IMPLEMENTATION OF SOC CORE FOR IOT ENGINE

P.Jennifer[1] and S.Ramasamy[2]

[1]ME. VLSI Design, RMK Engineering College, Anna University
[2]Professor, Department of ECE, RMK Engineering College, Anna University

ABSTRACT

Implementation of microprocessor core on a programmable device has been mostly sought by researchers due to its scalability and hardware reconfigurability. The proposed minimum version of 32-bit processor core is developed especially for arithmetic operations of fixed point numbers, branch and logical functions. This paper presents the complete design of a microprocessor core in synthesizable Verilog. It defines an instruction set architecture suitable to be used for Internet of Things (IoT) application. This works as co-processor for IoT engine. The System on Chip (SoC) core has been synthesised and simulated using Synopsys Design Compiler and VCS. The SoC core is designed for 14 classic arithmetic and logical instructions suitable for IoT applications. However, the design can be expandable to 64 and 128 bits. This optimized processor core can be pipelined up to 5 stages and can be used for high speed applications. Architectural approach for low power and high performance are described and the area occupied by the entire core is 66562.3µm². The total power consumed by the design is 1.72 mW at 126MHz.

KEYWORDS

instruction set architecture, Internet of Things(IOT), microprocessors, RISC, System on Chip(SOC)

1. INTRODUCTION

Gartner's 2015 predictions focus on the impacts of the evolution of digital business and pays closest attention to the Internet of Things (IoT), since it has introduced new concepts for identity management (every device interacting with users has an identity) and users and devices can have complex, yet defined, relationships. Further, as the smart wearables market continues to grow and evolve, Gartner predicts that by 2017, 30% of smart wearables will be completely unobtrusive to the eye[1]. The devices and objects that once were autonomous are becoming more connected to each other, to the Internet, or, more commonly, to both. From building and home automation to wearables, the IoT touches every facet of our lives. Every chip and OEM device manufacturer now building components and solutions for the IoT, especially wearable and battery-powered devices, faces a performance and power paradox challenge that is driving the need for a new type of low-power processor. With advent of IoT era, processor configurability is very important to achieving the right balance of performance, power, and area. The ability to easily configure the processor by selecting, minimizing, adjusting, or reducing features to tailor its performance for specific application requirements is essential. For example, selecting and optimizing the number of registers, the type of multiplier, and the number of interrupts and levels enables the core gate count and area to be modified to suit the application performance levels without wasting area and power[2]. Extensibility is also a key for designing a processor that supports next-generation IoT applications. It enables designers to add user-defined hardware like arithmetic logic unit (ALU) instructions, condition codes, core and auxiliary registers, and external interface signals to the processor core. By adding user-defined extensions to the processor, a new level of CPU performance efficiency can be achieved.

1.1. ARC EM4 Processor

The Synopsys DesignWare ARC EM4 Processor[3], with a configurable and extensible 32-bit RISC microarchitecture, was developed to address the power/performance paradox in IoT and other applications. It can be optimized with configurable hardware extensions for a sensor application, for instance, specifically aimed at reducing power or energy consumption. Eventhough there are several features, the added processor extensions in the configuration result in a small increase in area (4.5%) and a small increase in instantaneous dynamic power of the core (7.2%).

1.2. ARM Cortex-M0 Processor

ARM remains focused on driving a unified and simplified connected world involving truly ubiquitous and intelligent IoT systems. The ARM Cortex-M0+ processor[4] is the most sought processor in cortex family as far IoT is concerned. The processor will be used in microcontrollers for communication, management and maintenance across a multitude of wirelessly connected devices. The processor has been redesigned from the ground up to add a number of features including single-cycle IO to speed access to GPIO and peripherals, improved debug and trace capability and a 2-stage pipeline to reduce the number of cycles per instruction (CPI) and improve flash accesses, further reducing power consumption. Though, ARM processors are highly efficient and accepted by all the researchers, its instruction sets and the architecture was not fully available for research community. In this work we attempt to implement SoC core suitable for IoT applications with the inputs from [5] [6] [4].

In this paper, we present the high performance RISC processor[5] which is modified for 32-bits with special quality of instruction format. It is a fully synthesizable core and it is designed for 14 arithmetic and logical instructions. It gets instructions on a regular basis using dedicated buses to its program memory, executes all its native instructions and exchanges data with several devices using other buses.

