Design and Implementation Of a SHARC Digital Signal Processor Core In Verilog HDL

*N.Mozaffar and N. Z. Azeemi**
Department of Hardware, Streaming Networks Pvt. Ltd. Islamabad, Pakistan.
*Asst. Professor Department of Computer Engineering, VLSI Lab, CIIT, Islamabad, Pakistan
Emails: naveed_33@yahoo.com,nazeemi@comsats.edu.pk

Abstract—*This paper describes the design and implementation of an 8-bit fixed point Digital Signal Processor Core in verilog HDL. The architecture exploits the principles of pipelining and parallelism in order to obtain high speed and throughput. The modules of the design fit on a XILINX XC4010XL FPGA with 130K gates running at a clock frequency of 32.31 MHz.*
The proposed architecture follows the Analog Devices, SHARC DSP standard. This DSP architecture balances a high performance processor core with high performance buses, Program Memory (PM) and Data Memory (DM). In the core, every instruction can ***execute*** *in a single cycle. The buses and instruction cache provide rapid, unimpeded data flow to the core to maintain the execution rate.*

Keywords: *VLSI, Verilog HDL, FPGA, SHARC (Super Harvard Architecture), FFT(Fast Fourier transform),CLB (Configurable logic blocks).*

1. INTRODUCTION

A Digital Signal Processor is a type of microprocessor one that is incredibly fast and powerful. A DSP is unique because it processes data in real time. These real time processor makes up the fastest growing segment of the semiconductor market and is particularly well suited to handle the demands of processing information, whether as the engine of communication applications or by providing the processing platform for the convergence of the Internet and wireless applications.
One of the best ways of implementing Digital Signal Processor is the FPGA FPGA's have the capability of being reconfigurable within a system, which can be a big advantage in applications that need multiple trial version within development, offering reasonably fast time to market.
This motivates the design and implementation of Digital Signal Processor in hardware presented in this article.
The design is implemented using Verilog HDL. Each of the nine modules is coded individually. The functional and timing simulations is performed followed by the synthesis. All these sub-modules are interconnected in a top-module, which is also verified and synthesized.
Analog Devices' SHARC® DSP is based on a 32-bit super Harvard architecture that includes a unique memory architecture comprised of two large on-chip, dual-ported SRAM blocks coupled with a sophisticated I/O processor, which gives SHARC the bandwidth for sustained high-speed computations. The Block diagram of SHARC architecture is illustrated in Figure 1.

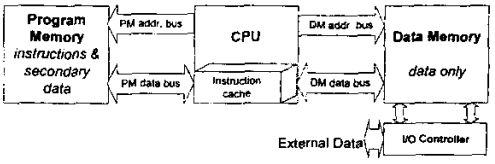

Figure 1. Super Harvard Architecture

The paper is organized as follows. An outline of the Digital Signal Processor is discussed in Section 2 and is followed by architecture details in Section 3. The synthesis results are provided in Section 8 with conclusion in Section 9.

2. ARCHITECTURE OUTLINE

The system architecture for the implementation is shown in Figure 2. In order to achieve high throughput, Super Harvard architecture is implemented. The architecture consists of a numeric execution unit, memory modules, and the main control unit.

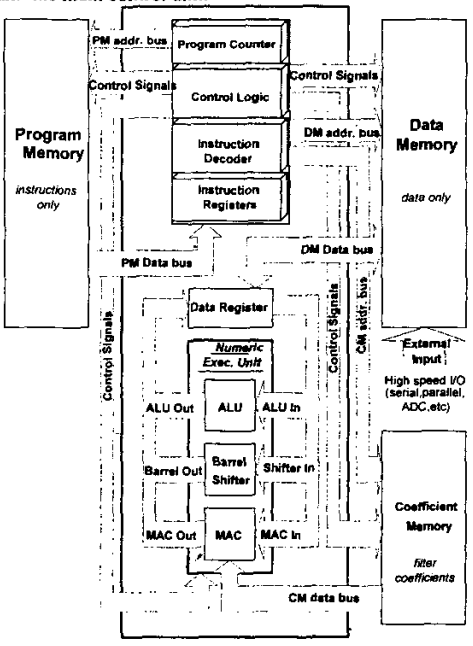

Figure 2. DSP System Architecture Block Diagram

The proposed design uses a register-register based data path prototype, where operands are fetched from the register by the functional units and then the results are written back to the register file.

