韓國컴퓨터情報學會 論文誌
第14卷 第11號, 2009. 11.

2009-14-11-3-2

Nios® II 임베디드 프로세서를 사용한 병렬처리 시스템의 설계 및 구현

이 시 현*

The Design and implementation of parallel processing system using the Nios® II embedded processor

SI-HYUN LEE*

요 약

본 논문에서는 시스템의 변경이 많고 적은 비용으로 고성능 데이터 처리가 요구되는 응용분야에서 시스템의 유연성, 가격, 크기 및 성능을 개선하기 위한 목적으로 알테라(Altera)의 Nios® II 임베디드 프로세서(embedded processor) 4개를 사용하여 주종(master-slave)과 공유메모리(shared memory) 구조를 가지는 병렬처리 시스템을 설계하고 구현하였다. 설계한 병렬처리 시스템은 Nios® II 32bit RISC 프로세서, SOPC® Builder, Quartus® II, ModelSim®으로 개발되었으며 설계한 병렬처리 시스템의 성능 평가는 Terasic®사의 DE2-70® 레퍼런스 보드(Cyclone® II(EP2C70F896C6N) FPGA)에서 검증하고 구현하였다. 설계한 병렬처리 시스템의 성능을 평가하기 위해서 1개, 2개, 4개의 프로세서로 512, 1,024, 2,048, 4,096, 8,192 N-point FFT(fast fourier transform) 연산을 수행하여 속도향상(Sp)과 시스템의 효율(Ep)을 평가하였다. 성능평가 결과 Sp는 1개의 프로세서를 사용한 경우에 비해서 2개의 프로세서를 사용한 경우 평균 1.8배, 4개의 프로세서를 사용한 경우에는 평균 2.4배의 속도향상을 보였다. 또한 Ep는 1개의 프로세서를 사용한 경우에는 1, 2개의 프로세서를 사용한 경우에는 평균 0.90, 4개의 프로세서를 사용한 경우에 평균 0.59를 보였다. 결과적으로 논문에서 구현된 병렬처리 시스템은 단일 프로세서를 사용하는 경우에 비해서 고성능 데이터 처리가 요구되는 분야에서 경제적인 시스템으로 구현할 수 있음을 보였다.

Abstract

In this thesis, we discuss the implementation of parallel processing system which is able to get a high degree of efficiency(size, cost, performance and flexibility) by using Nios® II(32bit RISC(Reduced Instruction Set Computer) processor) embedded processor in DE2-70® reference board. The designed parallel processing system is master-slave, shared memory and MIMD(Multiple Instruction-Multiple Data stream) architecture with 4-processor. For performance test of system, N-point FFT is used. The result is represented speed-up as follow; in the case of using 2-processor(core), speed-up is shown as average 1.8 times as 1-processor's. When 4-processor, the speed-up is shown as average 2.4 times as it's.

▸ Keyword : Parallel processing system, FPGA, SoC, RISC

• 제1저자 : 이시현
• 투고일 : 2009. 10. 23, 심사일 : 2009. 10. 29, 게재확정일 : 2009. 11. 26.
* 동서울대학 정보통신과 조교수

Ⅰ. 서 론

최근 가전, 통신 및 멀티미디어(multimedia) 응용 기기들은 점차 소형, 저가, 고성능 및 융합화(장치, 서비스) 추세에 있다. 이러한 시장 및 기술적인 추세에 따라 단일의 고성능 프로세서를 사용하는 대신 여러 개의 프로세서를 사용하여 경제성을 가질 수 있는 더 효율적인 시스템을 구성하려는 필요성과 요구가 증가되고 있다. 따라서 본 논문에서는 다양한 응용 분야에서 시스템의 시장 적기성(time-to-market), 융통성(flexibility), 처리속도 향상, 소형화 및 경제성을 갖는 시스템을 구현하기 위한 목적으로 Nios® II 32bit RISC(Reduced Instruction Set Computer) 임베디드 프로세서(embedded processor)를 사용하여 병렬처리 시스템을 구현하는 것이다. 연구결과는 시스템의 변경이 많고 적은 비용으로 고성능 데이터 처리가 요구되는 응용분야에서 단일 프로세서를 사용하는 시스템에 비해서 경제성을 갖는 시스템으로 구현할 수 있다.