The organization of the paper is as follows. Section 2 explains the architecture of the design of 32-bit RISC processor (SoC) core which includes the instruction set, instruction format and conditional codes. Section 3 presents the description of Logic blocks and the design of each module of the processor. Section 4 includes the implementation, Simulation results and Schematic view of RISC processor. The paper is concluded in Section 5 with References.

2. CORE ARCHITECTURE

The processor contains 32 general purpose registers with the capability of 32 bits which is used for all operations. To make decisions it uses three flags, „Zero", „Carry" and „Negative". With them it can evaluate up to four conditions.

2.1 Instruction Set

Operations are considered to be 32-bit operations with 32-bit results, except for the multiply, which produces a 64-bit result. For each operation, the zero flag is set if the result contains all zeros. The negative flag mirrors the most significant bit in the result. In the two's complement number system, this is equivalent to telling whether the value is positive or negative. Mathematical operations are performed exactly the same as for unsigned values, and the programmer can even choose to ignore the negative flag and work with the unsigned values. In the event that a subtraction causes a borrow, the carry bit will be set, and this can be used to determine if a larger value was subtracted from a smaller value. Instructions are shown in the Table I.

2.1.1. NOP

No operation. After reading this instruction, the processor continues immediately to the next instruction without any extra clock cycles.

2.1.2. ADD

It adds a value to a register. The source value can be another register, a memory location, or a 32-bit immediate. The carry flag is set if a one was carried out of the highest bit. The negative flag is set if the highest bit is a 1, and the zero flag is set if the result is zero.

Table 1. Instruction Set with Opcode and Operands

Instruction	**Opcode**	**Operands**
NOP	0x00	-
ADD	0x01	reg, reg/imm
SUBTRACT	0x02	reg, reg/imm
MULTIPLY	0x03	reg, reg/imm
Logical AND	0x04	reg, reg/imm
Logical OR	0x05	reg, reg/imm
Logical Shift Right	0x06	Reg
Logical Shift Left	0x07	Reg
Complement bits	0x08	Reg
LOAD	0x10	reg, imm/address
STORE	0x11	address, reg
MOVE	0x12	address, address
JUMP	0x0F	address,condition
HALT	0x1F	-

2.1.3. SUB

It subtracts a value from a register. The source value can be another register, a memory location, or a 32-bit immediate. The carry flag is set if a one was borrowed. The negative flag is set if the highest bit is a 1, and the zero flag is set if the result is zero.

2.1.4. MUL

It multiplies a 32-bit register by a 32-bit value, producing a 64-bit result. The source value can be another register, a memory location, or an immediate. The result will be written to the destination register and it"s a 64-bit pair, i.e., multiplying register a by register c will replace both a and b with the result. Best practice is to load both operands into a single register pair, so that no registers with other data are accidentally overwritten.

2.1.5. AND

It performs a bitwise AND between a register and a value. The source value can be another register, a memory location, or a 32-bit immediate. The negative flag is set if the highest bit is a 1, and the zero flag is set if the result is zero.

2.1.6. OR

It performs a bitwise OR between a register and a value. The source value can be another register, a memory location, or a 32-bit immediate. The negative flag is set if the highest bit is a 1, and the zero flag is set if the result is zero.

2.1.7. LSR

It performs a logical right-shift on a register and stores the result in the same register. All 32-bits are shifted, and a zero is shifted into the top bit position. The carry flag is set if a 1 was shifted out. The negative flag is set if the highest bit is a 1, and the zero flag is set if the result is zero.

2.1.8. LSL

It performs a logical left-shift on a register and stores the result in the same register. All 32-bits are shifted, and a zero is shifted into the lowest bit position. The carry flag is set if a 1 was shifted out. The negative flag is set if the highest bit is a 1, and the zero flag is set if the result is zero.

2.1.9. ASR

It performs an arithmetic right-shift on a register. Only the lower 31 bits are shifted; a copy of the sign bit is shifted into the bit position below the sign bit. The carry flag is set if a 1 was shifted out. The negative flag is set if the highest bit is a 1, and the zero flag is set if the result is zero.

2.1.10. ASL

It performs an arithmetic left-shift on a register. Only the lower 31 bits are shifted; a zero is shifted into the lowest bit position. The carry flag is set if a 1 was shifted out. The negative flag is set if the highest bit is a 1, and the zero flag is set if the result is zero.

