Academic Open Internet Journal www.acadjournal.com Volume 21, 2007
ISSN 1311-4360

A NOVEL ARCHITECTURE FOR VLIW PROCESSOR

R. SESHASAYANAN
Lecturer, Department of Electronics and Communication
Anna University, Chennai – 600 025 e_mail:se_sha_sa@yahoo.com
DR.S.K.SRIVATSA
Professor, Department of Electronics and Communication
Anna University, Chennai – 600 025

ABSTRACT

Technology has seen the development of processor industry right from micro to the latest Nano-technology with speed being important criteria. Not much attention has been given to the power required to drive these Integrated Circuits. With gaining popularity in mobile computing, developing mobile processors have gained popularity since these processors possess unique properties like low power consumption and dissipation. This paper aims at designing a low power Very Long Instruction Word (VLIW) processor with built in FFT processor, which uses single clock frequency. This VLIW processor is designed with two modules one for VLIW processor and the other one is Hybrid dynamic voltage scaling module. Using the DVS algorithm the power can be reduced up to 20 to 25 % of normal VLIW processor. The design is simulated and synthesized using Xilinx project Navigator and the report is given in the paper.
Keyword: VLIW processor, VHDL, interDVS, intraDVS, Hybrid DVS etc.,

1 INTRODUCTION VLIW PROCESSOR

Recent high performance processors have depended on Instruction Level Parallelism (ILP) to achieve high execution speed. ILP processors achieve their high performance by causing multiple operations to execute in parallel using a combination of compiler and hardware techniques [11]. Very Long Instruction Word (VLIW) is one particular style of processor design that tries to achieve high levels of instruction level parallelism by executing long instruction words composed of multiple operations. The long instruction word called a MultiOp (multi operation) consists of multiple arithmetic, logic and control operations each of which would probably be an individual operation on a simple RISC processor. The VLIW processor concurrently executes the set of operations within a MultiOp thereby achieving instruction level parallelism. It is different from super scalar processor in the fact that instruction scheduling is done by the complier as compared to super scalar processor where instruction scheduling is done dynamically by the hardware.

VLIW architectures which issue one long instruction per cycle, where each long instruction called a MultiOp consists of many tightly coupled independent operations each of which execute in a small and statically predictable number of cycles. In such a system the task of grouping independent operations into a MultiOp is done by a compiler or binary translator. The processor freed from the cumbersome task of dependence analysis has to merely execute in parallel the operations contained within a MultiOp. This leads to simpler and faster processor implementations.

Floating point add	Floating point mul	and	add	mul	load	store	Branch	FFT

Figure 1 Format of MultiOP

One reason programming VLIWs is more difficult than writing code for a super scalar processor is that the program for a super scalar processor is inherently sequential and it is left to the hardware to extract parallelism from the sequential program. On the other hand, when generating code for a VLIW processor [15,16] the assembly language programmer or the compiler is faced with the task of extracting parallelism from a sequential algorithm and scheduling independent operations concurrently. For this reason, instruction-scheduling algorithms are critical to the performance of a VLIW processor. So for using a VLIW processor [11] the programmers must have knowledge about the various register sets available. This will help in extracting high level of parallelism from the processor.

Compilers for the VLIW processors use a 3-phase method to generate code. The phases are:

- Generate a sequential program. Analyze each basic block in the sequential program for independent operations.
- Schedule independent operations within the same block in parallel if sufficient hardware resources are available.
- Move operations between blocks when possible.

Usage of VLIW processor has a few constraints:

- There must be high degree of ILP in the program. Functional units must be duplicated which posses hardware design difficulties.
- Efficient compiler is required which can predict the path of branch instructions more accurately.

VLIW processor requires duplication of hardware. In the figure 2 shown below there are 2 fixed point ALU, 1 fixed point multiplication/division unit, 1 floating point addition/subtraction unit, 1 floating point multiplication/division unit, 1 load unit, 1store unit, 1 branch unit and FFT unit. This hardware duplication facilitates execution of several instructions simultaneously.

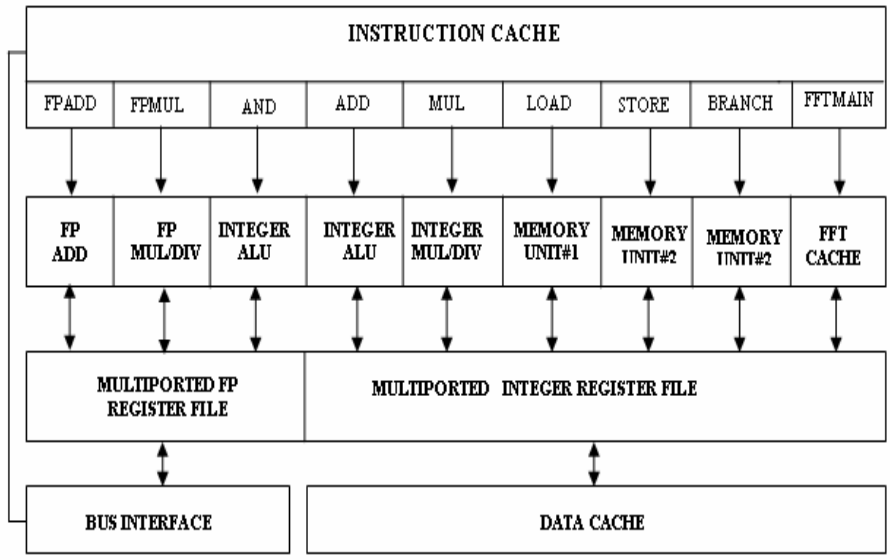

Figure 2 Functional Unit Duplication in VLIW Processor

Power consumption has become an overriding constraint for microprocessor designs, not only in mobile environments, but in desktop and server applications as well. Traditionally, the priority has been on performance, and consequently, the supply voltage has been set at the maximum allowable level based on device breakdown potentials to enable fast operation. However applications may not require the maximum achievable performance. A number of methods have been proposed that take advantage of these substantial periods of low utilization by scaling the supply voltage and clock frequency, resulting in a reduction in dynamic power consumption.

