International Journal on Future Revolution in Computer Science & Communication Engineering
Volume: 3 Issue: 12
ISSN: 2454-4248
220 – 225

Performance Enhancement of Cisc Microcontroller

Mr. K. Sai Krishna[1]
Department of Electronics and Communication Engineering
Anurag Group of Institutions, Hyderabad, Telangana, India
saikrishnaece@cvsr.ac.in

Mr. G. Sreenivasa Raju[2]
Department of Electronics and Communication Engineering
Anurag Group of Institutions, Hyderabad, Telangana, India
srinivasrajuece@cvsr.ac.in

Abstract- Increase in the speed of the system always demands for a major alteration on the existing system, which result in overall cost of the implementation of a system. Generally, CISC controllers are used for control operations, which have large number of instruction sets and take a large amount for processing due to its multiple sizes. For very high speed of controlling these controllers may fail to operate properly. The alternate solution is the RISC controllers, which are considerably faster than the normal CISC controllers. But these controllers have got various limitations as less instruction operations, complex register operation, costlier than the CISC controller etc. The only solution to this problem is the enhancement to the operational speed of a CISC controller, by enhancing the overall controller operation. Additionally, today's controller doesn't support the floating-point operation for signal processing. The enhancement of existing CISC controller by pipelining the overall operational flow of a CISC microcontroller and it includes the enhancement of UART. This research work is to be implemented using VHDL language and simulated using Active-HDL tool for functional verification.

Keywords *– CISC controller, RISC controller, VHDL.*

******

I. INTRODUCTION

1.1 The floating point unit (FPU) implemented during this project, is a 32-bit processing unit which allows arithmetic operations on floating point numbers. The FPU complies fully with the IEEE 754 Standard.

The FPU supports the following arithmetic operations:

1. Add 2. Subtract 3. Multiply 4. Divide

For each operation the following rounding modes are supported:

1. Round to nearest even 2. Round to zero 3. Round up 4. Round down

The FPU was written in VHDL with top priority to be able to run at approximately 100-MHz and at the same time as small as possible. Meeting both goals at the same time was very difficult and tradeoffs were made. In the following sections I will explain the theory behind the FPU core and describe its implementation on hardware.

The FPU core has the following features.

- Implements Single Precision (32-bit).
- Implements Floating point addition, subtraction and multiplication
- Implements all four rounding modes, round to nearest, round towards +inf, round towards -inf and round to zero.

1.2. Floating-point numbers

The floating-point representation is one way to represent real numbers. A floating-point number *n* is represented with an exponent *e* and a mantissa *m*, so that:

$n = b^e \times m$, …where b is the base number (also called radix)

So for example, if we choose the number n=17 and the base b=10, the floating-point representation of 17 would be: $17 = 10^1 \times 1.7$

Another way to represent real numbers is to use fixed-point number representation. A fixed-point number with 4 digits after the decimal point could be used to represent numbers such as: 1.0001, 12.1019, 34.0000, etc. Another way to represent real numbers is to use fixed-point number representation. A fixed-point number with 4 digits after the decimal point could be used to represent numbers such as: 1.0001, 12.1019, 34.0000, etc.

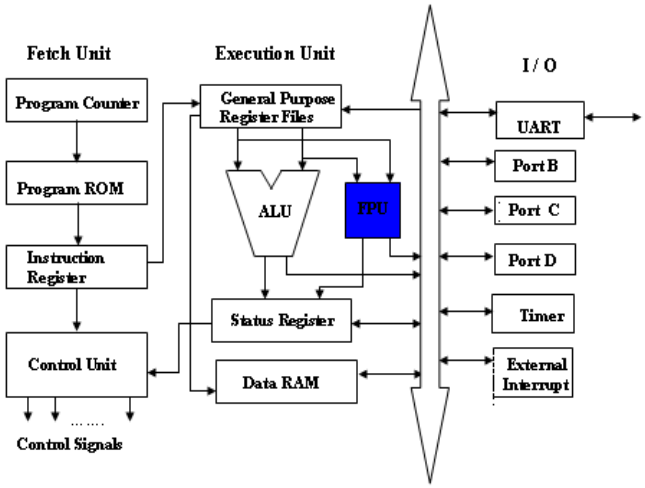

Fig 1. Enhanced CISC controller with FPU

Both representations are used depending on the situation. For the implementation on hardware, the base-2 exponents are used, since digital systems work with binary numbers. Using base-2 arithmetic brings problems with it, so for example fractional powers of 10 like 0.1 or 0.01 cannot exactly be represented with the floating-point format, while with fixed-point format, the decimal point can be thought away (provided the value is within the range) giving an exact representation. Fixed-point arithmetic, which is faster than floating-point arithmetic, can then be used. This is one

