form the heart of the processor, data path consisting of units such as ALU, barrel shifter, multiplier, register file and other modules in order to perform sequence of operations and decoder & control logic to provide control signals to carry out the operations as shown in the block diagram of the Figure 1 for ARM7 [1].

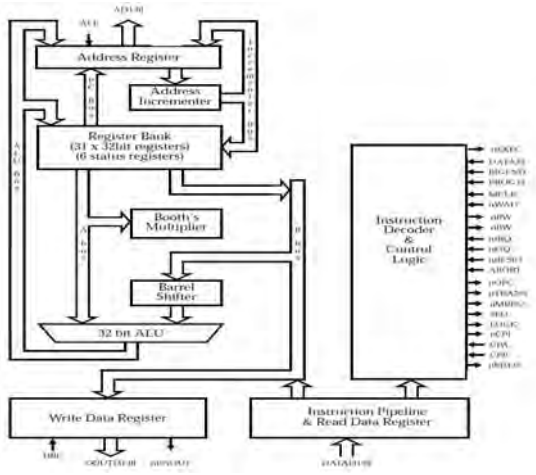

Figure 1: Block Diagram of ARM7 core [1].

The ARM7 core has 3 pipelining stages fetch, decode and execute as shown in the Figure 1. It has register bank of 37 registers made up of 31 general 32- bit registers and 6 status registers. It makes use of Von Neumann architecture with Big and Little Endian formats for storing. The operating frequency is in the range of 0 to 80 MHz with CPI of 1.9. It has a 32x8 booths multiplier and operations can be performed using 6 operating modes. The processor performs fully static operations. The simple yet powerful instruction set provides support for virtual memory and high level language of RISC processor core architecture

Suitable modifications have been made to the ARM7 processor architecture for designing the 32-bit enhanced RISC processor core. It has 3 stage pipelining and Von Neumann architecture with Big Endian format. The block diagram for the design is as shown in the Figure 2. The modifications to the ARM7 core shown in Figure 1 have been indicated in the Figure 2 which include single precision floating point multiplier [2], mask based data reversal barrel shifter [3] and single precision floating point adder / subtractor.

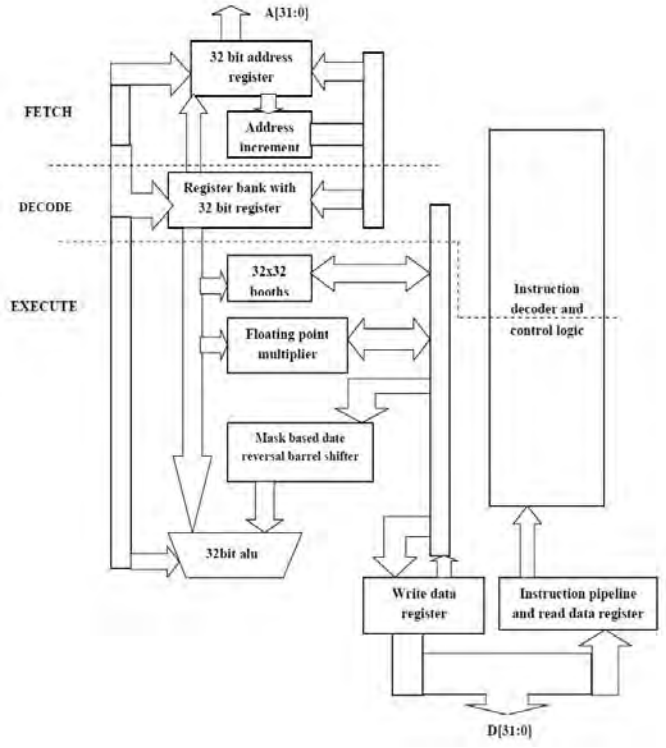

Figure 2: Block diagram of the designed 32 bit RISC processor core.

32 x 32 booths multiplier has been used to reduce multiplication cycles to 4 cycles compared to 32 x 8 booths multiplier in conventional ARM7 processor which require 7 cycles. 32-bit carry look ahead adder has been used to achieve faster speed because of logarithmic delay increase compared to ripple carry adders where delay

increases linearly with the operand size. The register bank has 32- bit registers with 8 registers for storing binary representation of floating point numbers, 8 registers for integers, 32-bit program counter and 32-bit current program status register. The designed processor operating at 50 MHz clock rate has a 1024 x 32 memory for storing the data and instructions. The processor has a 32-bit ALU and is capable of executing 33 instructions each of 32-bits.