The basic goal of Hybrid DVS is to quickly adjust the processor's operating voltage or frequency at run time to the minimum level of performance required by the application. By continually adapting to the varying performance demands of the application energy efficiency is maximized. The variable voltage generation is done by DC-DC converter. This paper presents a VLIW processor with built in Hybrid DVS algorithm is shown in figure 3. This processor is designed to run in synchronous mode and to reduce the power consumption the interDVS and intraDVS [4] algorithms are add on to this processor.

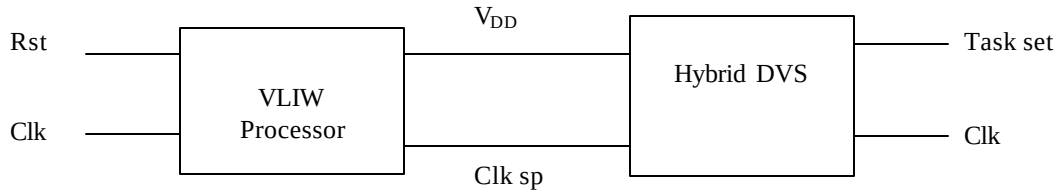

Figure 3 Block diagram of VLIW processor with HybridDVS

The VLIW processor has the ability to alter its execution voltage while in operation by using Dynamic Voltage scaling. This ability allows the processor to operate at the optimal energy/efficiency point and realize significant energy savings, which can be as much as 80% for some applications [1],[3].

$$E_{op} \propto V^2 \text{ and } f_{max} \propto (V-V_t)/V \quad \text{.....(1)}$$

Where E_{op} is the energy per operation f_{max} is the maximum clock frequency, and V is the operating voltage. To minimize the energy consumed by a given task; V can be reduced affecting a reduction in E_{op}. A reduction in V, as shown in the equation, results in a corresponding decrease in f_{max}. The implementation of Hybrid DVS [5, 6] and requires the application of scheduling algorithms. The effectively control DVS, a dynamic scheduler is used to dynamically adjust the processor speed and voltage at run time. Voltage scheduling significantly complicates the scheduling task since it allows optimization of the processor clock rate. Voltage schedulers analyze the current and past state of the system in order to predict the future workload of the processor.

2 NEED FOR THIS ARCHITECTURE

There are many VLIW processors are available, but these are not built with FFT application. The paper deals with VLIW processor with built in FFT processor. By using this architecture the processor can support a little amount of DSP application. This processor is capable of supporting different voltage levels. The voltage and the frequency of this processor are controlled by Hybrid DVS algorithm. By using this approach the power reduction can be achieved at about 20 to 25 % according to the workload of the processor.

3 ARCHITECTURE OF VLIW PROCESSOR

The VLIW processor designed is a 16-bit processor with Harvard architecture. It has separate code memory and data memory. So, the processor has access to code and data at the same time. This enables execution speedup. The code memory address bus is 16 bit and its data bus is 128 bit. The data memory address bus is 8 bit and its data bus is 16 bit.

It is capable of executing 6 instructions per cycle. It has a prefetch queue of size 6. The processor can execute the following operation at the same time.

- One floating point addition/subtraction
- One floating point multiplication/division
- Two fixed point ALU operation
- One fixed point multiplication/division
- One load/store operation.
- 64 point FFT operation

Floating point Add/Sub	Floating point Mul/Div	Fixed Point ALU	Fixed Point ALU	Fixed Point Mul/Div	Load/Store	FFT

Figure 4 Block diagram of VLIW processor

The processor has separate register sets for fixed-point operation and floating-point operation. Each register set has 64 registers. The fixed-point registers are 16 bit in length and floating-point registers are 32 bit in length. The individual registers are addresses with a six bit address. The basic block diagram of the VLIW processor is shown below

3.1 FIXED POINT ARITHMETIC AND LOGIC UNIT

The fixed point ALU consists of two components Arithmetic unit and Logic unit. The ALU is capable of handling 16 bits of data simultaneously, it has two 16-bit inputs A and B and has input carry bit C_{in} .The outputs are 16-bit G and carry output bit C_{out}. The 16 bit parallel adder is constructed using Carry Look Ahead logic to facilitate rapid addition of bits as compared to conventional Ripple Carry Adder. The B input logic is a digital circuit that enables use of just 2 select lines S_1, S_0 and minimal circuit components to differentiate between various arithmetic operations. The carry input and output bit are 'don't care' bits for logical operations. The logic unit is also capable of handling16 bit and it performs four basic logical operations AND, OR, NOT and XOR and gives the appropriate outputs based on the two select lines S_1, S_0. Using the mode select line S2 differentiates these two components and the basic operations done are listed below

- Arithmetic Operations
  - Addition
  - Subtraction
  - Increment
  - Decrement
- Logical Operations
  - AND
  - OR
  - NOT
  - XOR

The basic block diagram for the ALU is shown for one bit operation and the arithmetic unit and logic unit are differentiated using the Mode select line S_2.