Ⅱ. 병렬처리 시스템

2-1. 병렬처리 시스템의 분류

병렬처리시스템은 시간적 또는 공간적 동시성을 모두 포함하는 2개 이상의 프로세서로 처리하는 시스템으로 정의한다. Michael J.Flynn은 컴퓨터 구조를 명령어 흐름과 데이터 흐름에 따라 SISD(Single Instruction-Single Data stream), SIMD(Single Instruction-Multiple Data stream), MISD(Multiple Instruction-Single Data stream), MIMD(Multiple Instruction-Multiple Data stream)로 분류하였다.[1],[2] 병렬처리 시스템은 구조상의 특징에 따라 파이프라인 컴퓨터(pipeline computer), 어레이 프로세서(array processor), 멀티프로세서 시스템(multiprocessor system)으로 구분된다. 병렬처리 시스템은 주변장치와 프로세서간의 연결 구조에 따라 시분할공통버스, 크로스바 스위치 네트워크, 다중포트 메모리로 구분된다.[1],[2] Flynn의 방법의 MIMD 구조는 기억장치의 위치, 주소지정 방식 및 프로세서들에 의한 기억장치 액세스 시간에 따라 UMA(Uniform Memory Access), NUMA(Nonuniform Memory Access), COMA(Cach-Only Memory Access), NORMA(Non-Remote Memory Access)로 구분할 수 있다. 또한 최근에는 병렬컴퓨터의 고성능화를 위한 시스템 구성에 따라 SMP(Symmetric Multiprocessor), MPP(Massively Parallel Processor), CC-NUMA(Cache Coherence-NUMA), DS(Distribute System), CC(Cluster Computer)으로 구분한다.[3],[4] 병렬처리 시스템의 개발 목적은 1 개의 프로그램을 여러 개의 프로세서가 분담 또는 복수로 처리하게 하여 성능, 신뢰성 그리고 자원의 이용도를 향상하는데 있으며 오퍼레이팅 시스템의 구성 방법은 주종, Separate Supervisor, Floating Supervisor Control이 있다. 멀티프로세서 시스템은 Flynn의 분류 방법에 의해서 MIMD 구조이다.[1],[2]

2-2. 병렬처리 시스템의 소프트웨어 기술

병렬처리 시스템에서 소프트웨어는 시스템의 하드웨어들이 데이터를 효율적으로 처리하여 성능을 향상할 수 있는 매우 중요한 분야이다. 이러한 소프트웨어 기술에서 프로세서 동기화와 상호 배타 메카니즘이 효율적으로 구현되어야 한다. 또한 병렬처리의 효과를 높이기 위한 데이터 의존성을 최소화할 수 있는 최적의 문제분할 방법과 알고리즘의 병렬성을 높일 수 있는 연구가 매우 중요하다.[5],[6]

프로세서의 동기화와 상호배타 구현 방법에는 Bus-Locking 방식, Spin-Lock 방식, Semaphore 방식, Barrier Synchronization 방식, Fully-Empty Flag를 이용한 방식, Fetch-and-Add 명령어를 이용하는 방식 등이 제안되고 있다.[7],[8]

2-3. 병렬처리 시스템의 프로세서 스케줄링

병렬처리 시스템에서 N개의 프로세서들을 가진 병렬 컴퓨터의 성능(속도 향상)이 N배가 되지 못하는 한계성이 있다. 이것의 주요 원인은 병렬처리에 따른 오버헤드(over-head)와 낮은 시스템 효율(문제 분할, 시스템 의존성, 대기 시간, 프로세서간의 통신량 등)에 있다. 그래서 이러한 성능을 저하시키는 문제들을 개선하기 위해서는 다음 사항들을 고려하여 설계되어야한다. (1)프로세서들의 데이터 처리는 최대한 균등하게 할당되도록 한다. (2)프로세서들 간의 통신량이 최소화 되도록 한다. (3)데이터 의존성을 없애기 위해서 최대한 같은 프로세서 혹은 가장 가까운 프로세서에 할당되도록 한다. 프로세서 스케줄링(processor scheduling)은 태스크(task)에 대한 정보의 유무에 따라 결정적 스케줄링(deterministic scheduling)과 비결정적 스케줄링(non-deterministic scheduling)이 있다.[9],[10]