3. Hardware Modeling of 32-bit Enhanced RISC Processor core

Hardware modeling plays a major role for efficient implementation of logic circuits. The 32-bit enhanced RISC processor core has been modeled in verilog HDL and simulated using VCS. For the design of the 32-bit enhanced RISC processor, modifications have been made in the instruction formats of ARM7 in order to accommodate the floating point addition, subtraction and multiplication operations. The designed processor has 5 opcode bits in the instruction format for data processing instructions for including floating point addition and subtraction operations unlike ARM7 which require 4 opcode bits.

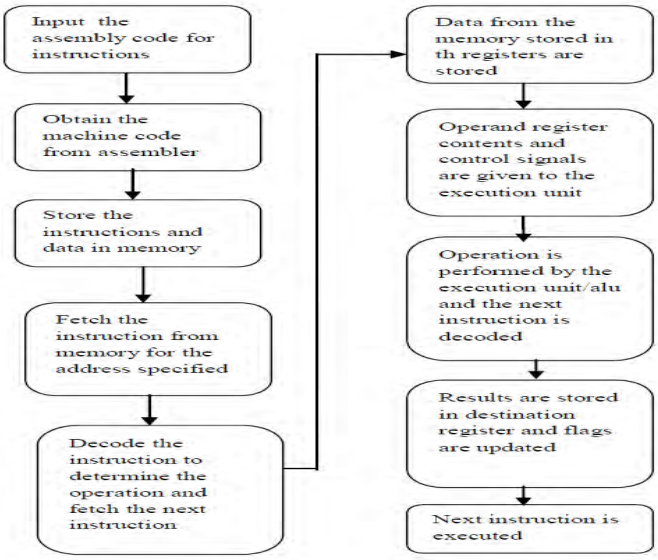

Figure3: Flow chart of the designed 32 bit RISC processor core.

The design of the 32-bit RISC processor architecture core has the pipeline stages fetch, decode and execute. The flow chart for the design of the enhanced RISC processor core is as shown in the figure 3. The assembly code for the instruction (for example ADD R8, R0, R1) is converted to 11000000001000010000000000000001 with the help of assembler. The machine codes for the instructions belonging to the instruction set of the designed RISC processor are stored in the memory along with the 32-bit data values during memory write operation. Then when the address for the instruction to be executed is specified, the corresponding instruction is fetched from the memory. Then in the decode stage, the operation to be performed is identified, the data values in the identified operand registers are read and these values are given to the execute stage along with the corresponding control signals. When the current instruction is in decode stage, the next instruction is fetched. In the execute stage of pipelining, the operation arithmetic or logical is performed on the integer or floating point data values obtained from the decode stage. While the current instruction is in execute stage, the next instruction will be decode. The corresponding result is stored in the destination register and the conditional flags are updated. After the execution of the current instruction, the next instruction will be executed.The simulation results for the RISC processor obtained from VCS have been shown in the Figure 4.

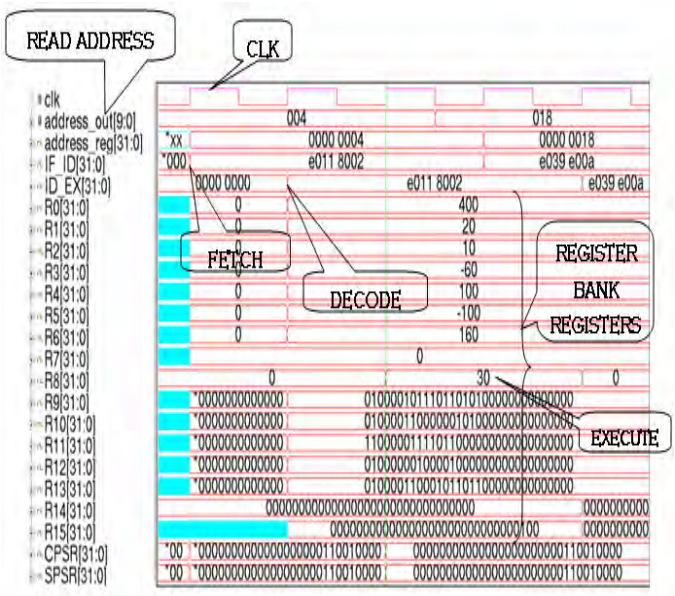

Figure 4: Simulation results of the designed RISC processor.

The Figure 4 shows the simulation results obtained after all pipeline stages have been integrated for the designed RISC processor operating at 50 MHz clock. Initially in the first clock cycle it is in reset condition (rst=1). Then when the reset is low, read is high and write is low and the address is given for instruction to be fetched from memory. In the next positive clock edge (20 ns after reset condition), the instruction will be fetched and stored in the IF_ID register. Then in the next clock cycle (at 40 ns from initial clock cycle) i.e. decode stage, the instruction is decoded, opr is updated, registers are read and the next instruction has been fetched. The current instruction is executed and the results written into the register in the next clock cycle (execute stage) and the next instruction reaches decode stage. The functionality has been verified by carrying out simulations for different test cases.