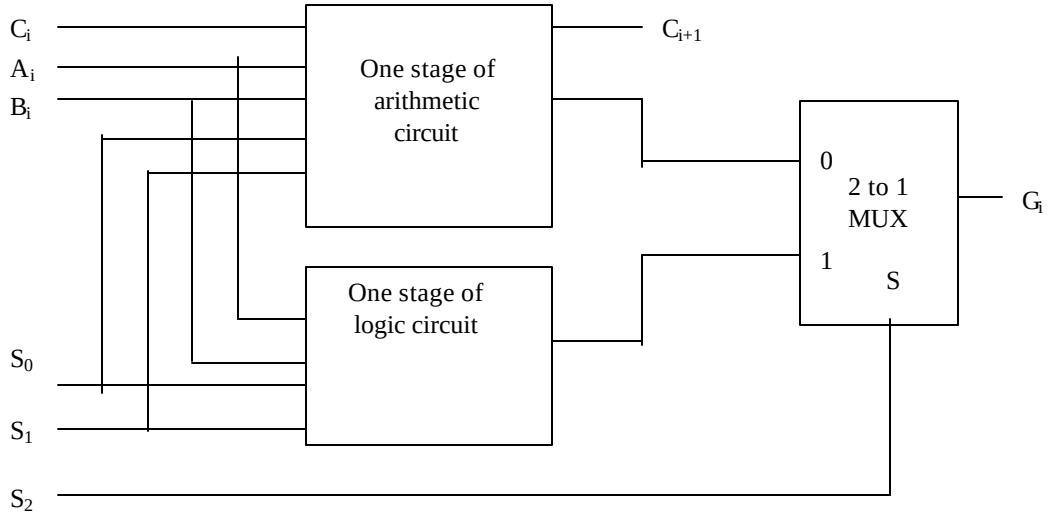

Figure 5 One Bit Operation of ALU

It takes into account one bit of inputs A_i, B_i and carry Ci and performs the corresponding arithmetic and logic operation based on the operation select lines S_1, S_0. The output G_i is selected from either arithmetic unit or logical unit output using a 2 to 1 Mux with S_2 as select line.

Operation Select				**Operation**	**Function**
S_2	**S_1**	**S_0**	**C_{in}**		
0	0	0	0	G = A	Transfer A
0	0	0	1	G = A + 1	Increment A
0	0	1	0	G = A + B	Addition
0	0	1	1	G = A + B + 1	Add with Carry input of 1
0	1	0	0	G = A + B’	A plus 1’s complement of B
0	1	0	1	G = A + B’ + 1	Subtraction
0	1	1	0	G = A – 1	Decrement A
0	1	1	1	G = A	Transfer A
1	0	0	x	G = A Λ B	AND
1	0	1	x	G = A v B	OR
1	1	0	x	G = A ⊕ B	XOR
1	1	1	x	G = A’	NOT (1’s Complement)

Table 1 Truth Table of ALU

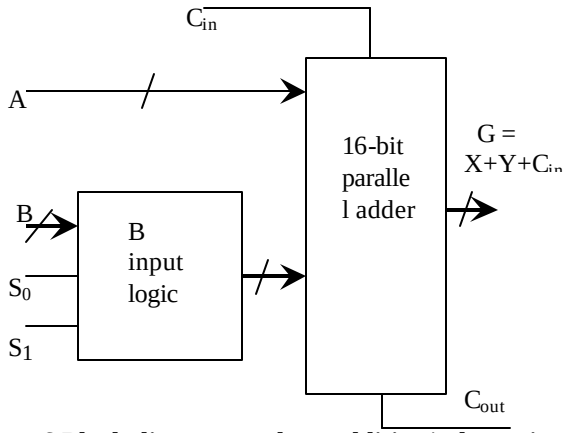

Figure 6 Block diagram to show addition/subtraction operation

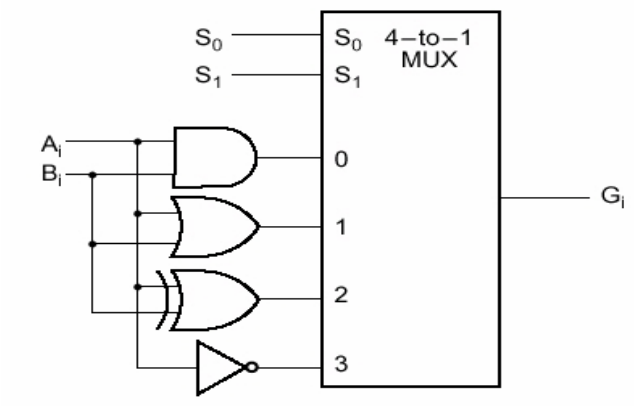

Figure 7 Block diagram for logical operation

3.2 FIXED POINT MULTIPLICATION/DIVISION UNIT

3.2.1 MULTIPLICATION UNIT

The multiplication unit performs both unsigned and signed multiplication. The select line S_0 is used to choose between unsigned and signed multiplication outputs. It takes two 8 bit inputs A and B and the output P is 16 bit. Both the multiplication is done using Extended Booth Algorithm. Extended Booth's Algorithm is preferred because of its rapid computation efficiency. Modified Booth's Algorithm needs just 'n/2' steps to arrive at the product of two n bit numbers compared to n steps in the case of Booth's Algorithm. It is twice as fast as Booth algorithm.