Ⅲ. Nios® II 임베디드 프로세서를 사용한 병렬처리 시스템의 설계

3-1. 병렬처리 시스템의 구성

그림 1은 Nios® II 임베디드 프로세서의 내부 구조와 인터페이스를 우한 버스(Avalon® bus) 구조이다.[11]

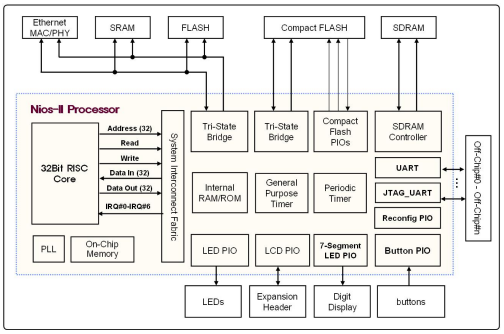

그림1. Nios® II 임베디드 프로세서의 구조
Fig.1. Architecture of Nios® II embedded processor.

3-2. Nios® II 임베디드 프로세서를 사용한 병렬처리 시스템의 설계

그림 2는 4개의 Nios® II 임베디드 프로세서를 사용한 병렬처리 시스템의 구조이다. 시스템은 주종 구조이며 CPU #1의 주 프로세서가 처리할 데이터를 종 프로세서(CPU #2, 3, 4)에 전송하고 처리하는 과정이다.

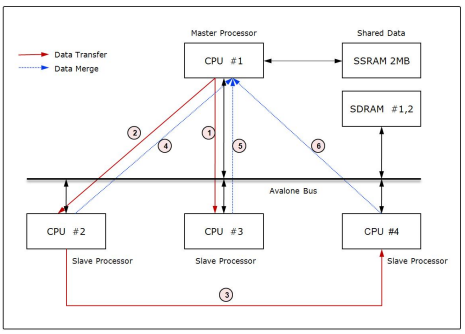

그림2. 설계한 병렬처리 시스템의 데이터 처리
Fig.2. Data processing flow of designed parallel processing system.

그림 3은 Nios® II 임베디드 프로세서를 사용하여 병렬처리 시스템을 설계(H/W, S/W)하는 환경이다.[11]

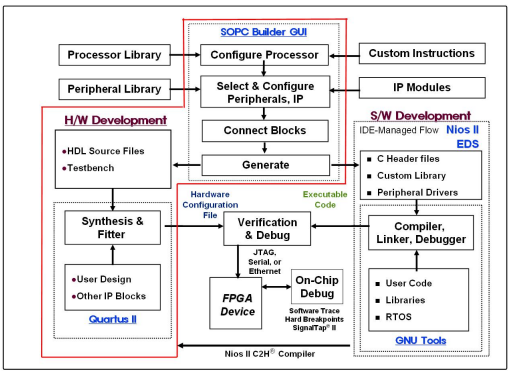

그림3. Nios® II를 사용한 병렬처리 시스템의 설계 환경
Fig.3. Design environment of parallel processing system using Nios® II Embedded Processor.

그림 4는 SOPC® Builder에서 설계된 4개의 Nios® II 임베디드 프로세서를 사용하여 공유메모리를 갖는 병렬처리 시스템이다.

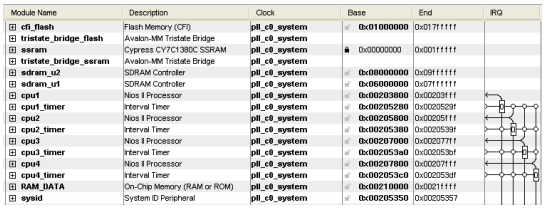

그림 4. 4개의 프로세서를 사용한 병렬처리 시스템의 H/W
Fig. 4. Parallel processing system H/W architecture using 4-processor.