 ISSN: 2454-4248
220 – 225

of the reasons why fixed-point representations are used for financial and commercial applications.

The floating-point format can represent a wide range of scale without losing precision, while the fixed-point format has a fixed window of representation. So for example in a 32-bit floating-point representation, numbers from 3.4 x 10^{38} to 1.4 x 10^{-45} can be represented with ease, which is one of the reasons why floating-point representation is the most common solution. Floating-point representations also include special values like infinity, Not-a-Number (e.g. result of square root of a negative number).

II. ALGORITHM

Enhancement of CISC microcontroller shown in figure 10. Here there are some extra features added to the general CISC microcontroller, discussed in chapter 5. They are:

1. General Purpose Register file.
2. FPU (Floating-Point Unit).
3. UART (Universal Asynchronous Receiver Transmitter).

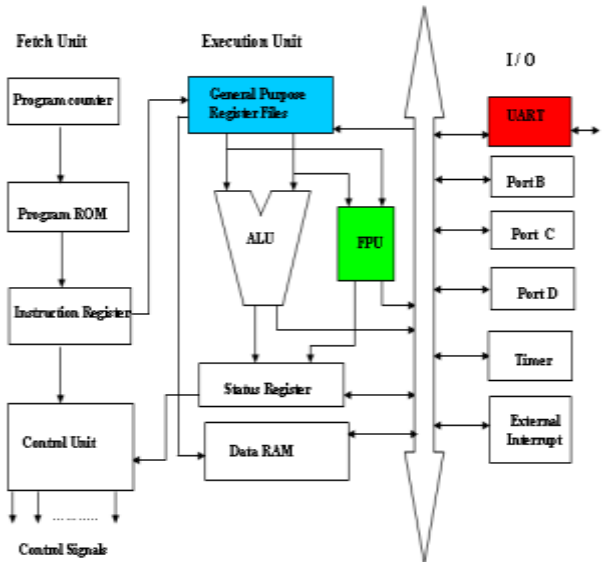

Fig 2. Enhanced CISC Microcontroller

2.1 General Purpose Register File

The structure of General Purpose Register File has 16 general-purpose registers. They are numbered from R16 to R31. R31 instead of R0 to R15 due to 2 reason. Firstly, immediate instructions like LDI can only address the upper register. Secondly, the indirect Z-pointer shares the same register as R30. The address bus connects the register file and data RAM together. R30 can be used as either a general register or the Z-pointer (ZP) to address the data RAM. The working function of GPRF is discussed in chapter 6. The instruction register is used store the fetched instructions and connected to the GPRF. The instructions from the register file passed to the ALU for evaluate the integer values and/or the FPU for evaluate the floating-point values.

2.2 Floating Point Unit

The floating-point unit (FPU), is a 32-bit processing unit that allows arithmetic operations on floating point numbers. An ALU allowed evaluating only integer arithmetic operations. Here decimal data is truncated and outputs the integer data only. In this situation, the FPU is used to evaluate the exponential operations also. The working function of the FPU is discussed in chapter 9. The input code (instructions) are taken from the GPRF and evaluated results are stored into the status register.

2.3 UART

UART stands for "universal asynchronous receiver transmitter". The basic principle of UART is described in chapter8. Popular serial communication devices in computers to interfacing low speed peripheral devices such as the keyboard the mouse modems etc. It is an integrated serial port means that it may very easily read and write values to the serial port. If it were not for the integrated serial port, writing a byte to a serial line would be a rather tedious process-requiring turning on and of one of the I/O lines in rapid succession to properly "clock out" each individual bit, including start bits, stop bits, and parity bits. Simple communitarian protocol, which can be implemented on hard or software. Asynchronous communications operate on independent clocks. Available as chip set software module soft macro and Hard Macro.