The DSP can be operated in either of the two modes.

- ***Real Time Mode:***
  In this mode the DSP accepts real time data from the field and process it.

- ***Off line Mode:***
  In this mode the DSP can operate on the data stored in the Data Memory.

The design is verified for Offline mode only.

3. ARCHITECTURE DEATILS

The design consist of three main numeric execution units for data processing as described below:

3.1 Arithmetic Logic Unit (ALU)

An Arithmetic Logic Unit is the center core of a central processing unit. It consists of a Purely combinational logic and performs a set of arithmetic and logic micro operations on two input buses. It has *n* encoded inputs for selecting which operation to perform. The select lines are decoded within the ALU to provide upto *2n* different operations. The ALU implements a wide range of arithmetic and logical functions. The ALU transfer the result to a destination accumulator after an operation is performed. Instructions that perform memory –to- memory operation are exceptions.
The proposed design of the ALU is capable of performing 8 different micro operations.

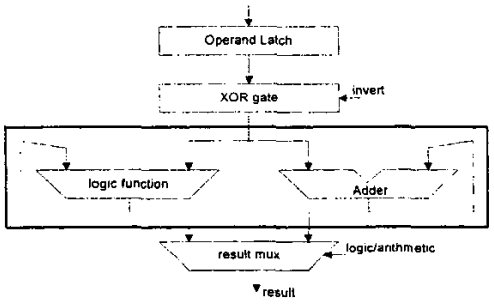

Figure 3. ALU System Architecture

3.2 Barrel Shifter

Barrel shifter circulates data bits in a synchronous manner. The Barrel Shifter is used for Scaling operations such as:

- Prescaling an input data memory operand or the accumulator value before an ALU operation.
- Performing a logical shift or arithmetic shift of the accumulator value.
- Normalizing the accumulator.
- Post scaling the accumulator before storing the accumulator value in data register.

In the Fig below the top register shows the pattern before the shift, and the bottom register shows the pattern that results from the shift.

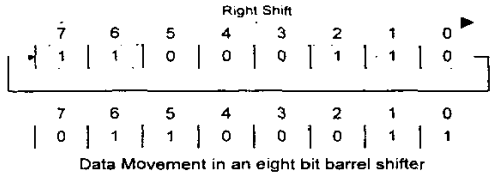

Figure 4. Right Shift operation

3.3 Multiply and Accumulate Unit (MAC)

A hardware Multiplier and Multiplier-Accumulator (MAC) are standard components in all off-the-shelf DSPs. Multipliers are extensively used in signal processing and communication systems.
The Multiplier/adder unit provides multiply and accumulates (MAC) capability. The overall function of the Multiply Accumulator (MAC) is given by equation (1).

$$Q = \sum_{n=0}^{Count-1} (+1)\ \mathbf{Multiplicand}(n) * \mathbf{Multiplier}(n) \qquad (1)$$

'Q' is the primary data output. 'A' and 'B' are multiplied together and the product added from the current result. The *count* value in the Equation is set to a fixed value by a parameter.
The MAC uses *Array Multiplier* for multiplication and a high-speed hardware Implemented *adder* for addition.

Figure 5 shows the proposed MAC architecture for implementing in the DSP.