3-3. 병렬처리를 위한 FFT 알고리즘

FFT는 DFT(discrete fourier transform)의 계산을 줄이고 고속으로 연산할 수 있는 알고리즘이며 FFT의 대표적인 변환 방법은 DIT(decimation-in-time)와 DIF(decimation-in-frequency) 방법이 있다. 여기서 사용하는 FFT는 DIT 방식을 사용하였다. N개의 샘플 값을 갖는 신호의 이산 푸리에 변환은 식(1)과 같이 정의되며 $X(k)$는 $x[n]$는 입력이며 X[k]는 $0 \le k \le (N-1)$)의 범위 안에서만 정의한다.

$$X(k) = \sum_{n=0}^{N-1} x[n] W_N^{kn} \ , \ (0 \le k \le (N-1))$$

$$where, \ W_N^{kn} = \exp^{-j\frac{2\pi}{N}kn} = \cos\frac{2\pi}{n}kn - jsin\frac{2\pi}{n}kn$$

$$(0 \le k \le (N-1)) \quad \cdots\cdots (1)$$

DIT에서 N개의 샘플 값을 가진 신호가 있다고 가정하면 N은 2의 승수이며 신호 x[n]을 2개(우수, 기수)의 N/2(N=2N1)개 샘플로 분할한다. 첫 번째는 x[n]의 우수 번째 신호이고 두 번째는 기수 번째 신호가 된다. 여기서 우수를 n=2r, 기수를 n=2r+1로 표현하면 $X(k)$는 식(2)와 같이 이산신호를 짝수 항과 홀수 항으로 분해할 수 있다.

$$\begin{aligned} X(k) &= \sum_{n=0}^{N-1} x[n] W_N^{kn} \\ &= \sum_{r=0}^{(N/2)-1} x[2r] W_N^{2rk} + \sum_{r=0}^{(N/2)-1} x[2r+1] W_N^{(2r+1)k} \\ &= \sum_{r=0}^{N1-1} x[2r](W_N^2)^{rk} + W_N^k \cdot \sum_{r=0}^{N1-1} x[2r+1](W_N^2)^{rk} \\ & where,\ N/2 = N_1 \end{aligned} \quad \cdots\cdots (2)$$

위의 식(2)에서 x[2r]과 x[2r+1]은 이산신호 x[n]의 짝수신호와 홀수신호이다. 이짝수신호와 홀수신호를 각각 다음과 같이 정의하면 식(3)에서 W_N^2는 다음과 같이 나타낼 수 있다.

$$W_N^2 = \exp^{-j\frac{2\pi}{N}2} = \exp^{-j\frac{2\pi}{N1}} = W_{N1} \quad \cdots\cdots (3)$$

따라서 식(4)는 식(2)와 같이 2개의 $N_1 - Point\,DFT$ 식으로 표현할 수 있다.

$$\begin{aligned} X(k) &= \sum_{r=0}^{N1-1} y[r] W_{N1}^{rk} + W_N^k \sum_{r=0}^{N1-1} z[r] W_{N1}^{rk} \quad \cdots\cdots (4) \\ &= Y(k) + \sum_{n=0}^{N-1} x[n] Z(k),\ (0 \le k \le (N-1)) \end{aligned}$$

식(5)의 $N - Point\,DFT\ X(k)$는 $N_1 - Point\,DFT\ X(k)$인 $Y(k)$와 $Z(k)$로 나타낼 수 있다. 그러나 식(4)에서 $Y(k)$와 $Z(k)$는 길이가 N_1이므로 N의 길이를 갖는 $X(k)$를 모두 구할 수 없다. 즉 위의 식(4)로는 $X(k)$의 0에서 $N-1$ 까지의 값을 구할 수 있다. N_1부터 $N-1$ 까지의 $X(x+N_1)$은 식(5)와 같이 구한다.

$$\begin{aligned} X(k+N_1) &= Y(k+N_1) + W_N^{(k+N_1)} Z(k+N_1) \quad \cdots\cdots (5) \\ &= Y(k+N_1) - W_N^k Z(k+N_1) \end{aligned}$$

위의 식(5)에서 $Y(k+N_1)$은 식(6)과 같이 표현할 수 있다.

$$\begin{aligned} Y(k+N_1) &= \sum_{r=0}^{N_1-1} y[r] W_{N1}^{r(k+N_1)} Z(k+N_1) \quad \cdots\cdots (6) \\ &= \sum_{r=0}^{N_1-1} y[r] W_{N1}^{rk} \\ &= Y(k) \end{aligned}$$

따라서 $Z(k+N_1)$는 $Z(k+N_1) = Z(k)$으로 나타낼 수 있으며 이 2개의 결과를 식(7), 식(8)과 같이 표현할 수 있다.