2.3.1 input considerations:

The Design implemented consider a swapping operation fed onto the Program memory as a test program as given below:

```
mov a, #data (00)
mov direct (0), #data (64)
mov direct (64), #data      (ff)
mov direct (01), #data      (65)
mov direct (65), #data      (88)
mov a,@r0
mov direct(48),a
mov a,@r1
mov @r0,a
mov a, direct(48)
mov @r1, a
 NOP
```

The program clears the content of accumulator by loading Accumulator with 00. The memory location (00) of the RAM is loaded with data 64 which indicate the address location of the first operand. The first operand is then loaded on to the location specified i.e. Location 64 is loaded with FF. The memory location (01) of the RAM is loaded with data 65 which indicate the address location of the second operand. The second operand 88 gets loaded onto the location 65. The two operand 88 and FF are then swapped down via the accumulator. The swapping operation is performed by indirect addressing mode where the content of indirectly addressed location given by register 0 is stored

 ISSN: 2454-4248
220 – 225

onto the accumulator i.e. value of location 64 (88) is stored to the accumulator. This accumulator data is then passed to a new memory location for temporary hold of data. This makes the accumulator free to carry the data of location 65.

The accumulator data is then passed down to location 64, which is indirectly pointed by register 0. Finally the data stored in temporary location is stored onto location 65 via accumulator. This performs the swapping of the two-memory location via accumulator.

III. Simulation Results

3.1 UART Results

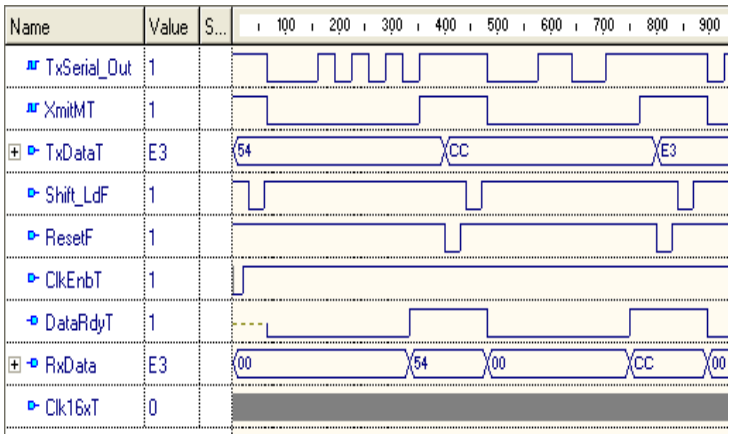

Fig 3.1 Simulation results of UART

This is a simulation result of UART module. Here TxDataT, Shift_ldf, clk, ResetF, clkEnbT, clk16xT are inputs and DataRdyT, RxData are outputs.

In Transmitter section data transmits to the parallel to serial. In Receiver section serial data converts into the parallel data. Here I have used Finite State Machine. In FSM I have used S0 to S10. Here state 0 is idle state when transmitter is waiting for the data to be high on input lines.

Transmitter Section:

Receiver Section:

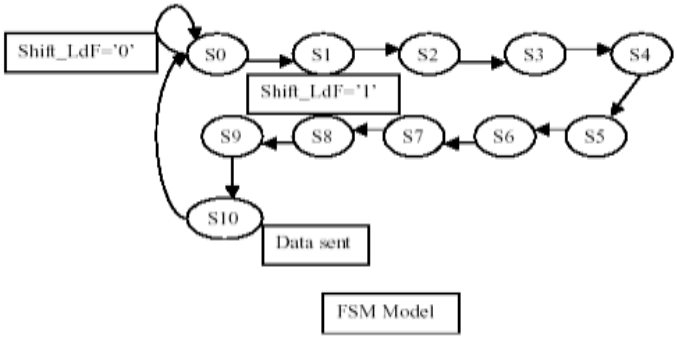

Fig 3.2 FSM model of Transmitter Section

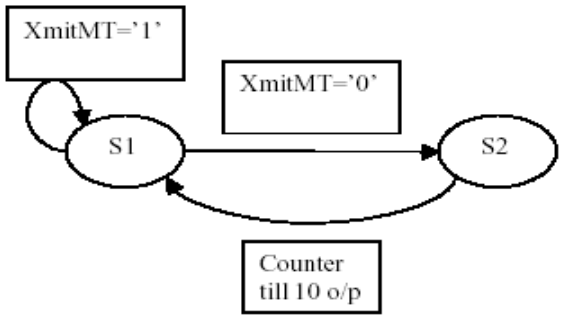

Fig 3.3 FSM model of Receiver Section

Here input is TxDataT
ClkEnbT, Clk16xT, shift_LdF are control signals
RxData is output of the Receiver section.
Example:
Input TxDataT=58
if shift_Ldf = 1, ResetF=1, CLKEnbT=1
then RxData=58

Parllel to Serial Converter

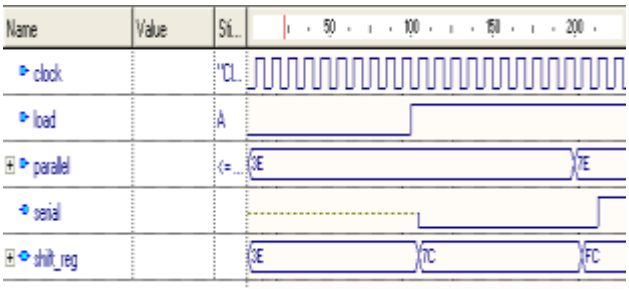

Fig 3.4 Simulation results of parallel to serial converter

This is a simulation result of parallel to serial converter. Here input is parallel, load is used for control signal, shift_reg is inout and serial is an output.

if load='0' then shift_reg <= parallel
elsif clock='1' then serial <= shift_reg(7)
shift_reg(7 downto 1) <= parallel(6 downto 0)

Example:
input: parallel data= 00111110
ouput: if load=0 then serial=0,shift_reg=00111110
else serial=0,shift_reg=01111100

Serial to Parallel Converter

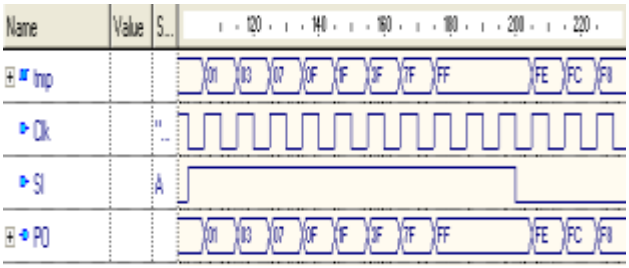

Fig 3.5 Simulation results of serial to parallel converter

 ISSN: 2454-4248
220 – 225

This is a simulation result of serial to parallel converter. Here input is si and po is output.

```
if (Clk'event and Clk='1') then
        tmp <= tmp(6 downto 0) & SI
            PO <= tmp;
```

Example: input: si = 1: ouput: po = 00000001,00000011,00000111,00001111,00011111,00111111,1111111,

11111111

General-purpose register result

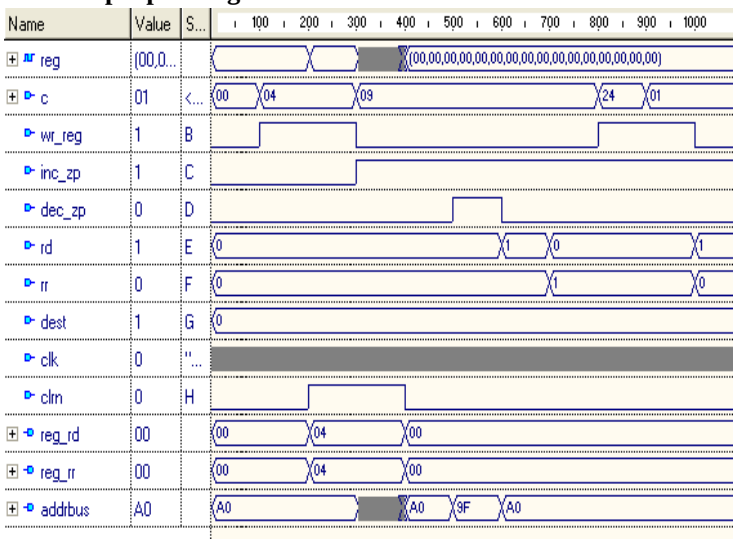

Fig 3.6 Simulation results of general-purpose register.

