Energy Efficient Multiple Clock Domain Array Processors by the Adaptive Tuning of Processor Clock Frequency

ABSTRACT
This paper investigates techniques which adaptively tune the processor frequency in a multiple-clock-domain array processor to achieve energy efficiency. The investigated clock tuning techniques include static clock configuration, runtime clock programming, and FIFO-based dynamic clock control. Several DSP applications are implemented on an array processor using different clock control methods as case studies. Simulation results show that, combined with optimistic voltage scaling methods, an average of about 40% power can be saved using adaptive clock tuning techniques, without degrading system performance.

Categories and Subject Descriptors
C.1.4 [**Parallel Architectures**]: Distributed architecture.

General Terms
Design

Keywords
Energy and Power Efficiency, Array Processor, Multiple Clock Domains, Adaptive Clock Frequency Tuning.

1. INTRODUCTION
Power and energy efficiency has become one of the most important parameters of the VLSI system and will be one of the key bottlenecks in future designs. Portable systems need low power implementations to increase battery life. Traditional non-portable systems are also concerned with power because their power consumption and power density are increasing dramatically [1]. Until now, the dynamic power has dominated the total power consumption which can be expressed as $P = aCV^2f$ [2]. Here a is the circuit state transition probability, C is the capacitance, V is the supply voltage, and f is the clock frequency. Recently, leakage is becoming another important power consumption source [3].

An array processor is a promising architecture which has the potential to provide a good balance of performance and energy consumption. Its basic idea is putting multiple processors into one chip to increase its parallelism and provide high performance. Research in array processor was active more than twenty years ago[4], but it has never been used widely commercially. Currently, increasing the clock frequency meets its bottleneck because of the design difficulty and huge power consumption [5]. On the other hand, submicron technology enables putting millions or even billions [9] of transistors into a single chip. Multiple-processor-chip shows a promising future. Not only much academic research [6 - 8], but also some commercial products such as the 2-core Itanium processor [9] and 8-core CELL processor [10] are appearing. An array processor also can provide energy efficiency because its parallel computing improves performance and gives the potential to decrease clock frequency and/or voltage.

Each processor in the array can use a global clock as in a systolic system [4], or use its own local clock as in a wavefront system [4]. Traditional systems use a single global synchronous clock because it is easy to design. As chip size and clock frequency increase dramatically, clock design has become one of the most difficult issues. At the same time, the high speed global clock consumes a huge amount of power. Globally asynchronous locally synchronous systems have become an active research topic, which potentially can provide high speed and low power.

Adaptively tuning the clock frequency and voltage is popularly used to get power and energy savings [11 - 17]. Most work focuses on a single-processor single-clock-domain system [11 – 14]. T. Kuroda et al. [11] created an optimal supply voltage generation scheme for specific fixed frequency at power-up time. T. Burd et al. [12] and K. Nowka et al. [13] use software to control the processor frequency and voltage at the runtime. S. Akui et al. [14] use hardware to control the processor frequency and voltage automatically at the runtime. G. Magklis et al. [15] analyzed the performance and energy efficiency for single-processor multiple-clock-domain systems. T. Fujiyoshi et al. [16] built one chip using adaptive frequency and voltage techniques with functional modules. The latest Intel 2-core Itanium [17] uses hardware to automatically control processor voltage and frequency by monitoring system power and temperature, but it does not maintain the same performance when decreasing the clock and voltage.

Little detailed analysis is available for chips using a multiple-processor multiple-adaptive-clock-domain design style. Very few multiple-processor chips use hardware to automatically change the processor clock and voltage, while maintaining the same performance. We use adaptive clock frequency tuning techniques as a low power solution for the multiple-clock-domain array processor. In particular, our proposed FIFO based dynamic clock control method can automatically tune the clock frequency according to the computation load, without degrading system performance.

