Embedded Soc using High Performance Arm Core Processor

D.sridhar raja
Assistant professor, Dept. of E&I, Bharath university, Chennai

Abstract: *ARM is one of the most licensed and thus widespread processor cores in the world. Used especially in portable devices due to low power consumption and reasonable performance. The ARM processor core is a leading processor design for the embedded domain. In the embedded domain, both memory and energy are important concerns. For this reason the 32 bit ARM processor also supports the 16 bit Thumb instruction set. For a given program, typically the Thumb code is smaller than the ARM code. Therefore by using Thumb code the I-cache activity, and hence the energy consumed by the I-cache, can be reduced. However, the limitations of the Thumb instruction set, in comparison to the ARM instruction set, can often lead to generation of poorer quality code. Thus, while Thumb code may be smaller than ARM code, it may perform poorly and thus may not lead to overall energy savings. We present a comparative evaluation of ARM and Thumb code to establish the above claims and present analysis of Thumb instruction set restrictions that lead to the loss of performance. We propose profile guided algorithms for generating mixed ARM and Thumb code for application programs so that the resulting code gives significant code size reductions without loss in performance. Our experiments show that this approach is successful and in fact in some cases the mixed code outperforms both ARM and Thumb code. This ARM processor is designed using pipelined architecture; through this we can improve the speed of the operation. In this we are using 5-stage pipelining. The 5 stages are Fetch, Decode, Execute, Memory and Write Back. During the design process we are including varies low power techniques in architectural level also we are proved that our proposed methods is more efficient than Back-end low power reduction techniques*

Keywords: *Architectural level power reduction, High speed architecture, Micro architecture, Common power format*

1. INTRODUCTION

Low power has emerged as a principle theme in today's electronics industry. The need for low power has caused a major paradigm shift where power dissipation has become as important consideration as performance and area. The ARM processor core is a key component of many successful 32-bit embedded systems. This processor will follow will follow the RISC architecture because it supports a predefined set of instructions. In this all the instructions have same length. RISC processors, first developed in the eighties, seem predestined to dominate the computer industry in the nineties and to relegate old microprocessor architectures into oblivion. These processors are used in a wide variety of applications include cars, phones, digital cameras, printers and other such devices. Embedded processors demand instruction set suitable for the specific applications, various and fast interrupt handling, low power consumption and so on. or this properties RISC is more efficient in comparison with micro programmed than CISC.

This paper describes a 32-bit ARM(Advanced RISC Machine) processor designed for embedded and portable application. The ARM, then, was born through a serendipitous combination of factors, andbecame the core component in Acorn's product line. Later, after a judicious modification of the acronym expansion to Advanced RISC Machine**,** it lent its name to the company formed to broaden its market beyond Acorn's product range. Despite the change of name, the architecture still remains close to the original Acorn design.

2. INSTRUCTION SET

There are three basic types of instructions supported by this processor. These are the Register type, Jump type, and the Immediate type. The instruction format is shown in the below figure.

Register type: In this format bits 31-26 represents the opcode. Bits 25-21 represent the address of the first source register. Bits 20-16 give the address of the second source register. Bits 15-11 represents the address of the destination register. The bits from 10-6 represents the number of bits to be shifted. The last six bits, 5-0, represent the function code that represents the ALU function that is to be performed. If the opcode bits, 31-26, do not indicate an ALU function, then the function bits are ignored.

Immediate type: As with the Register Type instruction, bits 31-26 represents the opcode. Bits 25-21 represent the address of the first source register. Bits 20-16 give the address of the second source register, and Bits 15-0 of this instruction type represent an 16-bit immediate value given in 2's complement form. When the opcode represents

a unary operation, the value in this immediate field is used as the operand.

Jump type: Bits 31-26 of this instruction format represents the type of branch operation to be performed. The remaining 26 bits, 25-0, represent the branch offset in 2's complement format. This number is added to the value of the PC to obtain the branch target address.

Field size	5 bits	1 bit	5 bits	5 bits	5 bits	5 bits	6 bits
R-type	Opcode	Thumb	Rs	Rt	Rd	shift	Function
I-type	Opcode	Thumb	Rs	Rt	Immediate value (16 bits)		
B-type	Opcode	Thumb	Branch offset (26 bits)				

Table 1. Instruction format

The most significant feature of instruction set is that all of instruction may be executed conditionally according to the condition flags. A condition field within all instructions is compared with the condition flag in program status register. And the result of the comparison determines whether the instruction can be executed or not. This reduces the need for forward branches and allows very dense in-line code to be written.