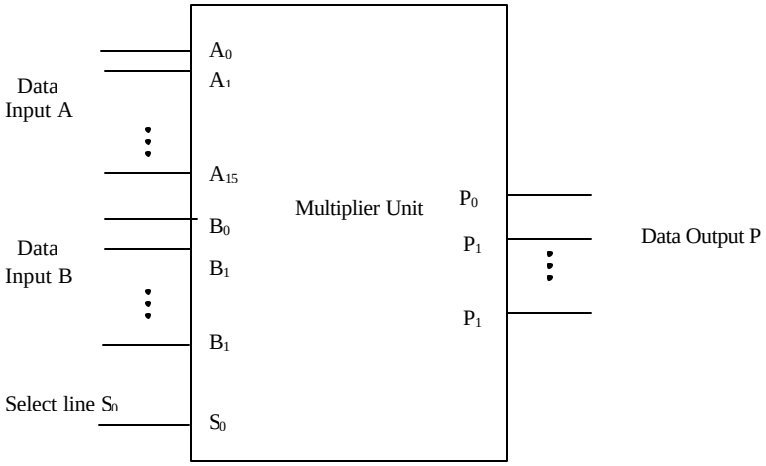

Figure 8 Multiplier Unit

3.2.2 DIVISION UNIT

The division unit performs both unsigned and signed division. The select line S_0 is used to choose between unsigned and signed division outputs. It takes 16 bit dividend (A) and 8 bit divisor (B) and the outputs are quotient (Q) 16 bits and remainder (R) 8 bit. Both the division is done using Non-Restoring

algorithm. Non-Restoring Algorithm needs just ‘n’ additions or subtractions compared to ‘3 n/2’ additions or subtractions in the case of Restoring Algorithm.

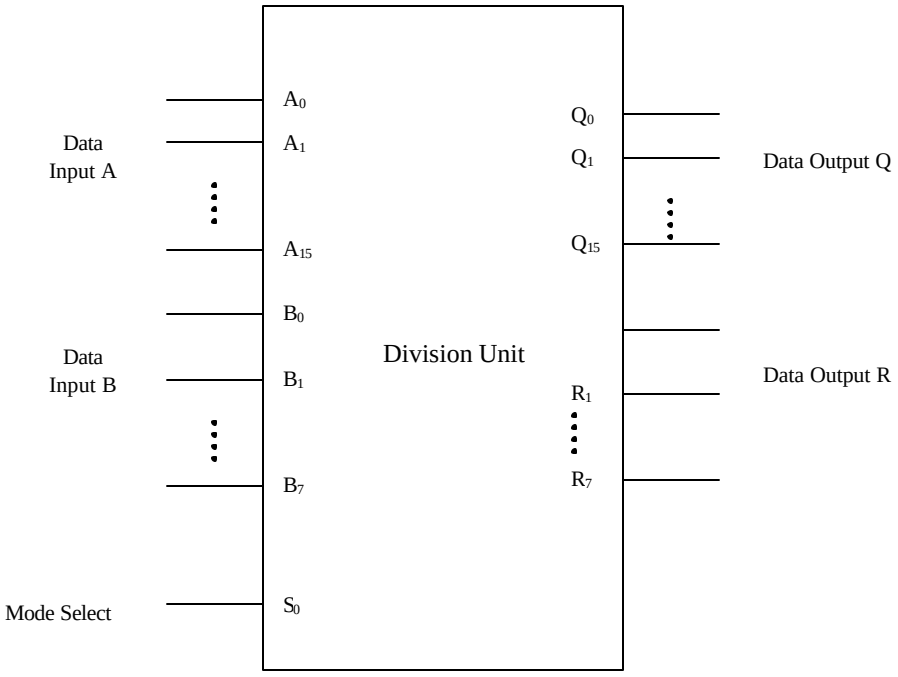

Figure 9 Division Unit

3.3 FLOATING POINT UNIT

3.3.1 IEEE 754 STANDARD

IEEE 754 Standard for floating point representation comprises a 23-bit mantissa M, an 8-bit exponent field E and a sign bit S. Floating point representation is redundant because the same number can be represented in more than one way. So, we have to represent in a unique or normal form. This process is called Normalization. The mantissa is said to be normalized if the digit to the right of the radix point is not zero, that is, no leading zeros appear in the magnitude part of the number. Hence the magnitude part of the normalized mantissa is always ‘1’. So there is no need to actually store the bit and it can be added later in the arithmetic circuits that process these numbers.

The same representation must be used for zero both in fixed and floating point formats, which facilitates the implementation of the instructions that test for zero. So, the floating point exponents should be encoded in excess-K code similar to excess-3 code, where the exponent field E contains an integer that is the desired exponent value plus K. The quantity K is called the bias and an exponent encoded in this way is called biased exponent. In IEEE 754 Standard bias K is taken as 127. The number N represented by a 32-bit IEEE Standard floating point number has the following set of interpretations:

1. If $0 < E < 255$, then $N= (-1)^S * 2^{E-127} * (1.M)$
2. If $E = 0$ and $M != 0$, then $N =(-1)^S * 2^{E-126} * (0.M)$
3. If $E = 0$ and $M = 0$, then $N = (-1)^S*0$

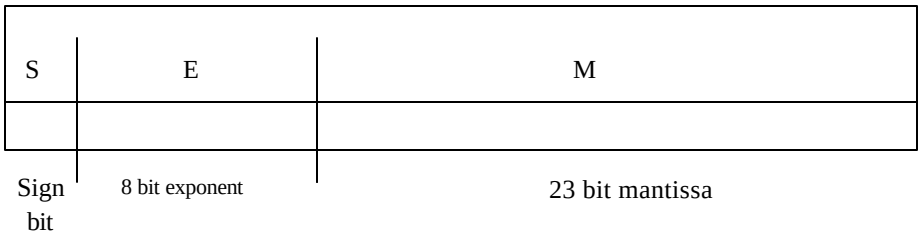

Figure 10 IEEE 754 Format

3.3.2 FLOATING POINT ADDITION AND SUBTRACTION

For the addition and subtraction of two floating point numbers, their exponents must be made equal before the corresponding mantissa can be added or subtracted. This exponent equalization is done by right shifting the mantissa X_M associated with the smaller exponent X_E a total of Y_E-X_E digits to form the new mantissa $X_M * 2^{(X_E - Y_E)}$ which can be combined directly with Y_M.