2. IMPLEMENTATION OF AN ARRAY PROCESSOR

2.1 Processor Architecture

An asynchronous array of simple processors [18] containing uniform processor units is used as the example platform in this paper, but these methods are widely suitable to any multiple-clock-domain array processor. Fig. 1 shows a chip diagram with 3×3 processors. The hardware related to the multiple-clock system are the local configurable oscillator and the 32-word depth asynchronous dual-clock FIFO [19].

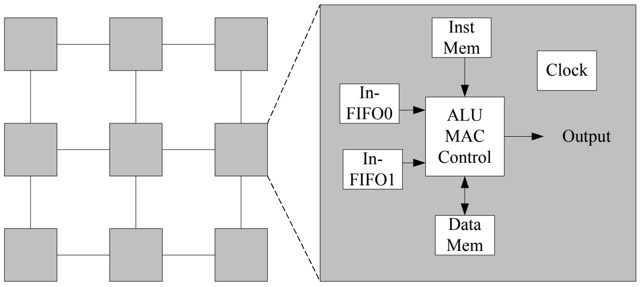


Fig. 1: Diagram of array processor with 3×3 processors

The main characteristics of our array processor include: small memory (64 32-bit word instruction memory and 128 16-bit word data memory), globally asynchronous locally synchronous system (GALS), and simple architecture and simple instruction set.

2.2 Mapping Applications on Array Processor

An 8-point DCT, an 8×8 DCT, a 64 point FFT and a JPEG encoder are implemented on the array processor to show some application mapping.

Eq. 1 is the 1-dimensional DCT algorithm and Eq. 2 is one efficient implementation [20] where $C_i = \cos\frac{i\pi}{16}$. Fig. 2 shows using two processors to do an 8-point DCT. The first processor deals with addition and subtraction, and the second processor deals with multiplication. The execution time (throughput) is 410ns when using a 100MHz clock frequency for both processors.

$$X(n) = \sum_{m=0}^{N-1} x(m)\cos(\frac{(2m+1)np}{2N}) \quad (1)$$

$$\begin{bmatrix} X(0) \\ X(2) \\ X(4) \\ X(6) \end{bmatrix} = \frac{1}{2}\begin{bmatrix} C4 & C4 & C4 & C4 \\ C2 & C6 & -C6 & -C2 \\ C4 & -C4 & -C4 & C4 \\ C6 & -C2 & C2 & -C6 \end{bmatrix}\begin{bmatrix} x(0)+x(7) \\ x(1)+x(6) \\ x(2)+x(5) \\ x(3)+x(4) \end{bmatrix} \quad (2a)$$

$$\begin{bmatrix} X(1) \\ X(3) \\ X(5) \\ X(7) \end{bmatrix} = \frac{1}{2}\begin{bmatrix} C1 & C3 & C5 & C7 \\ C3 & -C7 & -C1 & -C5 \\ C5 & -C1 & C7 & C3 \\ C7 & -C5 & C3 & -C1 \end{bmatrix}\begin{bmatrix} x(0)-x(7) \\ x(1)-x(6) \\ x(2)-x(5) \\ x(3)-x(4) \end{bmatrix} \quad (2b)$$

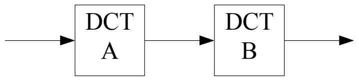


Fig. 2: 8-point DCT implementation using two processors

Eq. 3 shows the 2-dimensional DCT algorithm where $\alpha(0) = \frac{1}{\sqrt{N}}$ and $\alpha(m) = \sqrt{\frac{2}{N}}$ for $1 \le m \le N$. It can be processed using two 1-dimensional DCT steps as in Eq. 4. Fig. 3 shows using four processors to do an 8×8 DCT. The 1st and 3rd processors do an 8-point DCT, the 2nd and 4th processors do data transitions between rows and columns. The execution time (throughput) is 5060 ns when using a 100MHz clock frequency for all processors.