2.1.11. COMP

It complements all of the bits in a register and stores the result in the same register. The negative flag is set if the highest bit is a 1, and the zero flag is set if the result is zero.

2.1.12. NEG

It performs a 2's complement (binary negation) on a register and stores the result in the same register. The negative flag is set if the highest bit is a 1, and the zero flag is set if the result is zero.

2.1.13. LOAD

It loads a value into a register. The source value can be a memory location or an immediate. The source can also be another register, which is also handled internally with the LOAD opcode. The assembler uses the mnemonic "COPY" to refer to loading one register into another.

2.1.14. STORE

It stores a register to RAM. A memory address is required.

2.1.15. MOVE

It copies the contents of one memory address to another. Two addresses are required; the register contents are not affected.

2.1.16. JUMP

It is classified as conditional and unconditional jumps. When a conditional jump occurs, a condition is checked and if the condition is true, then the jump occurs. It relocates the current program counter to the memory location specified. After the jump instruction, the processor immediately continues executing instructions at the new memory location. An unconditional jump always occurs. There is no condition to check. The processor can jump always, regardless of ALU flags. This uses the assembler mnemonic "JMP". If the ALU carry bit was set in the last ALU operation, it uses the assembler mnemonic "JCAR". If the ALU zero bit was set in the last ALU operation, it uses the assembler mnemonic "JZERO". If the ALU negative bit was set in the last ALU operation, it uses the assembler mnemonic "JNEG"

2.1.17. HALT

It halts the processor‟s operation by putting the bus into High Impedance State. No more instructions are read and the processor stays in the HALT state until it is reset.

2.2. Requirements

The requirements for the design are 32-bit data bus, 32-bit address bus, 32 numbers of 32-bit general purpose registers which can be used in pairs as 16 numbers of 64-bit registers, 64-bit Instructions format, Instruction set, Jump condition codes. Jump Condition codes [6] are shown in the table below.

Table 2. Jump Condition Codes

Condition	**Bit designation**
Always	00
Carry	01
Zero	10
Negative	11

3. DESIGN OF MODULES

3.1. Generic 64-bit Register

A 64-bit register is used three times in the design: the jump register, the memory address register, and the instruction register. The same module is instantiated in all three of these cases. The module has a 32-bit input, which in all three instantiations comes from the data bus. Two signals, setHigh and setLow determine whether this input is stored in the top half or bottom half of the register.

3.2. Program Counter

The 64-bit program counter holds the address of the next instruction byte and is used to index ROM when fetching instructions. It increases the output count by one on the rising edge of the clock when the increment signal is high. When the set signal is asserted, it loads the value from the jump register through the input newCount.

3.3. General Purpose Register

There are 32 general purpose registers with the capability of 32 bits which is used for all operations. They are grouped into pairs, creating 16 registers of 64-bit wide which can receive the result of a multiply operation. Values are only stored on the rising edge of the clock, but the outputs are set via combinational logic. This means that the outputs are available immediately, so register-to-register copies and ALU operations from the registers take place immediately without having to wait an additional clock cycle for the data to become available.

3.4. Arithmetic Logic Unit (ALU)

The arithmetic logic unit implements all of the arithmetic operations specified: addition, subtraction, multiplication, logical AND and OR, left and right logical shifts, left and right arithmetic shifts, bitwise complement, and negation. It also contains a pass through instruction so that the ALU latch can be used as a temporary register for the MOVE operation. Output flags are set based on the results of the operation. The ALU consists entirely of combinational logic and operations are performed whenever the inputs change. The output is only 32 bits, except for the multiply, which is 64 bits. The zero flag is defined for every operation. If the result is all zeros, the flag is set. Likewise, the negative flag is set whenever the highest bit of the result is a 1, which indicates a negative number in the two's complement system. The behavior and meaning of the carry flag is dependent on the operation. For add and subtract, it indicates a carry out or borrow in; for shifts, it indicates the bit that was shifted out.