Floating point addition and subtraction has four steps:

1. Compute Y_E-X_E, a fixed point of subtraction.
2. Shift X_M by Y_E-X_E laces to the right to form $X_M * 2^{(X_E - Y_E)}$.
3. Compute $X_M * 2^{XE-YE} \pm Y_M$, a fixed point addition or subtraction.
4. Normalize the result

3.3.3 FLOATING POINT MULTIPLICATION AND DIVISION

Floating point multiplication involves a fixed-point multiplication of the mantissas and a fixed-point addition of their exponents. The multiplication is done using Modified Booth's Algorithm.

$$X * Y = (Xm * Ym) * 2^{Xe+Ye}$$

Floating point division involves a fixed-point division of the mantissas and a fixed-point subtraction of their exponents. The division is done using Non-Restoring Algorithm.

$$X / Y = (Xm / Ym) * 2^{Xe-Ye}$$

3.4 FFT IMPLEMENTATION

For recent wireless systems requiring 54Mbps data rate and throughput increase will need parallel architecture. It means more than one PE needs to be assigned per column to the FFT. Parallel-pipelined FFTs are suitable for both high throughput and high power efficiency. In parallel-pipelined architectures, only hardware cost for PEs will be increased, the actual size of the FIFOs between stages usually remains constant for a given FFT sized *N*. With the increase of the FFT size, the area of FFT processor will be dominated by the FIFOs. Hence, parallel pipelined FFTs have not significant area overhead, compared to pipelined FFT [8] [9]. For a given throughput, parallel-pipelined FFTs can operate at lower frequency than pipelined FFTs, therefore resulting in lower power consumption. Not many researchers have explored the scope of parallel-pipelined FFTs.

For 64-point pipelined FFT, we employ four radix-4 PEs in each stage. The block diagram of the architecture is depicted in Figure 11 termed as 4-parallel-pipelined FFT. The architecture can achieve four times throughput, compared to the pipelined FFT. The input data are separated into four streams. There are four commutators in stage1 and stage2, respectively. Each of them has 1/4 size of the commutators in pipelined FFTs and implements the transform from sequential data to parallel data. The coefficients in each stage are divided into four sections, responding to four input streams. Only 3 complex multipliers are used in stage2, because the output of butterfly1 in stage2 is multiplied by stage 2's coefficient1, which only contains (7fff,0000). In this architecture, the number of PEs per column is same as the radix, hence, no

shuffle units is needed in stage3. The multipliers' outputs in stage2 are fed into each of 4 simplified butterfly elements as shown in Figure 11.

The simulated a multiplier-less architecture to replace the conventional complex multiplier in pipelined FFTs [10]. Both area and power consumption for the multiplier unit are reduced. This paper explores a parallel-pipelined architecture with 4 radix-4 PEs in a column for 64-point FFT and a parallel-pipelined architecture with 2 radix-4 PEs in a column for 16-point FFT. The complex multiplication in parallel-pipelined FFTs is replaced by the minimum number of shift and addition operations based on common sub expression sharing across coefficients.

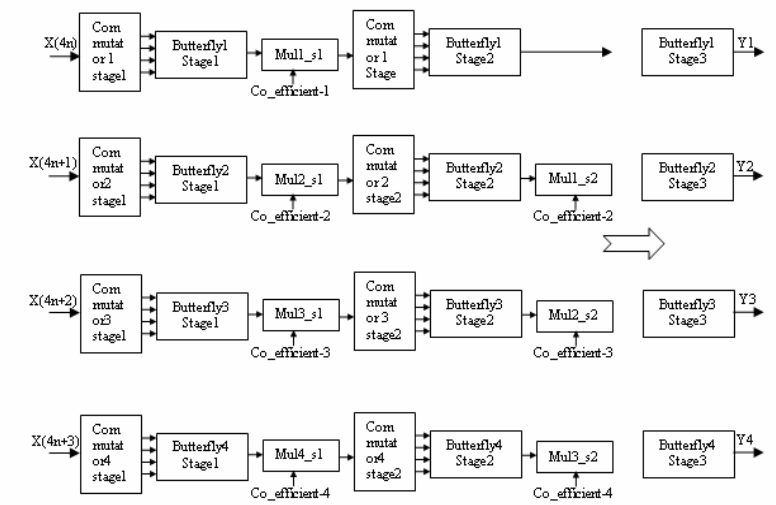

Figure 11 Block diagram of Pipeline FFT

4 HYBRID DVS ALGORITHM

The efficient algorithms for interDVS and intraDVS are taken and they are implemented in hardware description language [12]. Depending upon the task and slack time a particular algorithm is selected and it will adjust either processor voltage or clock. The block diagram is shown below and the selection of the algorithm is done as per the table

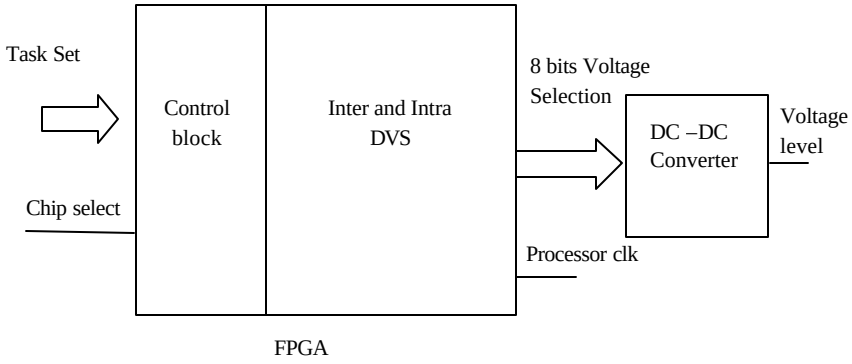

Figure 12 Block diagram of Hybrid DVS

Heiristic	Description
S_0	Uses only intra mode since only one task is available
S_1	Uses the intra mode when the unused slack time is more than a predefined amount of slack time
S_2	Uses inter mode since the number of task is more than 1

Table 2 Selection of DVS algorithm

5 INSTRUCTIONS - PREFETCH MODULE

The memory-prefetch module fetches the 128 bit VLIW instructions from the memory and places it in the prefetch queue of size 6. This enables increasing the speed of the processor through pipelining (simultaneous execution and fetching of subsequent instructions). The module passes the VLIW from the queue to the decoder module.