$$X(k+N_1) = Y(k) + W_N^k Z(k) \quad \cdots\cdots (7)$$

$$\begin{aligned} Z(k) &= Y(k) + W_N^k Z(k),\ (0 \le k \le (N-1)) \\ Z(k+N_1) &= Y(k) - W_N^k Z(k),\ (0 \le k \le (N-1)) \end{aligned} \quad \cdots\cdots (8)$$

DIT 구조에서 $N_1 - Point\,DFT$에서 스테이지(stage) 수는 $\log_2 N$이고 각 스테이지의 곱셈 수는 $\frac{N}{2}$이므로 총 곱셈의 수(T_m)은 식(9)와 같이 된다. 그림5는 8-Point를 사용한 FFT 연산 과정이며 그림6은 FFT 알고리즘의 실행과정이다.

$$T_m = \frac{N}{2} \times \log_2 N \quad \cdots\cdots (9)$$

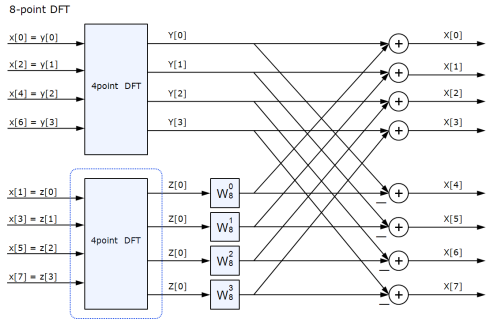

그림5. 8-Point를 사용한 FFT 연산 과정
Fig.5. Process of 8-point FFT Operation.

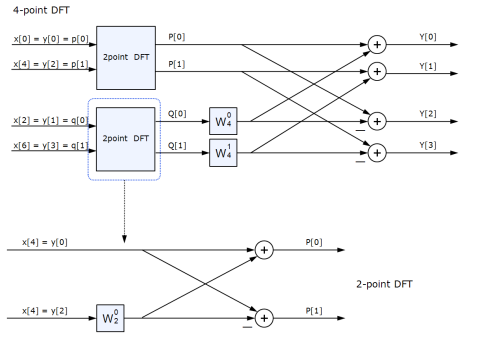

그림6. FFT 알고리즘의 실행과정
Fig . 6. FFT algorithm procedure.

3-4. FFT를 병렬처리하기 위한 S/W 구현

병렬처리 시스템의 S/W는 설계한 4-Core 병렬처리 시스템의 성능을 평가하기 위한 방법으로 N-point에 따른 FFT 연산을 수행한다. 연산은 N-point가 512, 1,024, 2.048, 4,096, 8,192를 1개, 2개, 4개의 프로세서로 처리한다. 먼저 처리할 데이터(sine, cosine 값)를 초기화한 다음 테이블로 저장한 사용 프로세서 개수를 입력 받으면 데이터 처리가 시작된다.

```
//-------------------------------------------------
// FFT Operation with 4-Core Embedded Processor

int main()
{
FFT_init_table(int N);                    // table initialization

input = getchar();
PERF_RESET(PERFORMANCE_COUNTER_BASE);

PERF_START_MEASURING(PERFORMANCE_COUNTER_BASE);

//-------------------------------------------------
// FFT starting
PERF_BEGIN(PERFORMANCE_COUNTER_BASE, 1);

FFT4_start(
                    N,
                    0,
                    1,
                    x,
                    X,
                    XX,
                    FFT_MSG_BUFF_BASE_CPU2,
                    FFT_MSG_BUFF_BASE_CPU3);

PERF_END(PERFORMANCE_COUNTER_BASE, 1);

perf_print_formatted_report(
                    PERFORMANCE_COUNTER_BASE,
                    alt_get_cpu_freq(),
                    1,
                    "FFT");

// Print resulting frequency-domain samples.
printf("\nX(k):");

for(i=0; i<N; i++)
        printf("\n       k=%d:  %12f  %12f", i, X[i][0],
X[i][1]);
        printf("\n\n");

        printf("\r\nFFT4 operation OK.....");

  return 0;
}
```