3.5. ALU Latch

The ALU latch grabs the result of the ALU operation, holds it, and then puts it on the data bus when the store signals are asserted. It also latches the flags, so that a jump operates based on the last time the result was grabbed. The ALU latch uses a simple sequential design. The alu_result and flags are stored on the rising edge of the clock if grab is high. Combinational logic is used to determine which half of the stored value is put out to the data bus. If neither store signal is high, the output is high-z. The flags_out output is always enabled**.**

3.6. Data Path

The datapath module combines the program counter, jump register, general-purpose registers, ALU, ALU latch, memory address register, and instruction register into a single unit connected by a data bus. The data bus is a bidirectional module port, so data can be brought in and out of the chip. A block diagram of datapath[5] is shown in Fig 1.

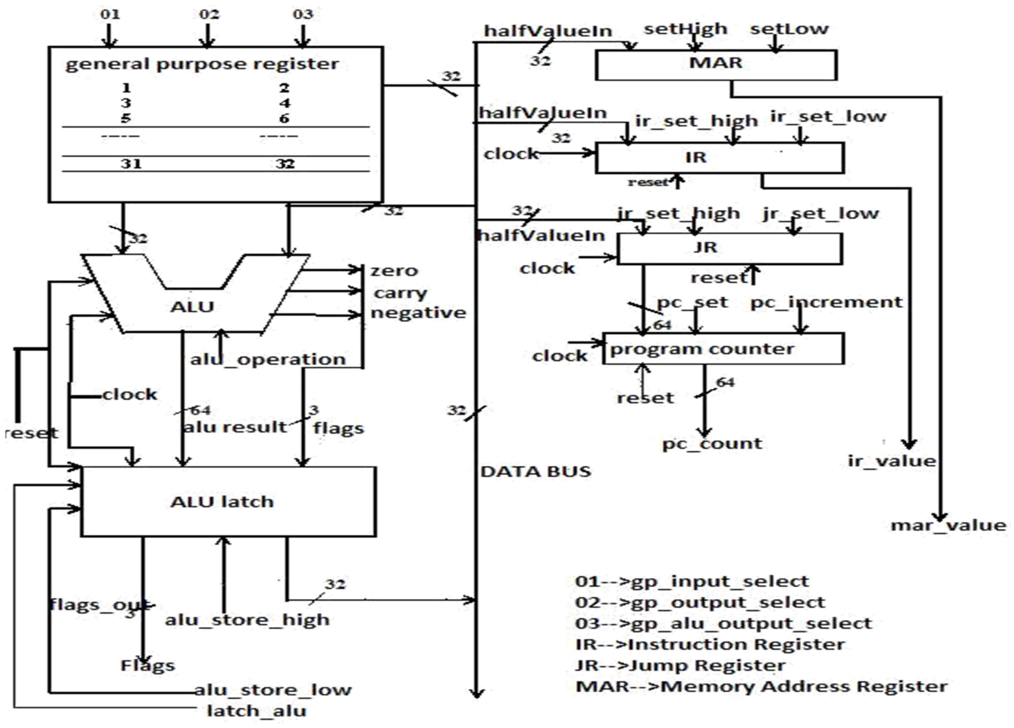


Fig 1. Block Diagram of Data Path

3.7. Control Modules

The control module consists of two separate modules: a state machine which reads the output of the instruction register and determines what to do on the next clock cycle, and a signal translation module which maps the control state into controls signals for all of the other modules.

3.7.1. State Machine

The states are defined as constants.

3.7.2. Control signal translation

Pure combinational logic with assign statements was used to produce the proper outputs. The general-purpose register addresses are taken directly from the opcode, except where the destination address is modified to produce a 64-bit store for the multiply operation. The ALU operation is taken directly from opcode bits [62:59]. The module also handles jump evaluation and execution. If the operation is a jump and the condition code is “always”, then the program counter set signal is asserted. If the operation is a jump, the condition code is carry, and the carry flag is set, then the signal is asserted. The same logic is used for the carry and zero jumps. No additional clock cycles are necessary for evaluating the jump.

3.8. Address Multiplexer

The address multiplexer switches the output address between the program counter and memory address register based on the state. If the processor is performing a load or store using a memory location, the MAR address is used; otherwise, the program counter is used.

3.9. Memory IO

The memory IO module performs memory mapping to locate the ROM and RAM in address space, and translates the read and write signals into ROM and RAM chip enable signals. Because the program counter's reset is at 0x0000, it needs to find its first instruction at that memory location. This is done by memory-mapping the ROM.

3.10. CPU

The CPU module[5] includes all of the other modules. It connects the control state machine to the datapath via the state translation module, connects the data path to the outside using the address multiplexer and memory IO module. The CPU module is shown in Fig 2.