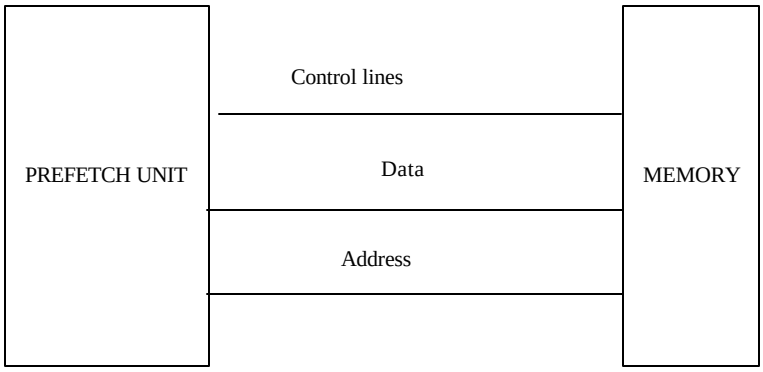

Figure 13 Memory - Prefetch Module

5.1 DECODER MODULE

The decoder module decodes the VLIW passed by the memory pre fetch module and generates the control signals and register addresses for all the seven instructions forming the VLIW.

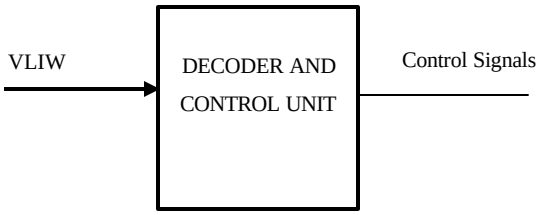

Figure 14 Decoder Module

5.2 FUNCTIONAL UNIT-REGISTER MODULE

The module uses the controls signals and register addresses from decoder module. It accesses the registers for data and passes them to functional units to compute the result. The result is then stored back into the register.

The functional unit-register module consists of

- Functional units
  - One floating point addition/subtraction unit
  - One floating point multiplication/division unit
  - Two fixed point ALU
  - One fixed point multiplication/division unit
  - One load/store unit
  - 64 point FFT
- Register set
  - Fixed-point register set
  - Floating-point register set.

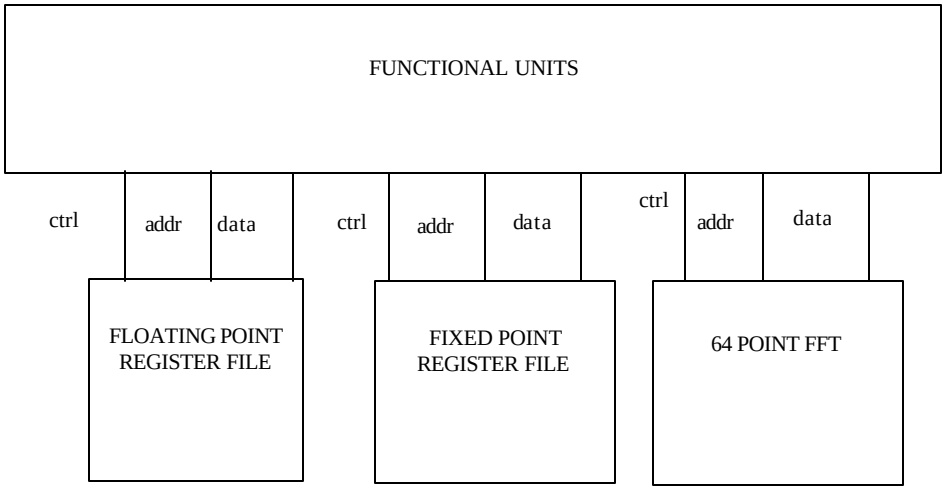

Figure 15 Functional Unit – Register Module

6 SIMULATION RESULT

The VLIW processor functional blocks are written in Hardware description language and this code is simulated and then synthesized using Xilinx project navigator. The results are shown below and the device utilization of the FPGA is given below