3. FUNCTIONAL BLOCK DIAGRAM

The architecture consists of five stage pipelining. Instruction fetch, Instruction decode, Execute, Memory and Write Back. Also, we added a Data Forward and Hazard detection unit to maintain proper data flow through the pipeline stages. The architecture of the ARM processor is shown in the below figure

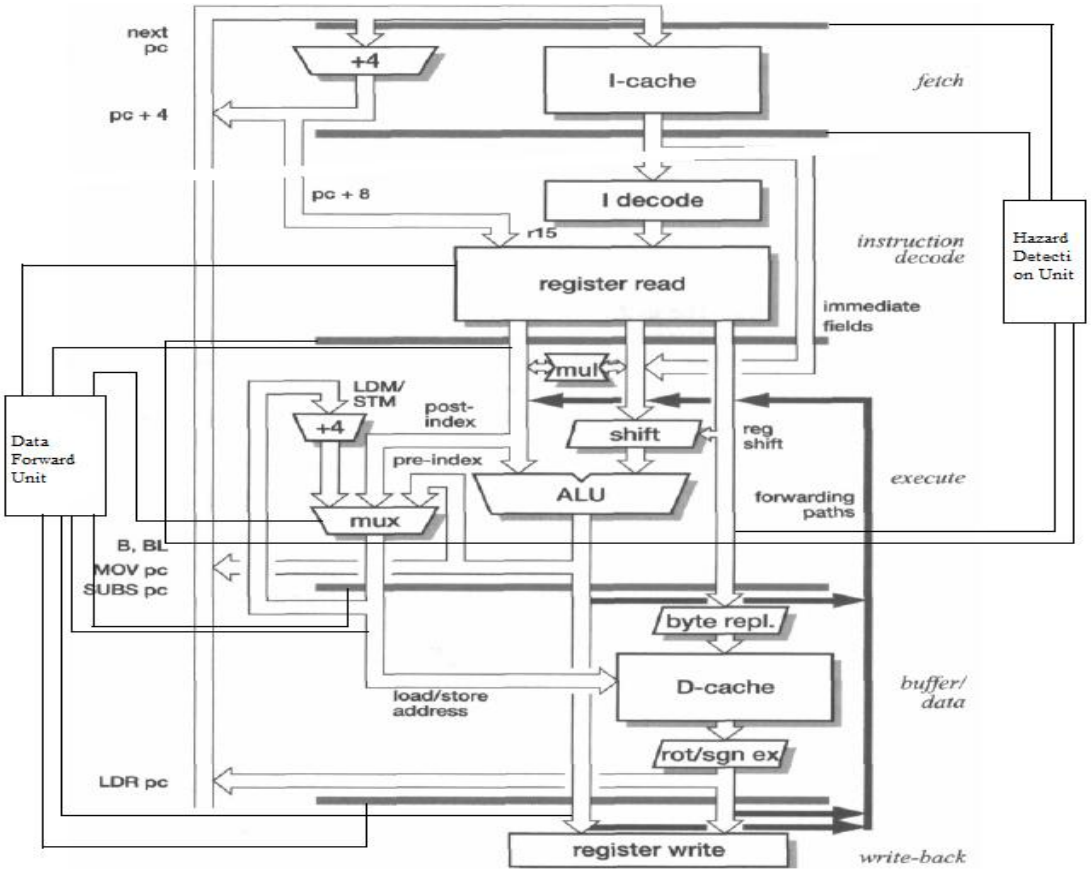

Fig 1

3.1 Instruction Fetch:

This stage consists of Program counter, Instruction Memory, and the Branch Decode unit.

3.1.1 Program Counter:

The program counter (PC) contains the address, of the instruction that will be fetched from the Instruction memory during the next clock cycle. Normally the PC is incremented by one during each clock cycle unless a branch instruction is executed. When a branch instruction is encountered, the PC is incremented/decremented by the amount indicated by the branch offset.

3.1.2 Instruction Memory:

The Instruction Memory contains the instructions that are executed by the processor. The input to this unit is a 32-bit address from the program counter and the output is 16-bit instruction word. This module supports up to 4 G words of memory, where each word is 32-bit long.

3.1.3 Branch Unit:

The Branch Decide Unit is responsible for determining whether a branch is to take place or not based on the 2-bit Branch signal from the control Unit and the Zero flag from the Arithmetic and Logic Unit (ALU). The output of this unit is a 1-bit value which is high when a branch is to take place, and otherwise it is low.

3.2 Instruction Decode:

This stage consists of the Control Unit, Register File, Y-Register, and the Sign Extend Unit.