4. TEST RESULTS AND DISCUSSION

The design is written using Verilog HDL‟s (Synopsys VCS) which easily allows the design to be simulated earlier in the design cycle in order to correct errors or experiment with different architectures. The simulation results have been verified using VCS and optimized using Synopsys Design Compiler. The verilog code is synthesized using faraday fsd0a_a_generic_core_tt1v25c CMOS technology. All the modules are verified separately. Result will give 32 bit outputs except for multiplication operation which produces 64 bit output. 64 bit register is instantiated 3 times in the design. halfValueIn is the 32 bit input given to the register. The general purpose register block contains 32 registers with the capability of 32 bits which is used for all operations. The register block has separate address inputs which are used to index the registers. alu_output_select address determines which address is sent directly to alu on the bus. Datapath combines all the previous modules. The data can be brought in and out of the chip. From the bidirectional data bus, values are loaded in to the registers. 14 operations are performed sequentially and the output is stored back in to registers. A part of the data path output is shown in Fig 3.

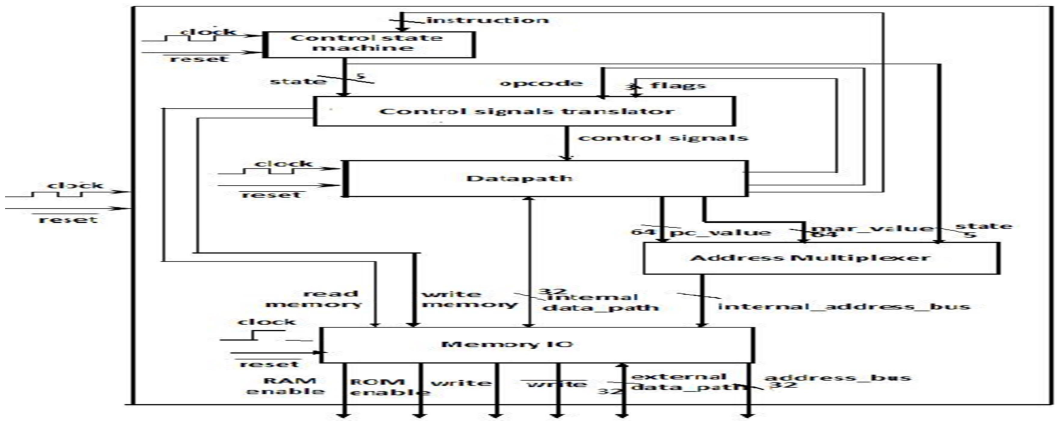


Fig 2. CPU Block Diagram

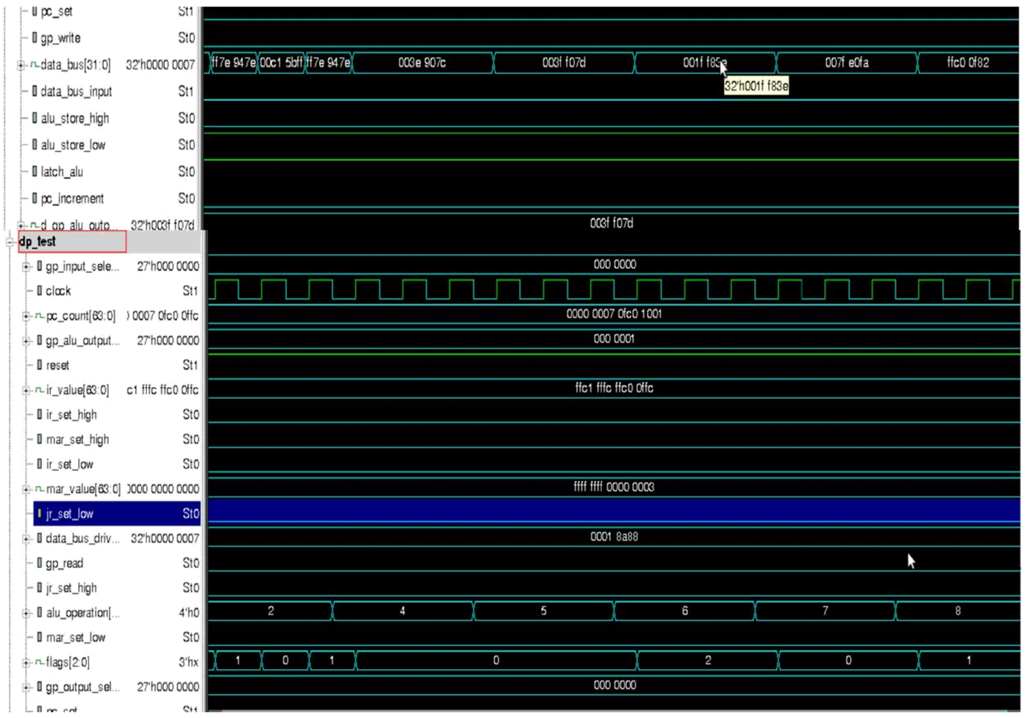


Fig 3. Datapath Output

The CPU module includes all the modules with control state machine and translator along with address multiplexer. External SRAM and ROM is used as a text file for testing purpose and the memory drives the CPU. The values passed are shown in the result below

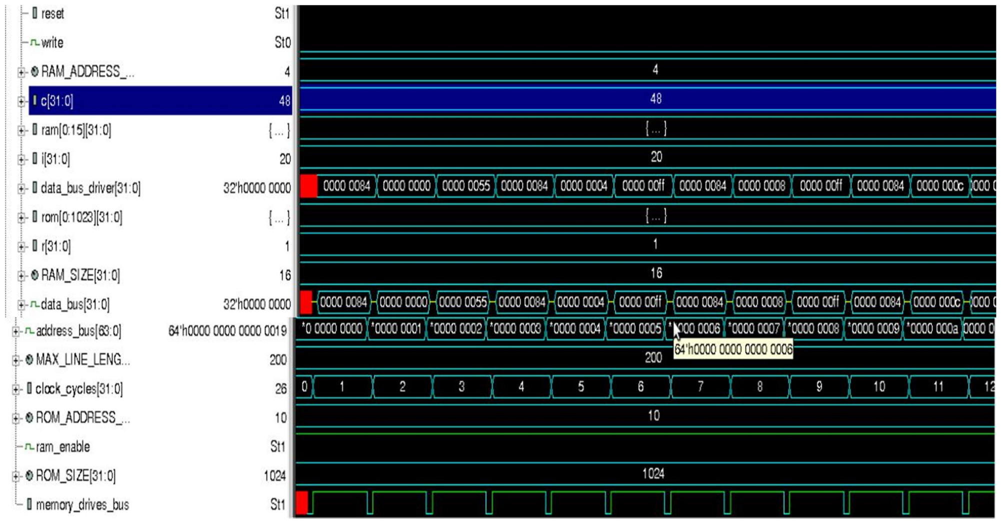
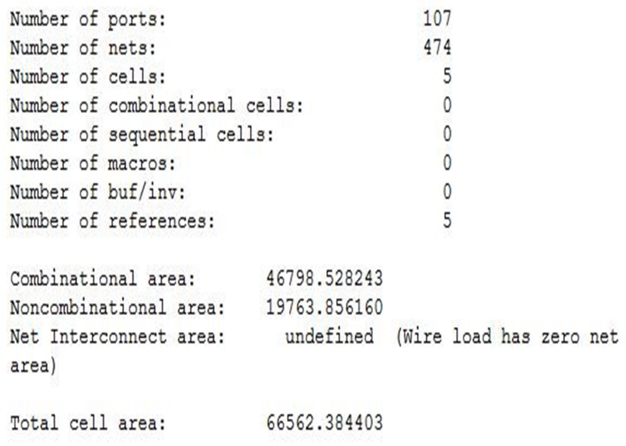


Fig 4. Simulation of CPU operation

RTL code is taken in to Design compiler. Area, timing and power reports are obtained using Design Compiler.

4.1. Area

The optimized area for the entire core is 66562.3 μm^2. Report is shown below.