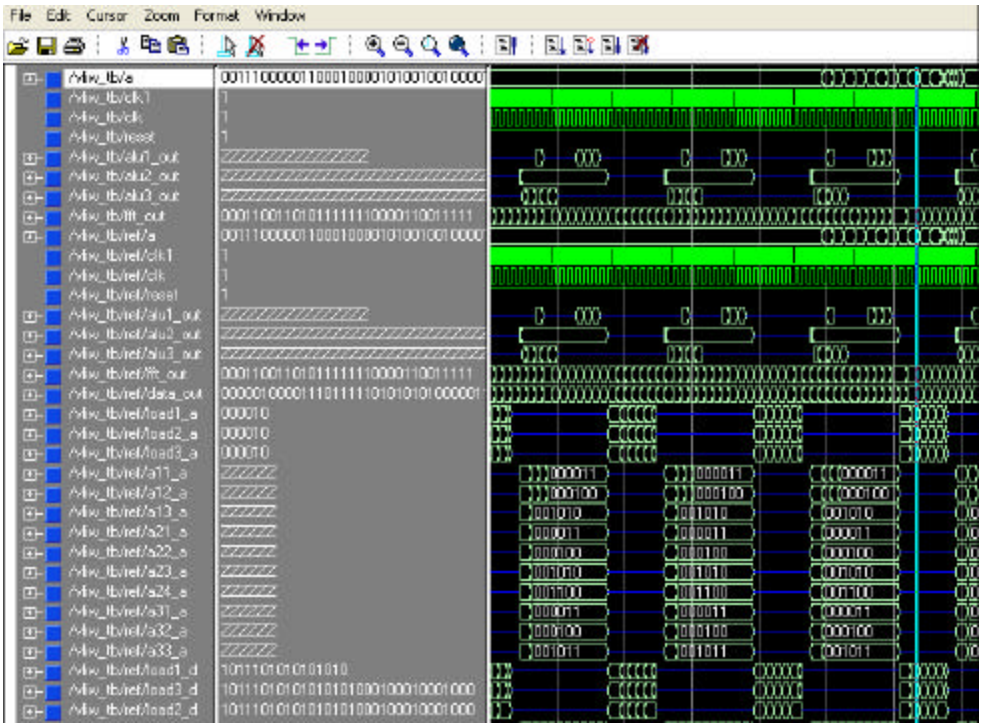

Figure 16 Simulation result of VLIW processor

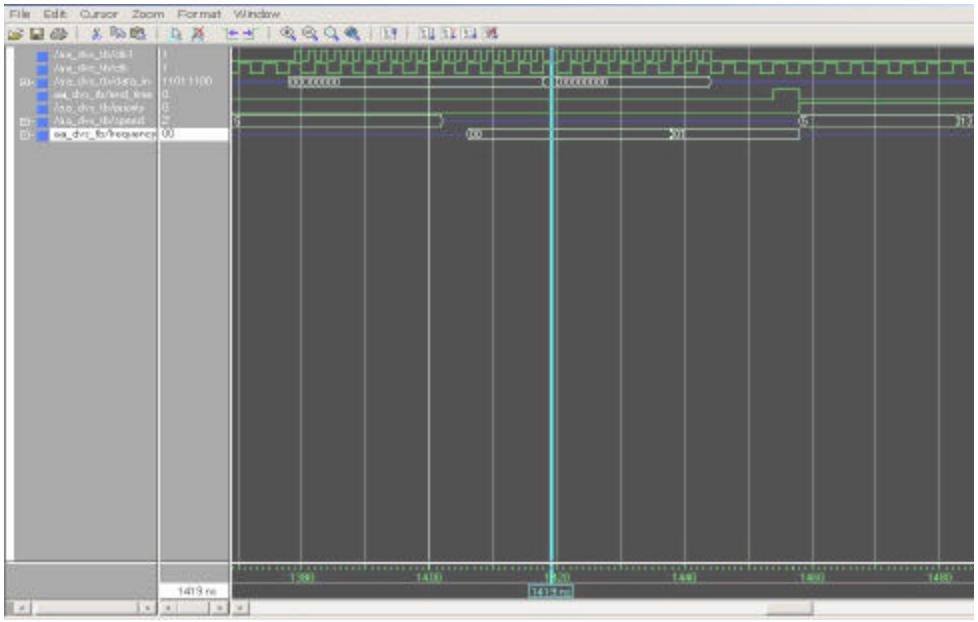

Figure 17 Hybrid DVS simulation result

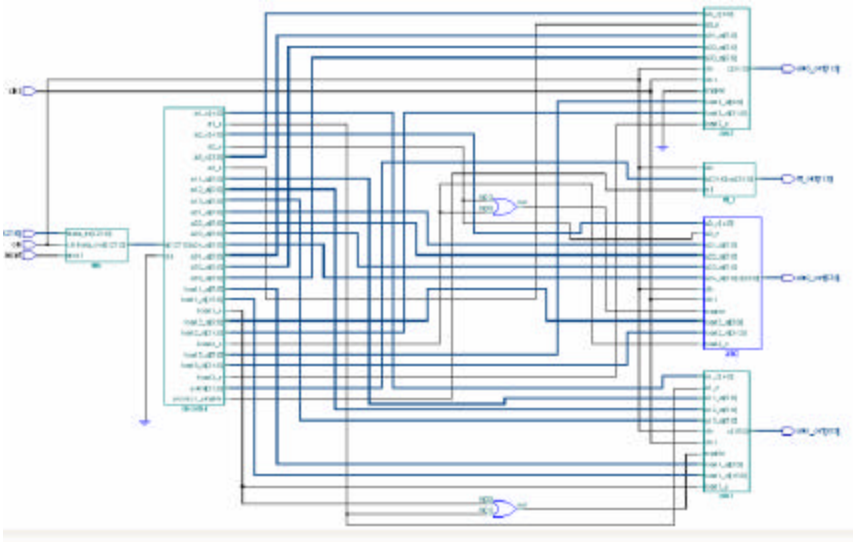

Figure 18 Synthesis report showing the various blocks in VLIW processor

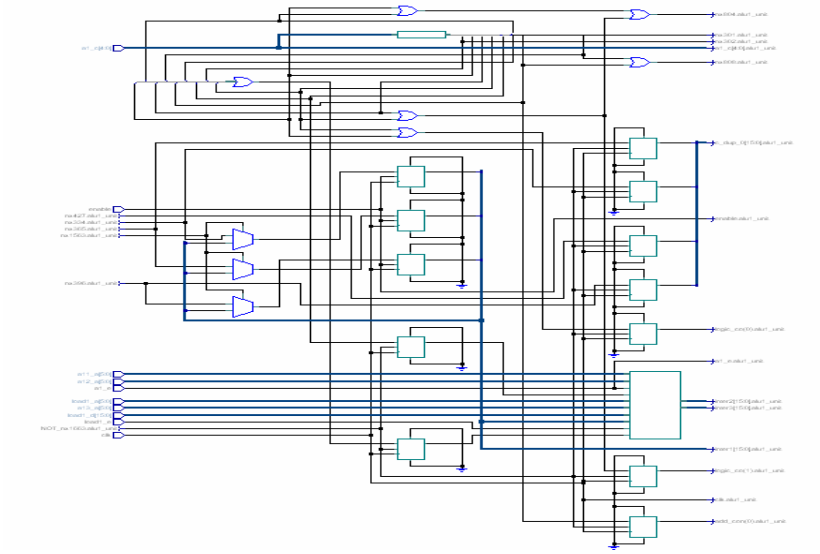

Figure 19 Synthesis report of VLIW processor

Device Utilization for 2VP125ff1696

****

Resource	Used	Avail	Utilization
IOs	275	1200	22.92%
Function Generators	24287	111232	21.83%
CLB Slices	12144	55616	21.84%
Dffs or Latches	5965	114832	5.19%
Block RAMs	0	556	0.00%
Block Multipliers	0	556	0.00%

Table 3 Device utilization for VLIW processor

Device Utilization for 2V1000bg575

****

Resource	Used	Avail	Utilization
IOs	22	328	6.71%
Function Generators	4840	10240	47.27%
CLB Slices	2420	5120	47.27%
Dffs or Latches	1185	11224	10.56%

Table 4 Device utilization for Hybrid DVS

Conclusion

This paper described the implementation of a low power dynamic voltage scaling (DVS) VLIW processor. Dynamic voltage scaling allows our processor to operate at maximal effic iency without limiting peak performance. The future work will be implementing this architecture in ASIC to further increase the operating frequency and further reduction in power. This design can be implemented as asynchronous VLIW processor for getting better power consumption.

References

[1] W. Kim, J. Kim, and S. L. Min. A Dynamic Voltage Scaling Algorithm for Dynamic-Priority Hard Real-Time Systems Using Slack Time Analysis. In Proceedings of Design, Automation and Test in Europe (DATE'02), pages 788–794, March 2002.

[2] W.-S. Liu. Real-Time Systems. Prentice Hall, Englewood Cliffs, NJ, June 2000.

[3] T. Pering and R. Brodersen. Energy Efficient Voltage Scheduling for Real-Time Operating Systems. In Proceedings of the 4th IEEE Real-Time Technology and Applications Symposium, Work in Progress Session, June 1998.

[4] D. Shin, J. Kim, and S. Lee. Intra-Task Voltage Scheduling for Low-Energy Hard Real-Time Applications. IEEE Design and Test of Computers, 18(2):20–30, March 2001.

[5] D. Shin,W. Kim, J. Jeon, J. Kim, and S. L. Min. SimDVS: An Integrated Simulation Environment for Performance Evaluation of Dynamic Voltage Scaling Algorithms. In Proceedings of Workshop on Power-Aware Computer Systems (PACS 2002), February 2002.