$$G_c(m,n) = \alpha(m)\alpha(n)\sum_{i=0}^{N-1}\sum_{k=0}^{N-1} g(i,k)\cos[\frac{\pi(2i+1)m}{2N}]\cos[\frac{\pi(2k+1)n}{2N}] \quad (3)$$

$$G_c(m,n) = \alpha(m)\sum_{i=0}^{N-1}[\alpha(n)\sum_{k=0}^{N-1} g(i,k)\cos\frac{\pi(2k+1)n}{2N}]\cos\frac{\pi(2i+1)m}{2N} \quad (4)$$

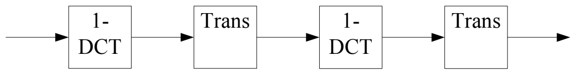


Fig. 3: 8×8 DCT implementation using four processors

A 64-point complex FFT algorithm is implemented using 8 processors [21] shown in Fig. 4. Here the *bit-reverse* processor transposes the order of input data, *butterfly processors* do FFT butterfly calculations, *memory processors* store intermediate data, and the *shuffle processor* reorders the output data. The execution time (throughput) is 117us when all processors use 100MHz.

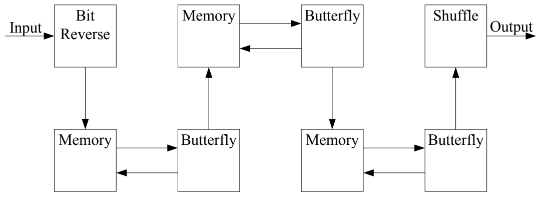


Fig. 4: 64 point FFT Implementation using eight processors

A JPEG encoder is implemented using 9 processors as shown in Fig. 5. The main functional blocks include level shifter, an 8×8 DCT, quantization, Zig-Zag reordering, and a varying length Huffman encoder. The execution time (throughput) is 14us per 8×8 block when using 100MHz clock frequency for all processors.

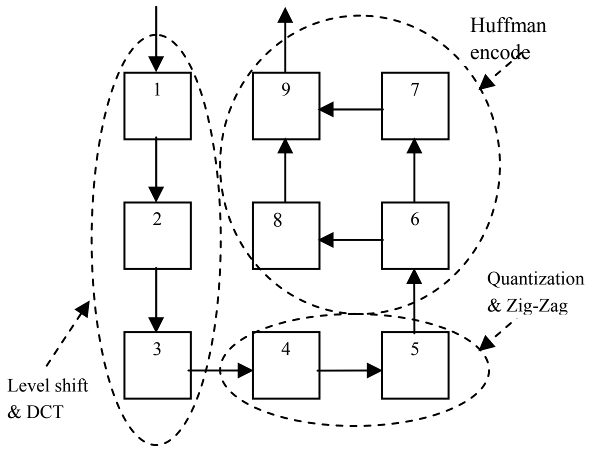


Fig. 5: JPEG encode implementation using nine processors

3. ADAPTIVE MULTIPLE CLOCK LOW POWER DESIGN

Three adaptive clock tuning techniques are investigated. The static clock configuration method configures the clock frequency at configuration time according to processor computation load and processor position in the array. The runtime clock programming method uses software to program the clock frequency at runtime according to sub-algorithm characteristics. The FIFO based dynamic clock control method uses hardware to change the clock frequency automatically according to FIFO fill information.

3.1 Static Clock Configuration

Statically deciding the processor clock frequency according to its computation load at power-up time and then adaptively tuning the processor voltage is the first adaptive clock/voltage method [11].

In the processor array, the computation and characteristics of each processor is different. Some processors can be configured using a slow clock frequency at configuration time, without degrading system performance. Not like in the single processor chip where its clock frequency only depends on its computation load, the optimal processor clock frequency in an array processor also depends on its position and relationship with other processors.

3.1.1 Relating optimal clock to computational load

The basic idea is using a slow clock frequency for processors with light computational load.