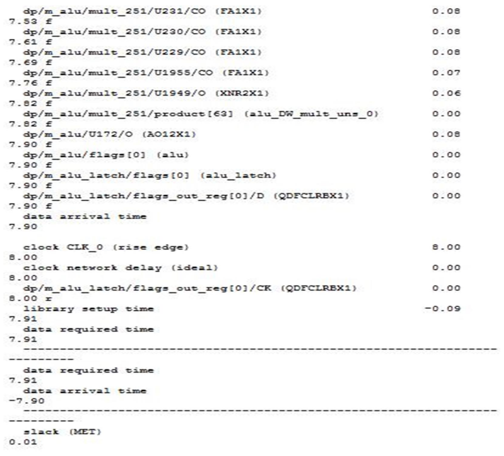
```
Number of ports:                      107
Number of nets:                       474
Number of cells:                        5
Number of combinational cells:          0
Number of sequential cells:             0
Number of macros:                       0
Number of buf/inv:                      0
Number of references:                   5

Combinational area:       46798.528243
Noncombinational area:    19763.856160
Net Interconnect area:      undefined  (Wire load has zero net
area)

Total cell area:          66562.384403
```

Fig 5. Simulation Result of Area Occupied

4.2. Timing

The timing report specifies an incremental delay and total path delay at each step. At the end, data arrival time is listed. The critical path delay is 7.90 ns, which shows that the core will work at 126 MHz, satisfies the requirement for IoT applications[2]. Data required time is the clock period which was tried to achieve,with time subtracted to account for the setup time the processor needs. Data required time is calculated and it is 7.91 ns.

Data required time and data arrival time is compared. The clock speed constraint has been met and there is a slack of 0.01 ns. A part of the timing report is shown in Figure 6.

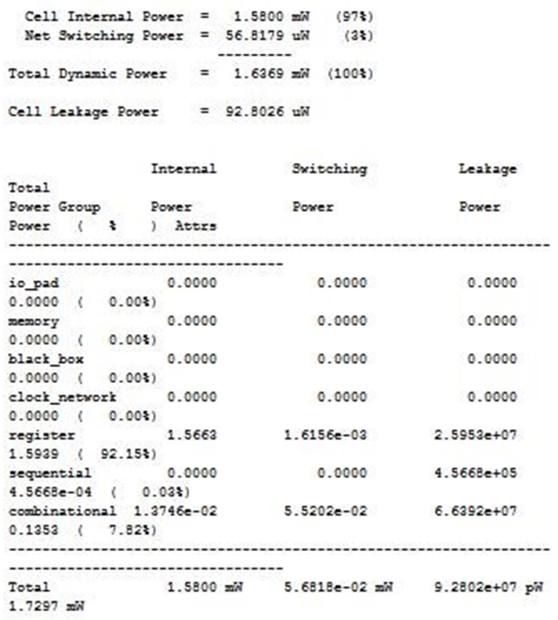


```
  dp/m_alu/mult_251/U231/CO (FA1X1)                       0.08
7.53 f
  dp/m_alu/mult_251/U230/CO (FA1X1)                       0.08
7.61 f
  dp/m_alu/mult_251/U229/CO (FA1X1)                       0.08
7.69 f
  dp/m_alu/mult_251/U1955/CO (FA1X1)                      0.07
7.76 f
  dp/m_alu/mult_251/U1949/O (XNR2X1)                      0.06
7.82 f
  dp/m_alu/mult_251/product[63] (alu_DW_mult_uns_0)       0.00
7.82 f
  dp/m_alu/U172/O (AO12X1)                                0.08
7.90 f
  dp/m_alu/flags[0] (alu)                                 0.00
7.90 f
  dp/m_alu_latch/flags[0] (alu_latch)                     0.00
7.90 f
  dp/m_alu_latch/flags_out_reg[0]/D (QDFCLRBX1)           0.00
7.90 f
  data arrival time
7.90

  clock CLK_0 (rise edge)                                 8.00
8.00
  clock network delay (ideal)                             0.00
8.00
  dp/m_alu_latch/flags_out_reg[0]/CK (QDFCLRBX1)          0.00
8.00 r
  library setup time                                     -0.09
7.91
  data required time
7.91
  ----------------------------------------------------------------
---------
  data required time
7.91
  data arrival time
-7.90
  ----------------------------------------------------------------
---------
  slack (MET)
0.01
```