Ⅳ. 성능 평가 및 결과 고찰

4-1. 성능척도

병렬컴퓨터는 여러 개의 프로세서를 포함하는 많은 하드웨어 자원들을 중복적으로 사용하여 구성하는 시스템이기 때문에 단일 프로세서 시스템 보다 성능이 높아지는 것은 당연하다 그러나 여러 가지 요인들에 의해서 성능 향상은 그에 비례하지 못한다. 병렬컴퓨터의 성능척도는 처리속도, 속도향상(speedup: Sp)은 식(10), 효율(efficiency: Ep)은 식(11)과 같다. 여기서 Tp는 전체 프로세서의 개수이며, Ts는 전체 처리시간이다.

$$S_p = \frac{T_s}{T_p}, (1 < S_p < p) \quad \cdots\cdots (10)$$

$$E_p = \frac{S_p}{p} = \frac{T_s}{P \bullet T_p} \quad \cdots\cdots (11)$$

4-2. 성능 평가

표. 2는 4개의 임베디드 프로세서를 사용하여 설계한 병렬처리 시스템에서 프로세서의 사용대수에 따른 처리속도 결과이다. 그림 5는 병렬처리 시스템에 사용된 프로세서의 수(Np)와 처리 데이터(N-Point)에 따른 속도향상 결과이다. 프로세서의 처리속도(T_p), 속도향상(S_p)은 표 2와 같다. 그림 8과 같이 병렬처리 시스템의 처리속도는 처리 데이터가 많을 경우에 증가됨을 알 수 있다. 그림7은 설계한 병렬처리 시스템이 프로세서의 수와 처리 데이터 수에 따른 시간 (처리시간)이다.

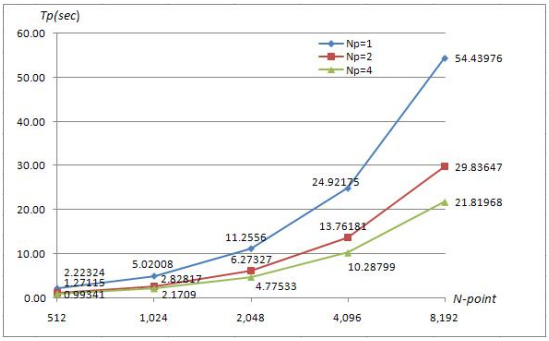

그림 7. 프로세서의 수와 처리 데이터 수에 따른 시간 비교(sec)
Fig. 7. Results of speed-up according to processor number and processing data.

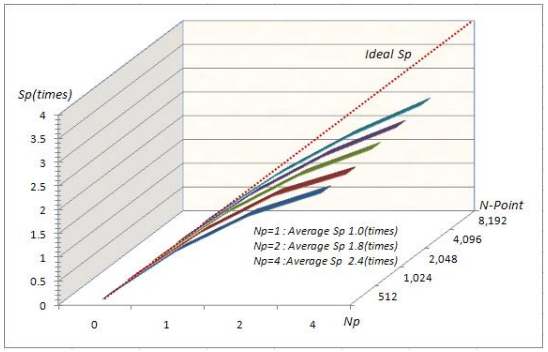

그림 8. 프로세서의 수와 처리 데이터 수에 따른 속도 결과(speed-up)
Fig. 8. Results of speed-up according to processor number and processing data.

표 3은 프로세서의 수(Np)와 처리 데이터(N-Point)에 따른 시스템의 효율이다. 설계한 병렬처리 시스템은 프로세서의 수가 증가함에 따라 시스템의 효율이 증가됨을 보였다 그러나 프로세서의 수에 따라 효율이 정비례하여 증가되지 않은 것은 각각의 프로세서에게 처리할 데이터를 할당하고 대기하는 시간 때문이다. 또한 데이터 수가 많을 경우 더 좋은 효율을 보였다.

4,096	T_p(sec)	24.92175	13.76181	10.28799
	S_p(times)	1	1.81094	2.42241
8,192	T_p(sec)	54.43976	29.83647	21.81968
	S_p(times)	1	1.82460	2.4950
평균 S_p(times)		1	1.8	2.4

표2. 프로세서의 수와 데이터 수에 따른 처리속도와 sped-up 결과 비교
Table2. results processing time and speed-up for processor number and data number.