3.2.1 Control Unit:

The control unit generates all the control signals needed to control the coordination among the entire component of the processor. The input to this unit is the 4-bit opcode field of the instruction word. This unit generates signals that control all the read and write operation of the register file, Y-Register and the Data memory. It is also responsible for generating signals that decide when to use the multiplier and when to use the ALU, and it also generates appropriate branch flags that are used by the Branch Decide unit.

3.2.2 Y-Register:

The Y-Register is a special 16-bit register that is used to store the upper 32 bits (bits 32-63) of the result generated by the multiplier. When the Y_Write signal is high new value is written to this register, otherwise the currently stored value is outputted.

3.2.3 Thump decoder:

Thumb code extends this advantage to give ARM better code density than most CISC processors. If the Thumb instruction set is considered part of the ARM architecture, we could also add: a very dense 16-bit compressed representation of the instruction set in the Thumb architecture. Thumb is not a complete architecture; it is not anticipated that a processor would execute Thumb instructions without also supporting the ARM instruction set.

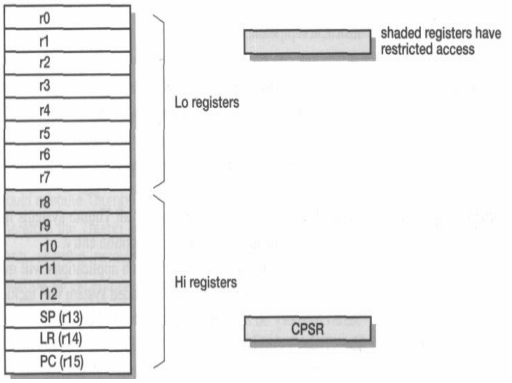

Fig 2

3.2.4 Register file:

This is a two port register file which can perform two simultaneous read and one write operation. It contains thirty two 32-bit general purpose registers. The registers are named R0 through R31. R0 is a special register which always contains the value zero and any write request to this register is always ignored.

3.2.5 Sign Extend Unit:

The input to this unit is an 16-bit immediate value provided by all the immediate type instructions. This unit sign extends the 16-bit value to a 32-bit value signed value.

3.3 Execute

This stage consists of the Branch Adder, Multiplier, Arithmetic Logic Unit (ALU), and the ALU Control Unit.

3.3.1 Branch Adder:

The branch adder adds the 12-bit signed branch offset with the current value of the PC to calculate the branch target. The 12-bit offset is provided by the branch instruction. The output of this unit goes to the PC control multiplexer which updates the PC with this value only when a branch is to be taken.

3.3.2 Multiplier:

The high-level block diagram of the multiplier is shown below. It consists of four distinct components. They are the Booth Encoder, Partial Product Generator, Carry Save Adder, and the Carry Look ahead Adder. Our multiplier architecture employs two main techniques to increase the speed of the multiplication process. First technique is to reduce the number of partial products and the second is to increase the speed at which the partial products are added.

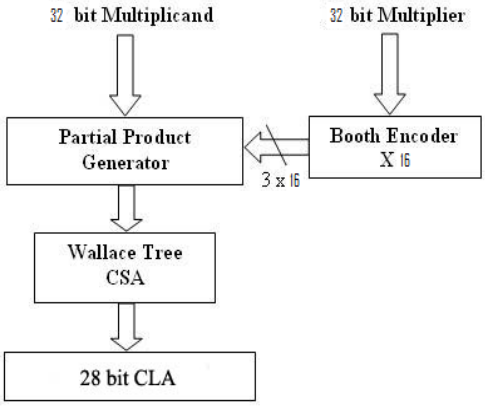

Fig 3.

3.3.2.1 Booth Encoder

This module encodes the 32-bit multiplier using radix 4 Booth's algorithm. Radix 4 encoding reduces the total number of multiplier digits by a factor of two, which means in this case the number of multiplier digits will reduce from 32 to 16. Each encoder produces a 3-bit output where the first bit represents the number 1 and the second bit represents the number 2. The third and final bit indicates whether the number in the first or second bit is negative.

3.3.2.2 Partial Product Generator (PPG)

The output from the Booth encoder is used in this module to generate the partial products. Since there are 16 Booth encoders there will be a total of 16 partial products. The multiplication by two is implemented by shifting the multiplicand left one bit and the negation is implemented by taking the two's complement of the multiplicand.