When using two processors to do an 8-point DCT as in Fig. 2, Fig. 6 shows that the system throughput changes with the scaling of the processor clock frequency. The throughput stays the same for a long period of time when scaling down the clock of the 1st processor. Minimal power consumption is achieved when choosing 46MHz and 100MHz for the two processors, with the same performance as using 100MHz for the two processors. Here the optimal processor clock relationship is exactly the same as the processor computation load relationship: 46MHz/100MHz = 19 cycles / 41 cycles, as shown in Fig. 7.

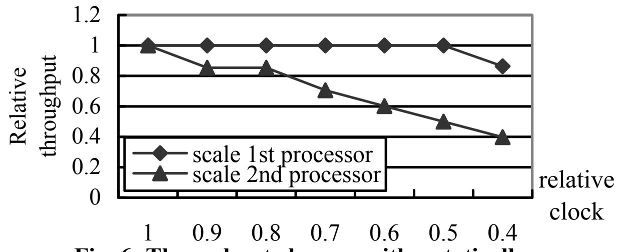
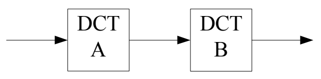


Fig. 6: Throughput changes with a statically configured processor clock for an 8-point DCT



computation load: 19 cycles : 41 cycles = 0.46:1
optimal clock: 46 MHz : 100 MHz = 0.46:1

Fig. 7: Optimal clock frequency ratio is the same as computational load in 8-point DCT

3.1.2 Relating optimal clock to processor position

The optimal processor clock frequency is also related to its position in the array processor.

When using four processors to do an 8×8 DCT as shown in Fig.3, Fig. 8 shows that the system throughput changes with the scaling of the processor clock frequency. The throughput changes with the scaling of the 2nd and 4th processor much slower than the scaling of 1st and 3rd processor, which shows the effect of computation load. In addition, although the 2nd and 4th processor have the same light computation load, their behavior is very different and the throughput changes with the 4th processor scaling much slower than the 2nd processor. Minimal power consumption is achieved when choosing 100MHz, 95MHz, 100MHz, 57MHz for four processors, with the same performance as with 100MHz for all processors. So here the optimal processor clock frequency not only depends on the processor computation load, but also depends on its position, as shown in Fig. 9 again.

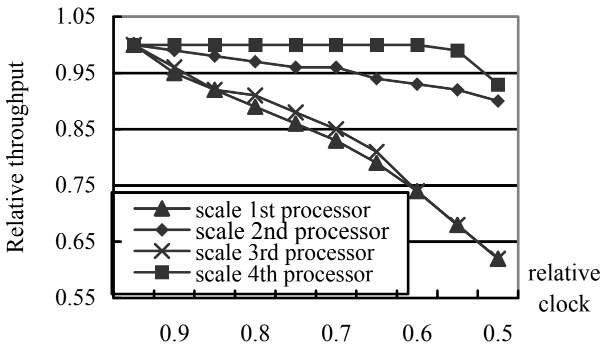
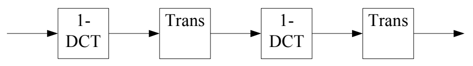


Fig. 8: Throughput changes with the statically configured processor clock for an 8×8 DCT



computation load (clock cycles): 408 : 204 : 408 : 204
optimal clock frequency (MHz): 100 : 95 : 100 : 57

Fig. 9: Optimal clock frequency is related to processor position in an 8×8 DCT

3.1.3 Combining with static voltage configuration

The processor voltage can be statically configured according to the static clock frequency to get further power saving. We simply assume that voltage is decreased linearly with the clock frequency.

3.2 Runtime Clock Programming

In the static configuration method, the processor frequency is decided at configuration time and will not be changed at run time. Looking into the algorithm of one processor, normally we can find different characteristics varying with the computing time. More energy can be saved if we use software to change the clock frequency at runtime for different parts of the program. Software based runtime clock programming is already commonly used in the single processor chip [12, 13, 16].

In the array processor, the characteristics of each processor should be checked and different methods should be applied to each of them.