Fig 6. Simulation Result of Timing Analysis

4.3. Power

The switching and the leakage power for all the cells are estimated. Total power consumed by the design is 1.72 mW @ 126MHz, which is well below the requirements specified in [2]. Power report is shown in Fig 6. and RTL schematic is shown in Fig 7.



```
  Cell Internal Power  =   1.5800 mW   (97%)
  Net Switching Power  =  56.8179 uW    (3%)
                         ---------
Total Dynamic Power    =   1.6369 mW  (100%)

Cell Leakage Power     =  92.8026 uW


                  Internal         Switching          Leakage
Total
Power Group       Power            Power              Power
Power    (   %    )  Attrs
-----------------------------------------------------------------
---------------------------------
io_pad              0.0000           0.0000            0.0000
0.0000   (    0.00%)
memory              0.0000           0.0000            0.0000
0.0000   (    0.00%)
black_box           0.0000           0.0000            0.0000
0.0000   (    0.00%)
clock_network       0.0000           0.0000            0.0000
0.0000   (    0.00%)
register            1.5663       1.6156e-03        2.5953e+07
1.5939   (  92.15%)
sequential          0.0000           0.0000        4.5668e+05
4.5668e-04   (    0.03%)
combinational   1.3746e-02       5.5202e-02        6.6392e+07
0.1353   (    7.82%)
-----------------------------------------------------------------
---------------------------------
Total               1.5800 mW    5.6818e-02 mW     9.2802e+07 pW
1.7297 mW
```

Fig 7. Simulation Result of Power Consumption

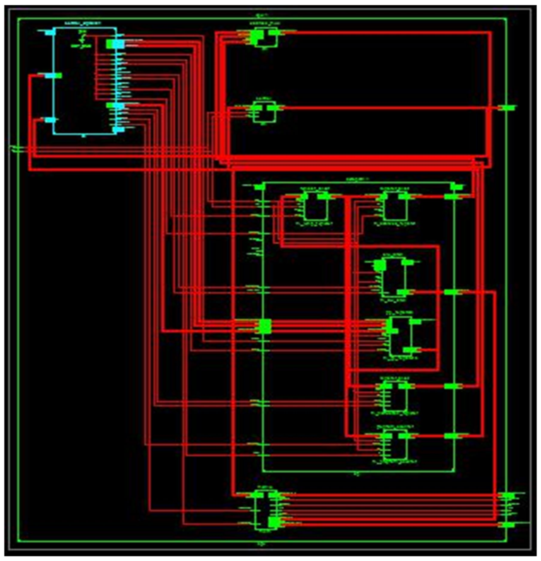


Fig 8. RTL Schematic of Core

5. CONCLUSION

The complete procedure of designing and simulating the generic 32 bit SoC core (co-processor) for IoT applications has been out laid in this paper. It has unique ISA that support several features. The processor is designed in verilog and verified using synopsys VCS tool. Report from the Design Compiler demonstrates that this is an optimized code. This Verilog code fits well into small field-programmable gate arrays (FPGAs), complex programmable logic devices (CPLDs) and application specific integrated circuits (ASICs) and therefore is ideally suited for IoT applications, especially wearable devices. This supports five stage pipelining, inorder to increase the speed without undue power consumption. The clock speed constraint has been met and there is a slack of 0.01 ns. The total estimated power consumed is 1.72 mW and the area constraint is met and the consumed area for the entire core is 66562.3 μm^2 which proves that it is suitable candidate for IoT co-processor.

ACKNOWLEDGEMENT

The authors would like to acknowledge the help rendered by RISC V group, RISE lab,IITM, Chennai.

REFERENCES

[1] http://www.gartner.com/technology/research/hype-cycles
[2] http://electronicdesign.com/datasheet/iot-requires-new-type-low-power-processor-pdf-download
[3] http://www.synopsys.com/dw/doc.php/ds/cc/dw-processor-solutions.pdf
[4] http://www.arm.com/products/processors/cortex-m/cortex-m0plus.php
[5] StevenBell, "Microprocessor Final Design Document,"Elect. Eng., Oklahomo Christian Univ,Dec. 2010.
[6] John L. Hennessy, David A.Patterson, "Computer Architecture," vth ed. Kundli: Elsevier,2012, pp. 262-334.

Authors

P.Jennifer received the B.E. degree from Asan Memorial College of Engineering and Technology, Anna University, Tamil Nadu and currently pursuing M.E in VLSI Design in RMK Engineering College, Anna University.

S.Ramasamy received the M.Tech degree in VLSI Design and the Ph.D. degree in Mixed signal Design from National Institute of Technology,Tiruchirappalli, Tamil Nadu.He is with the Department of Electronics and Communication Engineering, RMK Engineering College, Anna University, since 2010.His research interests include SoC Design, IoT and Mixed signal Systems.