This is a simulation result of general-purpose register file. Here inputs are c, wr_reg, inc_zp, dec_zp, rd, rr, dest, clk, clrn and reg_rd, reg_rr, addrbus are outputs.

Example: if wr_reg=1,clrn=1 then reg_rd<=c
else reg_rd<='0'

Floating Point Unit Result

This is a simulation result of addition & subtraction for Floating Point Unit. Here inputs are opa, opb, and sum, co are outputs.

Example:

Addition & Subtraction operation:

Inputs: opa=22 (in hex=16) opb=18 (in hex=12)

if add ='1' then it takes addition operation otherwise it will take subtraction

Output: sum= 40 (in hex=28) sub= 4 (in hex=4)

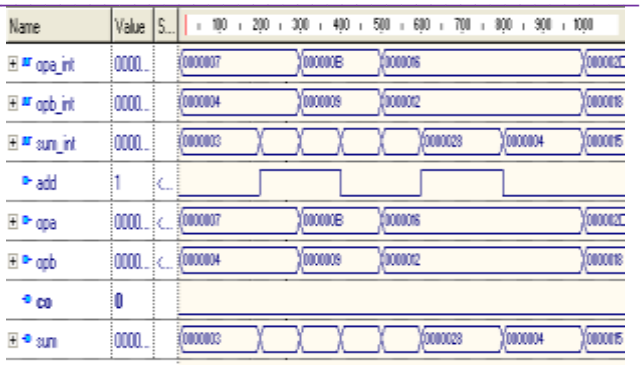

Fig 3.7 Simulation results of addition & subtraction for FPU

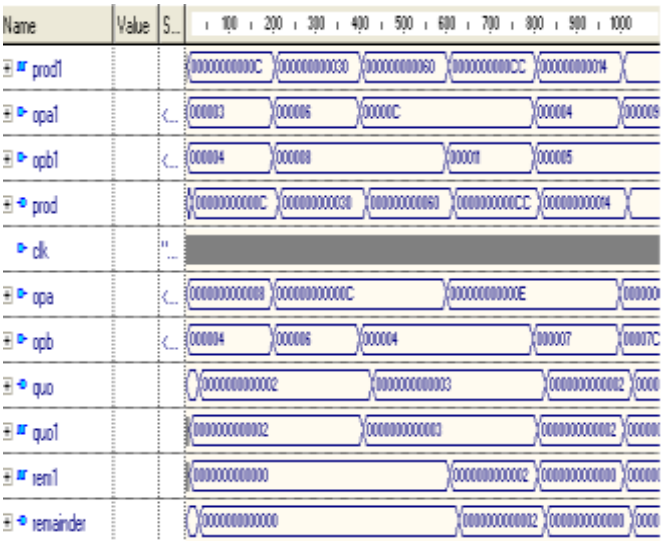

Fig 3.8 Simulation results of multiplication and division for FPU

This is a simulation result of multiplication and division for Floating Point Unit. Here inputs are opa1, opb1,opa,opb and prod, quo, remainder are outputs.

Example:

Multiplication operation:

Inputs: opa1=6 (in hex=6);

opb1= 8 (in hex=8) ;

Output: prod= 48 (in hex=30)

Division operation:

Inputs: opa = 12 (in hex=c) opb= 4 (in hex= 4) ;