3.2.1 *Characteristics of sub-algorithm*

In the 8×8 DCT implementation as shown in Fig. 4, each processor has a different behavior during executing time. Fig. 10 shows the stall probability of the 1st processor and 2nd processor along with time. The 1st processor has low stall probability most of the time, so we can not scale its frequency down without degrading system performance. The 2nd processor has high stall probability most of the time. Further more, an obvious difference exists between the first half and second half of execution. It always has high stall probability during the first half time and has a little less stall probability during the second half. This reality allows us to lower clock frequency for the first part of the program.

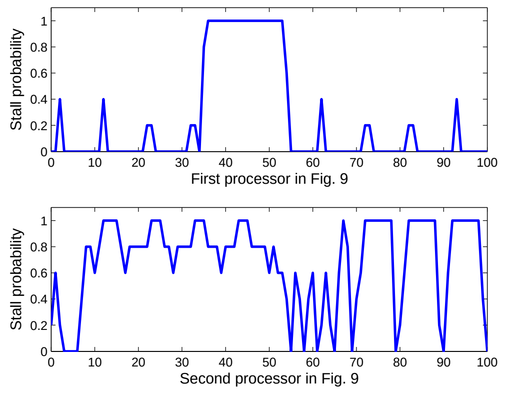


Fig. 10: Stall probability of (a) the 1st processor and (b) the 2nd processor changes along with computing time in 8×8 DCT

We can also see the behavior of the 2nd processor from table 1. Here the stall-input is the stall when reading empty FIFOs. The stall-output is the stall when writing full FIFOs. The first part reads the input data, and the second part reorders the data then outputs the result. The *reading* sub-algorithm is much simpler than the *reordering* algorithm, but their total time is similar because the *reading* sub-algorithm wastes many cycles on the input-stall.

Table 1: Sub-algorithm characteristics for 2nd processor

Sub -algorithm	**Execution Time (# cycles)**	**Stall-input (#cycles)**	**Stall-output (#cycles)**	**Total Time (#cycles)**
Reading	73	191	0	264
Reordering	133	0	108	241

3.2.2 *Program clock frequency using instructions*

Fig. 11 shows an example pseudo assembly code with runtime clock programming for the 2nd processor in an 8×8 DCT. Two extra instructions are used to program the clock frequency.

Fig. 12 shows the system throughput for different runtime clock programming schemes for the 2nd processor. Minimal power consumption is achieved when choosing a clock frequency around 70MHz and 100MHz respectively for the two program parts, with the same performance as using a static 100MHz.

```
main:
MOVE start=0, end=63
MOVI stride = 1
MOVI osc   82   //              Programming Clk1
RPT #64
MOVE [addr+] Input

MOVI osc   50   //              Programming Clk2
MOV start=0, end=56
MOV stride = 8
RPT #8
MOVE output [addr+]
MOV start=1, end=57
RPT #8
MOVE output [addr+]
MOV start=2, end=58
RPT #8
……
```

Fig. 11: Assembly code for Transposing DCT processor with runtime clock programming

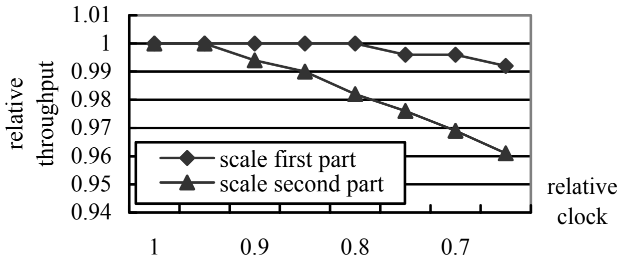


Fig. 12: Throughput changes with different runtime clock programming for an 8×8 DCT

3.2.3 *Combining with runtime voltage programming*

Runtime clock programming can be combined with runtime voltage programming to get further power saving. Using DC-DC converters to change the voltage is too slow. Another possible method called voltage dithering [22] is using a multiple voltage supply and choosing a low or high voltage using program instructions. Potentially the voltage can be changed almost simultaneously with the clock frequency.