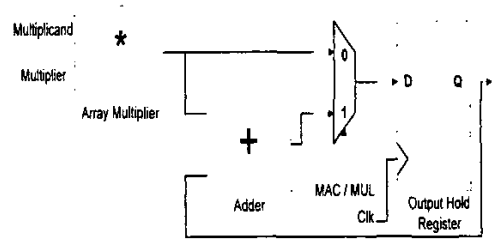

Figure 5. The MAC System Architecture

As evident form the Figure this architecture is base on an Array Multiplier, a Adder, a Mux, and Output hold register.

There are two modes of MAC operation control by the input pin MAC / MUL:

Mode 1:

MAC / MUL = 0

Array Multiplier performs only Multiplication on the input data.

Mode 2:

MAC / MUL = 1

In this mode it performs Multiplication and accumulation of the input data i.e. it is in MAC mode.

3.3.1 Array Multiplier

Due to today's VLSI technology, the array (or parallel) multiplier has become increasingly economical and popular. Some multiplier basics, notations and conventions are described here.
Generation of partial products is the first step of multiplication operation. These partial products need to be added together for generating the final product of multiplication. Figure 6 shows the process of partial product generation in the multiplication of two unsigned numbers *a* and *b*. Adding all partial products generates the final product. So the process of multiplication consists of generating partial products and adding them together.

$$
\begin{array}{cccccccc}
 & & a_5 & a_4 & a_3 & a_2 & a_1 & a_0 \\
 & & b_5 & b_4 & b_3 & b_2 & b_1 & b_0 \\
 & & a_5b_0 & a_4b_0 & a_3b_0 & a_2b_0 & a_1b_0 & a_0b_0 \\
 & a_5b_1 & a_4b_1 & a_3b_1 & a_2b_1 & a_1b_1 & a_0b_1 & \\
 a_5b_2 & a_4b_2 & a_3b_2 & a_2b_2 & a_1b_2 & a_0b_2 & & \\
 a_5b_3 & a_4b_3 & a_3b_3 & a_2b_3 & a_1b_3 & a_0b_3 & & \\
 a_5b_4 & a_4b_4 & a_3b_4 & a_2b_4 & a_1b_4 & a_0b_4 & & \\
 a_5b_5 & a_4b_5 & a_3b_5 & a_2b_5 & a_1b_5 & a_0b_5 & &
\end{array}
$$

Figure 6. Multiplication of two 4-bit no.

An NxM array multiplier architecture consists of three well-defined major sections performing different operations:

- The generation of N partial products layers.
- The reduction of these partial products into 2 layers.
- And addition of the two layers into a final product

Figure 7. Visually demonstrates the three major sections for performing array multiplication.

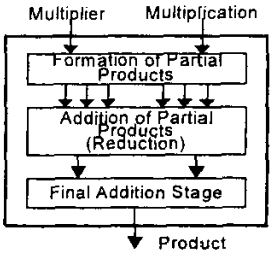

Figure 7. Steps required in multiplication

For partial product generation multiplicand x is multiplied by x^2^i for all i. Adding all bits of the multiplicand with ith bit of the multiplier generates pp_i (partial product for ith bit) and then shift pp_i left. Due to its concurrency, only one level delay is introduced Independent of the multiplication word length. Hence this area needs no fast speed algorithms for generation of partial products.

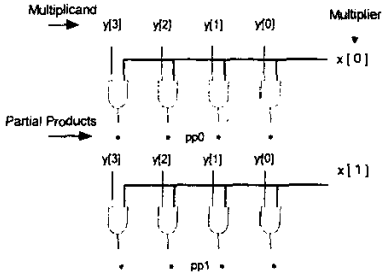

Figure 8. Generation of Partial Products.

The term reduction will be best explained by the following visual method of reduction in Figure 9.

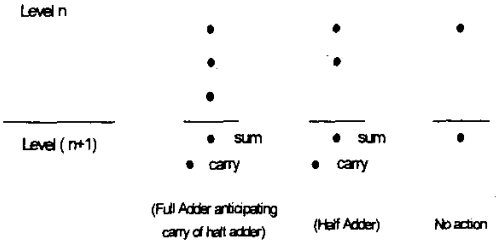

Figure 9. Reduction of Partial Products.