Output: quo= 3 (in hex=3) Remainder =0

ISSN: 2454-4248
220 – 225

Enhanced Microcontroller Results

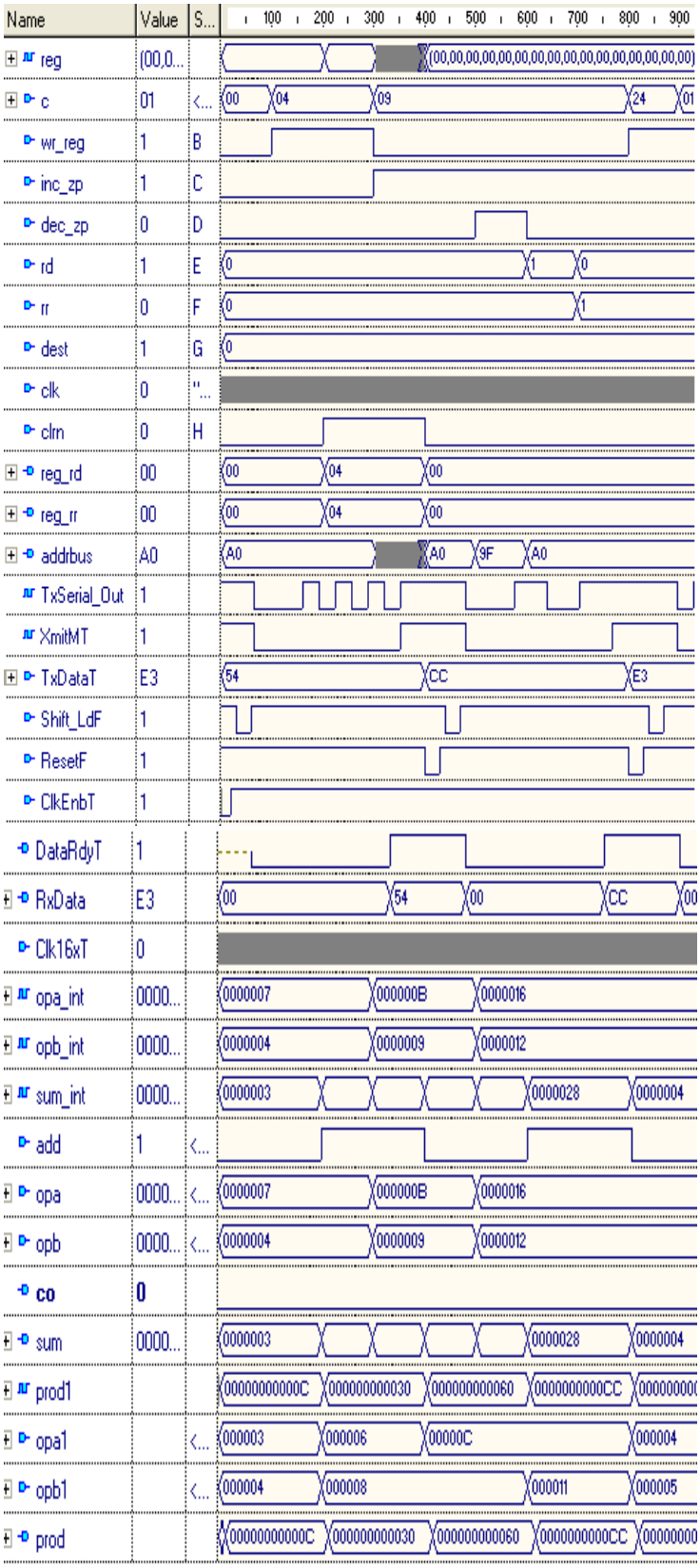

Fig 3.9. Simulation results of Enhanced Microcontroller

This is a simulation result of the Top Level module microcontroller. Here inputs are c, wr_reg, inc_zp, dec_zp, rd, rr, dest, clk, clrn, TxDataT, Shift_ldf, clk, ResetF, clkEnbT, clk16xT, opa, opb, opa1, opb1 are inputs and reg_rd, reg_rr, addrbus, DataRdyT, RxData, sum co, prod are outputs.

Example:

if wr_reg=1,clrn=1 then reg_rd<=c
else reg_rd<='0'
Input TxDataT=cc
if shift_Ldf = 1, ResetF=1, CLKEnbT=1
then RxData=cc
Addition & Subtraction operation:
Inputs:
opa=22 (in hex=16)
opb=18 (in hex=12)
if add ='1' then it takes addition operation otherwise it will take subtraction
Output:
sum=40 (in hex=28)
sub=4 (in hex=4)
Multiplication Operation:
Inputs:
opa1=6 (in hex=6)
opb1=8 (in hex=8)
Output:
prod=48 (in hex=30)

IV Conclusion

The goal of the project is to design a Performance Enhancement of CISC Microcontroller that is capable of achieving high speed, low power consumption and reduced cost. Today's technology requires high speed for faster computations; Microcontrollers designed using CISC architecture has high speed than the normal microcontroller architectures.