3.3 FIFO based Dynamic Clock Control

Previously discussed methods both use software methods to decide the processor frequency. One of the difficulties in using these methods is that the programmer needs to calculate the optimal clock frequency complicating programming. Hardware based methods have been used to automatically and dynamically change the clock frequency at the runtime [14, 17]. Implementation in [14] only focuses on a single processor, while the 2-core Itanium [17] does not keep the same system performance when decreasing the clock and voltage.

In an array processor, different processors transfer the processed data through the FIFO, which give us a chance to monitor each processor's computational load according to the FIFO fill rate and automatically adjust the clock frequency for each processor at run time, and maintain system performance.

3.3.1 Hardware implementation

The basic function of the circuit is to increase the clock frequency when the input FIFO is too full (processing is too slow), and decreasing the clock frequency when the FIFO is too empty (processing is too fast).

Fig. 13 shows the circuit diagram which automatically tunes the oscillator frequency according to the FIFO fill rate. Four signals are extracted from the FIFO. Signal *total_full* indicates the FIFO is full and no further data can be written in, signal *too_full* indicates the FIFO is nearing full (we define *too_full* is 1 when FIFO is filled with more than 16 data units), signal *total_empty* indicates the FIFO is empty, signal *too_empty* indicates the FIFO is nearing empty (we define *too_empty* is 1 when FIFO is filled with less than 8 data units). The reason to separate signals *total_full* and *total_empty* from *too_full* and *too_empty* is because they should have higher priority.

Signals *too_full* and *too_empty* are fed into counters (we choose 4-bit counter). When the FIFO is near full for several consecutive cycles (16 in our case), or is totally full, the clock frequency will be increased by 1.25. When the FIFO is nearing empty for several consecutive cycles (16 in our case), or is totally empty, the clock frequency will be decreased by 1.25. Otherwise the clock frequency keeps the same. The clock frequency range we choose is between 20MHz and 625MHz.

Different clock steps are evaluated as shown in Fig. 14. Here *2* means the clock frequency is multiplied or divided by 2 each time, 1.25 means the clock frequency is multiplied or divided by 1.25 each time. The 1.25 case performs a little better in these studied cases and is the reason why we change the clock 25% each time.

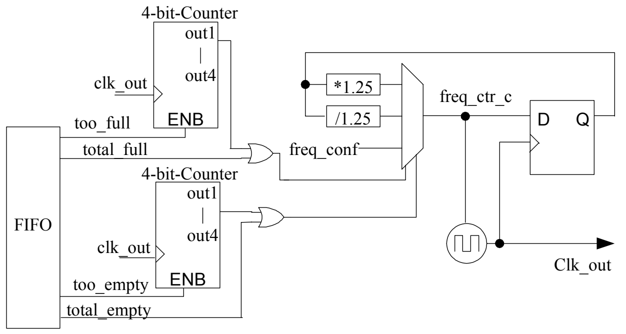


Fig. 13: FIFO based dynamic clock control circuit diagram

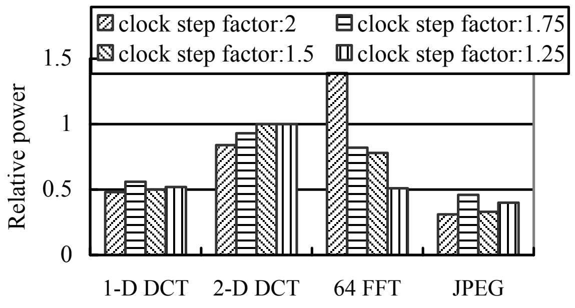


Fig. 14: Relative power of several applications using different clock change step

3.3.2 Self-adaptivity

The FIFO based dynamic clock control method has self-adaptivity characteristics which can be observed in two situations.