4. Functional Verification of 32-bit Enhanced RISC Processor core

The purpose of functional verification is to ensure that the design works as intended. Functional verification can be divided into categories

- Simulation with test benches manually derived by designers (Direct/Linear)
- Simulation with random patterns automatically generated by constraints solver (Random and Constraint Random)

4.1 *Direct/Linear testbench simulation*

The tests, which called the test bench procedures in sequence to apply manually selected input stimuli to the DUT and check the results, were directed towards specific features of the design. So this type of stimuli generation checked only specific scenarios and left the other scenarios unchecked. As the design becomes complex this process becomes tedious.

4.2 *Random testbench simulation.*

Here the input stimuli were automatically generated. By writing a single test and running it multiple times with different seeds, an engineer could, in effect, use the environment to create multiple tests. Although the test writing effort in this environment was greatly reduced, the additional work to maintain the generator not to provide redundant test was difficult. This made random substantial to handle complex design.

4.3 *Constraint random testbench simulation*

Here stimuli is generated based on the divided range, were the tests are tightened to avoid redundant test. This type of test is highly productive since it covers the maximum verification space. Directed test is highly constraint random test.

5. Results and Discussion

The following tables show the result of three types of functional verification carried out.

Table1: Linear/Direct coverage report

LINEAR/DIRECT COVERAGE REPORT				
DUT	SCORE	LINE	TOGGLE	BRANCH
RISC PROCESSOR	59.93%	100%	19.89%	
FETCH	61.60%	98.6%	14.72%	71.43%
DECODE	72.46%	80.8%	60.32%	76.19%
REGISTER BANK	46.05%	72.3%	15.81%	50.00%
ALU	70.05%	96.3%	57.45%	56.86%
BOOTHS MULTIPLIER	71.18%	92.0%	71.55%	50.00%
CARRY LOOKAHEAD SUB/ADD	93.35%	100%	86.68%	
BARELL SHIFTER	80.44%	76.4%	98.10%	86.67%

Table 2: Random coverage report

RANDOM COVERAGE REPORT				
DUT	SCORE	LINE	TOGGLE	BRANCH
RISC PROCESSOR	60.06%	100%	25.12%	
FETCH	71.31%	100%	13.94%	100%
DECODE	21.94%	45.59%	1.19%	19.05%
REGISTER BANK	59.90%	89.36%	15.35%	75.00%
ALU	28.69%	14.21%	64.41%	7.45%
BOOTHS MULTIPLIER	50.09%	100%	0.28%	50.00%
CARRY LOOKAHEAD SUB/ADD	79.75%	100%	59.51%	
BARELL SHIFTER	96.30%	100%	100%	88.89%

Table 3: Constraint Random coverage report

CONSTRAINT RANDOM COVERAGE REPORT				
DUT	SCORE	LINE	TOGGLE	BRANCH
RISC PROCESSOR	64.98%	100%	28.75%	
FETCH	59.81%	98.21%	9.79%	71.43%
DECODE	53.02%	55.88%	69.84%	33.33%
REGISTER BANK	64.77%	89.36%	29.04%	75%
ALU	87.81%	31.58%	65.88%	15.96%
BOOTHS MULTIPLIER	70.58%	100%	61.74%	50.00%
CARRY LOOKAHEAD SUB/ADD	96.77%	100%	93.54%	
BARELL SHIFTER	100%	100%	100%	100%

From the results above, it is clear that constraint random method proves better way compared to other methods.

Conclusion

The 32-bit RISC processor was modeled using Verilog HDL and simulated using VCS. Functional Verification is carried out with three methods direct, random, constraint random. Constraint random proved to be better.

In future work assertions can be added in the design; a verification environment can be developed for the design.

References

[1] Advanced RISC Machines Ltd, ARM7DI Data Sheet, December 1994
[2] Hyun Woo Cho, Ahn Woo Lee, Hua Jun Chi, Seung Won Song, Gyeong Su Gwon and Ju Sung Park,An ARM7 processor with the Modified Multiplier and the IEEE IFOST, pp 68-71,October 2006.
[3] Matthew R. Pillmeier, Michael J. Schulte and E. George Walters III, Design alternatives for barrel shifters, Proceedings of the Society of Photographic Instrumentation Engineers, vol.4791, and pp 436- 447, December 2002.
[4] www.testbench.in