N-Point	T_p, S_p	N_p		
		1	2	4
512	T_p(sec)	2.22324	1.27115	0.99341
	S_p(times)	1	1.74900	2.23799
1,024	T_p(sec)	5.02008	2.82817	2.17090
	S_p(times)	1	1.77571	2.31333
2,048	T_p(sec)	11.25560	6.27327	4.77533
	S_p(times)	1	1.79422	2.35703

표 3. 프로세서 시스템의 효율(Ep)
Table 3. Efficiency of multiprocessor system

N-Point	N_p		
	1	*2*	*4*
512	1	0.87	0.56
1,024	1	0.89	0.58
2,048	1	0.90	0.59
4,096	1	0.91	0.61
8,192	1	0.91	0.62
평균 E_p	1	0.90	0.59

4-3. 결과 고찰

설계한 병렬처리 시스템의 성능을 평가한 결과 1개의 프로세서를 사용한 경우에 비해서 2개일 경우 평균 1.8배의 처리속도가 향상되었으며 4개의 프로세서를 사용한 경우 평균 2.4배의 처리속도 향상을 가져왔다. 또한 시스템의 효율은 처

리되는 데이터의 수가 많을수록 처리속도가 향상됨을 보였다. 그러나 처리되는 데이터를 정확하게 분할하고 프로세서간의 통신을 적절하게 분담하는 방법에 따라 약간이 차이가 있음을 보였다. 이것은 동일한 데이터를 최적으로 분담하고 프로세서의 스케줄링을 최적으로 할 경우 처리속도는 개선될 수 있음을 보였다.

따라서 설계한 병렬처리 시스템은 시스템의 변경이 많고 적은 비용으로 고성능 데이터 처리가 요구되는 분야에 적용할 경우 시스템의 유연성, 처리속도 향상, 크기 및 가격에서 경쟁력을 가져올 수 있다.

V. 결 론

본 논문에서는 시스템의 변경이 많고 적은 비용으로 고성능 데이터 처리가 요구되는 분야에서 시스템의 처리속도를 향상하기 위한 목적으로 알테라(Altera) FPGA 환경에서 Nios® II 임베디드 프로세서를 4개 사용한 병렬처리 시스템을 구현하였다. 설계한 병렬처리 시스템은 1개의 프로세서를 사용한 것에 비해서 4개의 프로세서를 사용한 결과 평균 2.4배의 속도 향상을 가져왔다. 향후 진행될 연구내용은 설계된 병렬처리 시스템의 성능을 지속적으로 개선하는 것이며 동시에 또한 이를 저전력 형의 칩으로 제작하는 것이다.

참고문헌

〔1〕 P.C. Patton, "Multiprocessor Architecture and Applications," IEEE Computer, June 1985.

〔2〕 Vasilii Zakharov, "Parallelism and Array Processing," IEEE Computer, June 1985.

〔3〕 D.D.Gajski, J.K.Peir, "Essential Issue in Multiprocessor System," IEEE Computer, June 1985.

〔4〕 Vipin Kumar, Ananth Grama, Anshul Gupta, Georage Karypis, "Introduction to Parallel Computing Design and Analysis of algorithm," The Benjamin/Cummings Publishing Company, Inc., 1994.

〔5〕 Angel L. Decegama, "*The Technology of Parallel Processing*" Prentice-Hall, 1989.

〔6〕 Stephen Taylor, "*Parallel Logic Programming Techniques*" Prentice-Hall, 1898.

〔7〕 K. Mani Chandy, Jayadev Misra, "Parallel Program Design", Addison Wesley, 1988.

〔8〕 김종현, "*병렬컴퓨터구조론*," 생능출판사, 2004.

〔9〕 Lynn Fuerst, "*Introductory Digital Signal Processing with Computer Applications(2nd Edition)*"WILEY, 2008.

〔10〕 Paul A. Lynn & Wolfgang Fuerst, "*Introduction Digital Signal Processing*" WILEY, 2008.

〔11〕 Altera Corp, Nios® II User Manual. *www.altera.com*

저 자 소 개

이 시 현

1998년 : 건국대학교 대학원 전자공학과 졸업 (공학박사)

1991년-1995년 : (주)현대전자(현: 하이닉스 전자) / 산업전자연구소, 정보통신연구소, 멀티미디어연구소 근무

1999년-2009년 현재 : 동서울대학 정보통신과 조교수 재직중

관심분야 : Parallel System Design, SoC(ultra-low power processor & video signal processing), Bio-chip, Embedded System Design