First, self-adaptivity exists in the FIFO fill status: the circuit will adjust the processing speed according to the FIFO fill status to avoid always being in the too full or too empty state. Fig. 15 shows the 4th processor's FIFO fill status in an 8×8 DCT before and after using the dynamic clock control method. The FIFO is always near empty if using a static 100MHz clock, while the FIFO is less near empty status after using dynamic clock control method.

Second, self-adaptivity exists in the clock speed: it will not only be in a very fast or very slow state. The clock speed is increased during heavy computation and decreased during light computation. Fig. 16 shows the clock frequency of the 4th processor in an 8×8 DCT with the dynamic clock control method. The average clock frequency becomes 50.8MHz, having the same performance as when using a static 100MHz clock frequency.

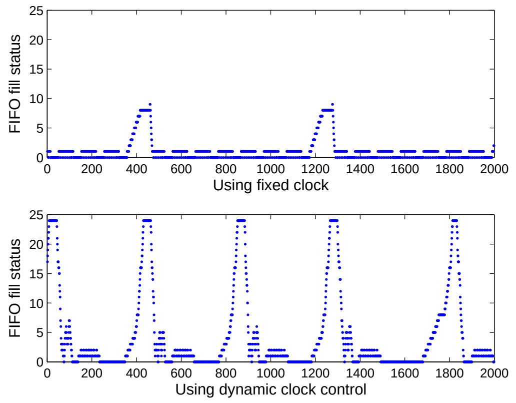


Fig. 15: Processor FIFO fill status before and after using the dynamic clock control method

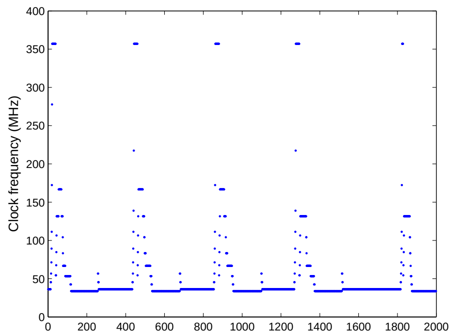


Fig. 16: Processor clock frequency changes with time when using the dynamic clock control method

3.3.3 Combining with voltage selection

Similar to the idea of multiple voltage selections in the runtime clock programming, we can select a high or low voltage together along with clock frequency, to get further power saving.

4. METHODS COMPARISON

Table 2 and Fig. 17 compare the power consumption using three adaptive clock control methods together with optimistic voltage scaling. Single processor power is assumed as 1 for normal speed.

Table 2: Power comparison of adaptive clock control methods

Application	Initial method	Static config	Runtime program	Dynamic control
8-pt. DCT	2	1.09	1.09	1.04
8×8 DCT	4	3.03	2.77	4
64 FFT	8	5.63	5.21	4.08
JPEG	9	2.59	2.43	3.6

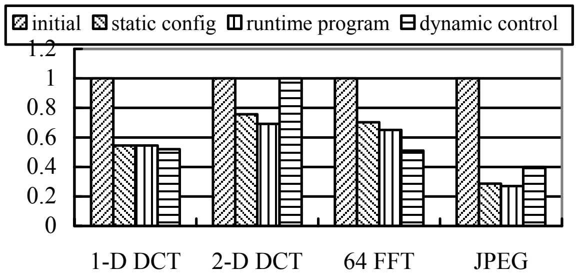


Fig. 17: Relative power of several applications using clock frequency control methods

Table 3 compares the characteristics of three clock frequency control methods. The power savings data comes from the average power saving of the example applications. The runtime programming method has a small advantage on the power savings. The static configuration method has simple hardware and not difficult programming. The advantage of the dynamic control is its easy programming and self-adaptivity.