3.3.2.3 Wallace Tree

This module is responsible for adding the partial products that were generated in the PPG module. This module uses 3 to 2 carry save adders (CSA) to implement the Wallace Tree. The individual CSAs are nothing more than full adders with the exception that the carryins and the carry-outs are handled in a special way. Each column of numbers in the partial product is added using this method. The carry-outs generated in each stage of addition are transferred to the Wallace Tree of the column of bits of partial products on the left and the carry-ins comes from the column to the right.

3.3.2.4 Carry Look ahead Adder (CLA)

This unit is used to add the final sum and carry vectors generated by the Wallace Trees for each column of bits from the partial products. Only a 28-bit CLA is needed, instead of a full 32 bits, because some of the bits of the final result are already available from the Wallace Trees.

3.3.3 Arithmetic Logic Unit (ALU)

The ALU is responsible for all arithmetic and logic operations that take place within the processor. These operations can have one operand or two, with these values coming from either the register file or from the immediate value from the instruction directly. The operations supported by the ALU include add, subtract, compare, and, or, not, xor, rotate, logical shift, and arithmetic shift. The output of the ALU goes either to the data memory (in the case where the output is an address) or through a multiplexer back to the register file.

3.3.4 ALU Control Unit

This unit is responsible for providing signals to the ALU that indicates the operation that the ALU will perform. The input to this unit is the 6-bit opcode and the 6-bit function field of the instruction word. It uses these bits to decide the correct ALU operation for the current instruction cycle. This unit also provides another set of output that is used to gate the signals to the parts of the ALU that it will not be using for the current operation.

3.4 D. Memory

This stage consists of the Data Memory module.

3.4.1 Data Memory

This module supports up to 64k words of 16-bit data words. The Load and Store instructions are used to access this module. When new data is to be written to the memory, the Mem_Write signal is asserted. When the Mem_Write signal is low, a read operation is performed for the given memory location.

3.5 Write Back

This stage consists of some control circuitry that forwards the appropriate data, generated by the ALU/MAC or read from the Data Memory, to the register files to be written into the designated register.

3.6 Data Forward Unit

This unit is responsible for maintaining proper data flow to the ALU and the Multiplier. The primary function of this unit is to compare the destination

register address of the data waiting in the Memory and Write Back pipeline registers to be written back to the register file with the current data needed by the ALU or the Multiplier and forward the most up-to-date data to these units. By forwarding the data at the appropriate time, this unit makes sure that the pipeline works smoothly and does not stall as a result of data dependencies.

3.7 Hazard Detection Unit

This unit detects conditions under which data forwarding is not possible and stalls the pipeline for one or two clock cycles in order to make sure that instructions are executed with the correct data set. When it detects that a stall is necessary, it disables any write operation in the instruction decode pipeline registers, stops the PC from incrementing, and clears all the control signals generated by the control unit. By taking these steps it can delay the execution of any instruction by one clock cycle.

4 LOW POWER TECHNIQUES

The power reduction techniques are given below

Clock gating: The main power reducing method that has been explored in this architecture is clock gating. Clock gating is a method where the clock signal is prevented from reaching the various modules of the processor. The absence of the clock signal prevents any register and/or flip-flop from changing their value. As a result of this, the input to any combinational logic circuit remains unchanged, and thus no switching activity takes place in those circuits. Since, in CMOS circuits, most of the power dissipation results from switching activity, clock gating greatly reduces the overall power consumption.

5. HARDWARE IMPLEMENTATION

This processor is described and verified by using VHDL. The design is simulated using the MODELSIM simulator. This design is synthesized by using Altera Quartus II synthesizer. The assembly codes are made and compiled to generate the machine codes. These instruction codes are used to verify the operations of each functional unit. The logic and timing verifications are performed with the netlist provided by the Altera synthesizer. The output of the processor is verified by using text io.

The implementation is done on Altera Cyclon II FPGA and verified the operations of the processor. The pwer dissipation will reduce in this design.

6. CONCLUSION

The ARM processor for embedded and portable application has been designed with 5-stage pipeline and verified it. This processor was designed with high speed and it is performed in low power. We can reduce the power using the thumb decoder. We got the power dissipation as µW range. This is less than the conventional one.

7. REFERENCE

[1] A 32-bit RISC core for embedded application “Sung Ho Kwak, Seung Ho Lee, Byeong Yoon Choi, Moon Key Lee”

[2] ARM system on chip architecture “steve Furber”.

[3] ARM system developers guide “Andrew N Sloss, Dominic Symes, Chris Wright”.

[4] Reuse methodology manual for SOC designs “Michael keating and Pierre Bricaud”.

[5] Synthesis and Optimization of DSP algorithms “George A Constantinides, Peter Y. K Cheung and Wayne Luk”.

[6] ARM architecture Reference manual.