[6] Y. Shin, K. Choi, and T. Sakurai. Power Optimization of Real-Time Embedded Systems on Variable Speed Processors. In Proceedings of the International Conference on Computer-Aided Design, pages 365–368, November 2000.

[7] R. Seshasayanan and Dr S.K.Srivatsa "DYNAMICALLY RECONFIGURABLE ARCHITECTURE FOR HYBRID DYNAMIC VOLTAGE SCALING"IEEE International conference held in Srilanka, August 2006

[8] S. Hong, S. Kim, M. C. Papefthymiou, and W. E. Stark, "Power-complexity analysis of pipelined vlsi fft architectures for low energy wireless communication applications," in *Circuits and Systems, 1999. 42nd Midwest Symposium* , Aug. 1999, vol. 1, pp. 8–11.

[9] Steve F. Gorman and Jeffrey M. Wills, "Partial column fft pipelines," *IEEE Transactions on circuits and systems*, vol. 42, no. 6, June 1995.

[10] *Wei Han, T. Arslan, A. T. Erdogan and M. Hasan* "Multiplier-less based parallel-pipelined fft architectures for wireless communication applications" IEEE proceedings 2005

[11] D. Houzet, "Design of risc/vliw processor", Course work ,sept. 1995.

[12] Robert P. Colwell, Robert P. Nix, John ***J.*** *O'donnell, David B. Papworth, and Paul k. Rodman,* ***"****A VLIW Architecture for a Trace Scheduling Compiler" IEEE Transactions on computers, Vol.* ***37,*** *No. 8, August 1988.*

[13] Christian Iseli and Eduardo Sanchez "A Reconfigurable VLIW Processor using FPGAs" IEEE International Conference, 1993.

[14] B.Ramakrishna Rau, HP Laboratories "Dynamically Scheduled VLIW Processors", IEEE International Conference, 1993.

[15] Jean Paul Theis and Lothar Thiele "VLIW Processors under Periodic Real Time Constraints", IEEE International Conference, 1996.

[16] Jan Hoogerbrugge,Philips Research Laboratories, "Dynamic Branch Prediction for a VLIW Processor", IEEE International Conference, 2000.

[17]N. Bus& A. van der Werf, and M. Bekooij. Scheduling Coarse Grain Operations for VLI W processors. Madrid, Spain, 2000. ISSS.

[18] Yuki K, Shinsuke K, Koji O and Masaharu Imai " Synthesizable HDL Generation Method for Configurable VLIW Processors", IEEE International Conference, 2004..