Table 3: Characteristics of three clock control methods

Method	Hardware	Program	Power Saving	Others
Static configure	Simple	Middle	~43%	--
Runtime program	Middle	Complex	~46%	--
Dynamic control	Complex	Simple	~38%	Self-adaptivity

5. SUMMARY

This paper investigates techniques which adaptively tune the clock frequency of an array processor to get a low power implementation. Simple static clock configuration can achieve around 40% power saving without degrading system performance. Programming the frequency at the runtime can increase power savings with the cost of more difficult programming. Hardware based dynamic clock control method can automatically tune its clock frequency according to processor hardware state.

6. ACKNOWLEDGEMENTS

The authors acknowledge the other processor co-designers and support from Intel, NSF and a Faculty Research Grant.

7. REFERENCE

[1] S. Borkar, “Obeying Moore’s Law beyond 0.18 micron,” 13th IEEE International ASIC/SOC Conference, Sep. 2000, pp:26-31.

[2] A. P. Chandrakasna et al., “Low-Power CMOS Digital Design,” JSSC, April 1992, pp: 473-484.

[3] K. Roy et al., “Leakage Current Mechanisms and Leakage Reduction Techniques in Deep-Submicrometer CMOS Circuits,” IEEE Proceedings, Feb. 2003, pp:305-327.

[4] S. Y. Kung, “VLSI Array Processors,” IEEE ASSP Magazine, July 1985, pp: 4-22.

[5] K. Krewell, “Intel Cancels 4GHz P4, Focus Shifts to System Features,” www.MPRonline.com, Nov. 1, 2004.

[6] K. Mai et al., “Smart Memories: A Modular Reconfigurable Architecture,” ISCA, 2000, pp: 161-171.

[7] H. Zhang et al., “A 1-V Heterogeneous Reconfigurable DSP IC for Wireless Baseband Digital Signal Processing,” JSSC, Nov. 2000, pp: 1697-1704.

[8] M. B. Taylor et al., “The Raw Microprocessor: A Computational Fabric For Software Circuits And General-Purpose Programs,” IEEE Micro, 2002, pp. 25-35.

[9] S. Naffziger et al., “The Implementation of a 2-core Multi-Threaded Itanium Family Processor,” ISSCC, 2005, pp. 182-183.

[10] D. Pham et al., “The Design and Implementation of a First-Generation CELL Processor,” ISSCC, 2005, pp. 184-185.

[11] T. Kuroda et al., “Variable Supply-Voltage Scheme for Low-Power High-Speed CMOS Digital Design”, JSSC, Mar. 1998, pp: 454-462.

[12] T. Burd et al. “A Dynamic Voltage Scaled Microprocessor System,” ISSCC, 2000, pp: 294-295.

[13] K. Nowka et al., “A 0.9v to 1.95v Dynamic Voltage-Scalable and Frequency-Scalable 32b PowerPC Processor”, ISSCC, 2002, pp:340-341.

[14] S. Akui et al., “Dynamic Voltage and Frequency Management for a Low-Power Embedded Microprocessor,” ISSCC 2004, pp: 64-65.

[15] G. Magklis et al., “Dynamic Frequency and Voltage Scaling for A multiple-Clock-Domain Microprocessor,” IEEE Micro, 2003, pp: 62-68.

[16] T. Fujiyoshi et al., “An H.264/MPEG-4 Audio/Visual Codec LSI with Module-Wise Dynamic Voltage/Frequency Scaling,” ISSCC 2005, pp: 132-133.

[17] T. Fischer et al., “A 90nm Variable-Frequency Clock System for a Power-Managed Itanium-Family Processor,” ISSCC, 2005, pp: 294-295.

[18] Omitted for reviewer

[19] Omitted for reviewer

[20] K. K. Parhi, “VLSI Digital Signal Processing Systems”, John Wiley & Sons, 1999, pp275-285.

[21] Omitted for reviewer

[22] B. H. Calhoun et al., “Ultra-Dynamic Voltage Scaling Using Sub-threshould Operation and Local Voltage Dithering in 90nm CMOS,” ISSCC, 2005, pp:300-301