In the above Figure three dots each symbolizes a partial product. Using FA (Full Adder) reduces these to two bits, where one has the weight of 2^0(sum) and the other 2^1(carry). This type of reduction is known as 3 to 2 reduction or carry saves addition. The two dots are reduced to 2 using a HA (Half Adder). It can be seen that this stage (Level n+1) does not yield any reduction. The rightmost diagram has 1 dot, which is carried down without any action.

3.3.2 Adder

The speed of a signal processing or communication system ASIC depends heavily on these functional units. Adders are in the critical path of many other arithmetic operations like multiplication, scaling, add-compare select, and division.
Two basic adder architectures are studied for implementation:

- ***Rippie Carry Adder.***
- ***Carry Look-ahead Adder.***

A ripple adder is not normally used in high-speed arithmetic. However in situations in which a minimum amount of hardware is required and speed is not critical, then a ripple adder can prove advantageous. The proposed design implements a *Carry Look-ahead Adder scheme for fast arithmetic operations.*

3. 4 CONTROLLER MODULES

This unit of the design serves as the torchbearer for the all the signals flowing. This means to say that it routes all the signals to their proper destinations. For some signals it makes sure that the signals arrive at their destined place to ensure the efficient operation of architecture. The Controller Unit includes three sub modules.

- Program Counter (PC)
- Instruction Decoder (ID)
- Control Unit (CU)

3. 4.1 Program Counter

The Program Counter is a 6-bit counter its output is connected to Program Memory. It generates a 6-bit count value, which is used to address the Program Memory, which stores the instructions. It can select 64 memory locations. The Program Counter usually holds the address of the next instruction from the instruction, which is currently executing.
It addresses a 64X11 bit Program Memory

3. 4.2 Instruction Decoder

The instruction decoder decodes the 11-bit instruction. It also generates control signals for the memory. The instructions contain the following information.

- Opcode.
- Address.
- Read control.
- Write control.
- CM address.
- CM read control

3. 4.3 Control Unit

The 5-bit opcode is further decoded by the control unit This module separate the 5-bits of opcode and assign them to the control register of the appropriate functional unit. These Control register is connected to the Function select pin of the three functional units.
This opcode contains the following information:

- Defines the function to be performed.
- Select a specific functional unit for that operation.
- Specify the source of data to that functional unit.

3.5 MEMORY MODULES

The proposed design uses three Memories for maximum throughput of the DSP. The design has separate memories for storing data, filter coefficients and instructions as specified below.

- Data Memory (DM)
- Coefficient Memory (CM)
- Program Memory (PM)

3.5.1 Data Memory

Data Memory is used to store only Data coming from either external source or Funcional units output registers.It can store 64 words each of 8 bits.The address bus width is of 6-bit thus it easily address 64 memory ocations.The memory is designed with separate read and write address bus thus enabling read and write from different locations of the memory at the same time in the interval of one clock cycle.At first the write instructions must be executed to write the data in the Data Memory.

3.5.2 Coefficient Memory

Coefficient Memory is used to store only Filter coefficients and twiddle factors in case of FFT.It can store 16 words each of 8 bits.The address bus width is of 4-bit thus it easilyaddress 16 memory ocations.The memory is designed with only a single address bus and a read control signal.Before simulation the memory is initialized with *Memory initialization file* which contains the required Coefficients needed for a particular operation.

3.5.3 Program Memory

Program Memory is used to store only intructions which are first given as input by the user during simulation .It can store 64 words each of 11 bits.The address bus width is of 6-bit thus it easily address 64 memory ocations.At first the write instructions must be executed to write the data in the Data Memory.

4. IMPLEMENTING FFT ALGORITHIUM

The proposed architecture is tested for FFT implementation, a digital signal processing algorithm used for calculating DFT. A 4-point radix-2 algorithm is implemented for this architecture as shown in Figure 10.