In this project a low power, high-speed microcontroller was designed based on the normal microcontroller architecture. By executing powerful instructions in a single clock cycle, the designed CISC microcontroller achieves very high speed of operation. In this project were also increased the number of blocks in the microcontroller, which enhances the performance of CISC architecture, and the instructions are executed directly through the registers by reducing the storage time and memory space usage.

The *Salient features* of the *new approach blocks* added to the existing microcontroller are,

- General Purpose Register
- UART (Universal Asynchronous Receiver Transmitter)
- FPU (Floating Point Unit)
- The key difference between normal microcontroller and *enhanced microcontroller* and the improvements are

 ISSN: 2454-4248
220 – 225

- Enhancement to the operational speed of a CISC controller, by enhancing the overall controller operation by adding GPR, UART &FPU.
- Enhancement of existing Microcontroller by pipelining the overall operational flow of a CISC microcontroller.
- By adding FPU the compatibility and flexibility is increased for floating point operations.
- More number of instructions can be developed for this enhanced microcontroller.
- Efficiency of the microcontroller increased.
- Simple, and therefore manageable, with less number of instructions.
- Using this effective design can do the more complex accelerated computations.

This research was design flow allows the designer to freely optimize each unit of the design. Users who need to do real-time data acquisition would benefit by the Combination of the high-speed performance capabilities available today with the ability to transfer data. The HDL code for the proposed system is completely developed in VHDL language and logically verified on Active-HDL tool.

V. Future Scenario

At first, the microcontroller does not contain any data RAM. So the stack is implemented using hardware just like controller and is only 4-level deep. Future works should have the stack implemented in the data RAM using a stack pointer. This will save up some area and more important, the stack will be able to keep a few times more entry than the original hardware stack.

There is only one indirect pointer, the Z-pointer in this design. If memory access is frequent, more indirect pointers would make the job easier. Future works should also include the X-pointer and Y-pointer.

There are many more extra features available in the CISC microcontroller family, such as the UART serial interface, SPI serial interface, the 16-bit timer (with output compare and input capture), etc. The work from this project should be used as a platform to implement these features in.

References

[1] Daniel Tabak, RISC Systems, Research Studies Press Ltd.: Taunton, Somerset, England TA1 1HD, 1990

[2] M.Morris Mano, *Computer System Architecture*, Prentice Hall inc.: Englewood Cliffs, New Jersey 07632, 1993.

[3] *The 8051 Microcontroller and Embedded systems* by Muhammad Ali Mazadi, Janice Gillispie Mazadi; Pearson Edition

[4] Randy H. Katz, *Contemporary Logic Design*, The Benjamin/Cummings Publishing Company, Inc.: Redwood City, California 94065, 1994.

[5] Douglas L. Perry, VHDL, McGraw-Hill Companies, Inc.: Singapore, 1999.

[6] Jan Gray, *Building a RISC system in an FPGA*: Part 1,2 & 3, Circuit Cellar Magazine (http://www.circuitcellar.com), 2000.

[7] J.Bhasaker, A VHDL Primer, Revised Edition, Englewood Cliffs NJ: Prentice Hall 1995

[8] K. Hoganson, "*Mapping Parallel Application Communication Topology to Rhombic Overlapping-Cluster Multiprocessors*", accepted for publication, to appear in The Journal of Supercomputing, To appear 8/2000, Vol. 17, No. 1.

[9] Navabi, Zainalabedin, *VHDL Analysis and Modeling of digital Systems* New York; McGrawHill 1993

Test books

[10] Unified Parallel System Modeling project, Directed Study, Summer-Fall 2000

[11] IEEE computer society: IEEE Standard 754 for Binary Floating-Point Arithmetic, 1985.

[12] David Goldberg: What Every Computer Scientist Should Know About Floating-Point Arithmetic, 1991.

[13] W. Kahan: IEEE Standard 754 for Binary Floating-Point Arithmetic, 1996.