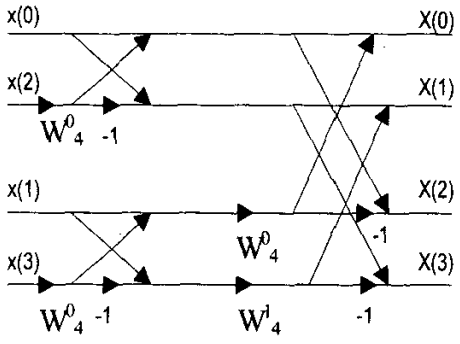

Figure 10.Flow graph for a 4-point radix-2 algorithm.

The W^0_4 and W^1_4 are the twiddle factors and are stored in the CM prior to the processing. For the convenience of understanding, we have supposed the values of these as the 1 and 2 respectively and are stored in the CM.The algorithm uses four butterflies and each requires 11 instructions and so 4 point DFT needs 44 instructions to be computed. Four more have been added to output the four outputs from the DM thus making a total 48 instructions. Now these 48 instructions will need 53 clock cycles to execute complying with the pipeline specifications.
The four.data inputs samples are coming from the external environment (in our case given input by the simulation waveform) and are stored in the memory. An efficient program (set of EEDSP001-the proposed DSP architecture-instruction sets) is written as shown in Table A, so that data samples are fetched in the bit reversed order. Since each write instruction takes 2 cycles to execute 8 operations take 11 cycles, as three write are included.

Similar three more sets of instructions are required with different operands and addresses to compute 4 point DFT.

4. 1 Simulation Waveform

The simulation waveform is generated by XILINX ISE Ver. 4.2 showing the computation of 4 point DFT. The input data samples are supposed as x (n) = {2, 4, 6, 1}. The computation is completed in 53 cycles and at last we have the output X (k) = {D, 2, 3, F6}. These values are displayed in the ALU_OUT register.

Table A. Instructions for the Computation of one butterfly.

Steps	Operation	Machine Instruction	Action
1	Multiply x (2) by W04	100_0000_0011	MAC
2	Store the result of step 1 in DM	110_0001_0000	WRITE
3	Retrieve the data of step 2 from DM	001_1001_0000	ALU
4	Subtract the data of step 3 from x (0)	000_0100_0001	ALU
5	Store the result of step 3 on the address of x (2)	110_0000_0011	WRITE
6	Retrieve the data of step 2 from DM	001_1001_0000	ALU
7	Add the data of step 6 from x (0)	000_0000_0001	ALU
8	Store the result of step 7 on the address of x (0)	110_0000_0001	WRITE

5. PIPELINING

The proposed DSP design is a fully pipelined architecture that has five stages. The pipelining in the design can be shown as follows:

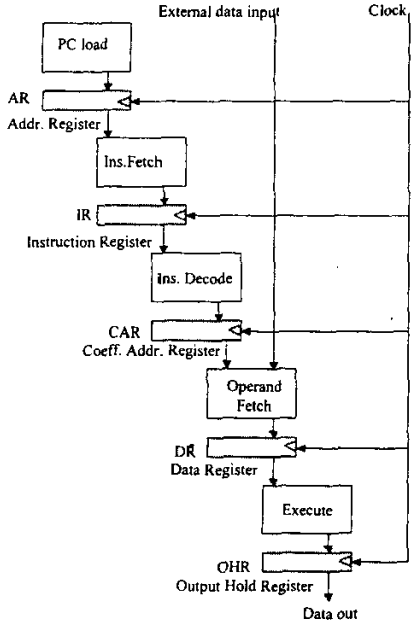

Figure 10. Five stages of pipelining in DSP Architecture.

In this pipelined architecture first instruction takes five cycles to execute and thereafter the subsequent instructions take only one cycle to execute. Therefore, the throughput has been increased. This initial delay of five cycles is called the ***latency*** of the architecture. Hence N instructions take N + 5 cycles to execute as compared to serial processing where N instructions will require Nx5 cycles showing an increase of five times in the throughput. One of the problems that has been encountered in the pipeline is that of the conflicting instructions. In our design it occurs in the write instruction. Therefore, in the pipelined architecture, control the flow of pipeline is a bit changed by delaying conflicting instruction. This method is called ***interlocking.*** Thus the write instruction takes two cycles to execute.

6. INTEGRATION OF COMPONENTS

To achieve the design of the Digital Signal Processor, the modules were implemented according to the plan set. The Digital Signal Processor was divided into several Components (top-down approach). The Functionality of each module that it must satisfy to Communicate with other modules were defined at that stage with great caution. This helped us in the bottom-up approach of integration of the modules to get a final Digital Signal Processor circuitry. The individual modules were combined to form the Digital Signal Processor core.

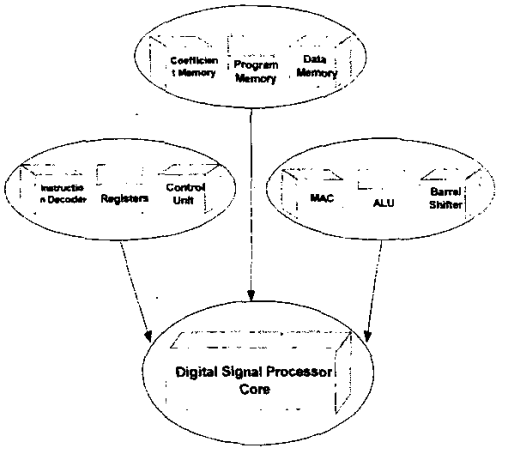

Figure 11.Integration of Components.

7. DESIGN FLOW USING EDA TOOL

The EDA Tool used for the synthesis and simulation of the DSP processor is XILINX ISE Ver. 4.2. First the architecture of the DSP is studied then the design is made first using traditional Paper-pencil approach and then the design is translated to the HDL language, Verilog HDL. After the designing process is completed the verilog description of the design is written and it is then imported in the EDA tools.

8. SYNTHESIS RESULTS

The architecture presented is highly modularized one that makes it very suitable for VLSI implementation. It has high input data rate of one sample per cycle and high throughput rate. Due to smooth data flow, control circuits are designed compactly as counter based logic with pause function to freeze all operations. The architecture is verified by Verilog simulation based on register transfer level descriptions. The entire architecture is fitted on a XILINX XC4010XL FPGA running at a clock frequency of 32.31 MHz. The implementation/Synthesis results are shown in the Table B.

Table B. Synthesis results on XILINX XC4010XL FPGA

Number Of Flip-Flops	154
Critical Path Timing	30.95 ns
Estimated Frequency	32.31 MHZ
Number Of CLBs	476
Total Equivalent Gate Count For The Design	7765
Additional Gate Count For IOBs	14408

9. CONCLUSION

DSPs is an answer to the intense need of high-speed and intensive processing technologies, which is both cheap and easy to use. DSPs is finding favor not only for computer systems, but also in consumer electronics products such as cellular phones.
The CPU and the Data path are designed to comprehend the details of DSPs architecture. The author develops the logic, architecture and interface.

9. REFERENCES

[1] Modeling, Synthesis, And Rapid Prototyping with the Verilog HDL, *Micheal D. Ciletti.*

[2] VLSI Digital Signal Processor, *Vijay K Madisett i.*

[3] The ADSP-21535 *DSP Hardware Reference Manual.*

[4] Erskine, C., and S. Magar, "Architecture and Applications of a Second-Generation Digital Signal Processor," Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing, USA, 1985.

[5] Lin, K., G. Frantz, and R. Simar, Jr., "The TMS320 Family of Digital Signal Processors," Proceedings of the IEEE, USA, Volume 75, Number 9, pages 1143-1159, September 1987.