論文 88-25-10-14

그래픽스 프로세서의 기능적 설계 및 시뮬레이션

(Functional-Level Design and Simulation of a Graphics Processor)

裵 成 玉*, 李 熙 轍***, 慶 宗 旻**

(Seong Ok Bae, Hee Choul Lee and Chong Min Kyung)

要 約

본 논문은 여러가지 그래픽스 시스템에서 사용될 수 있는 GP(graphics & processor)의 기능적 설계와 검증에 대해 기술한다. GP는 두 부분으로 나뉘어지는데 하나는 CPU이고 다른 하나는 입출력 기능을 담당하는 부분이다. 그래픽스 명령어를 빨리 수행하기 위해 CPU는 특별한 연산기와 배럴 쉬프터 및 윈도우 비교기를 가지고 있으며 명령어를 미리 읽어오기 위한 FIFO도 가지고 있다. 입출력 부분은 GP의 국소 메모리를 구성하고 있는 VRAM과 DRAM을 제어하고, 모니터를 동작시키기 위한 신호들을 발생시키며 HP(host processor)와의 정보교환을 담당한다. CPU의 기능 검증은 Daisy 워크스테이션에서 행해졌으며 입출력 부분은 실리콘 컴파일러의 일종인 GENESIL을 사용하여 설계되었다.

Abstract

This paper describes a functional-level design and simulation of Graphics Processor(GP) which can be used in various graphics systems. GP is divided into two parts: One is CPU, and the other is the interface to I/O peripherals. In order to achieve fast execution of graphics instructions, the CPU has special ALU, barrel shifter and window comparator and a FIFO for instruction prefetch. I/O part controls the DRAM and VRAM which constitute the GP's local memory, generates the signals to drive monitor, and communicates with the host processor. The functional simulation of CPU was done on Daisy workstation while the I/O part was designed using GENESIL, a silicon compiler.

I. 서 론

인간과 기계와의 interface 방식중 가장 효과적인 방법의 하나인 컴퓨터 그래픽스는 CAD/CAM, flight simulation, animation 등 여러분야에서 널리 사용되고 있으나 그 특성상 각 화소에 대해 일일히 연산을 해야하므로 software만 가지고는 그래픽스에 필요에 연산을 빨리 수행할 수 없다. 따라서 본 연구에서는 이러한 문제점을 해결하기 위한 방법의 하나로 그래픽스 하드웨어의 가장 기본이 되는 GP(graphics processor)를 설계하였다. GP는 VLSI의 집적도가 높아지고, 특히 VRAM (video RAM)[4]의 출현과 우수한 디스플레이(display)소자의 개발에 힘입어 꾸준히 발전해오고 있는데 현재 대표적인 것으로는 TMS 34010과 Intel 82786을 들 수 있다. TMS34010은 내

*準會員, **正會員, 韓國科學技術院 電氣 및 電子工學科
(Dept. of Electrical Eng., KAIST)
***準會員, 韓國電子通信研究所
(Electronics & Telecommunications Research Institute)

接受日字 : 1988年 5月 3日

부에 32bit 버스를 갖고 130여개의 명령어(그 중 23개의 그래픽스 명령어)를 수행하는 프로세서로 많은 일반 목적 명령어로 인하여 코프로세서(coprocessor)로 뿐 아니라, 독자적인 프로세서로도 사용될 수 있다.[1] Intel 82786은 TMS34010보다 그래픽스 명령어는 많으나 일반목적 명령어가 적은 코프로세서로 TMS34010보다는 범용성이 떨어진다.[2] 또한 이들은 공통적으로 CRT 제어기능과 DRAM 리프레쉬(refresh), host interface 기능을 하나의 chip으로 집적시킴으로써 주변장치와의 interface를 쉽게 할 수 있게끔 하였다. 이들 GP와는 달리 픽셀(pixel)의 크기가 커져도 빠른 속도로 그래픽스 명령어를 수행할 수 있는 그래픽스 시스템을 제안하고 이에 필요한 여러개의 chip을 개발한 경우도 있다.[3]

본 연구에서는 설계과정에서 범용성과 속도의 두 측면을 고려하였다. 일반 마이크로 프로세서에 버금가는 많은 일반목적 명령어를 포함시키고 여러 종류의 주변장치들과 효과적으로 interface할 수 있는 기능을 부여하여 범용성을 갖도록 하였으며 명령어를 미리 읽어오기 위한 FIFO 및 크기가 가변적인 픽셀을 효과적으로 처리하기 위한 새로운 구조의 masking 회로를 통해 속도향상을 꾀하였다. Daisy Workstation 및 GENESIL(silicon compiler)에서 제공하는 functional library를 이용하여 GP를 설계하고 필수적인 부분은 gate-level까지 설계를 수행하였으며 각 부분 및 전체적인 동작을 논리검증을 통해 확인하였다.

Ⅱ. 개 요

GP의 전체적인 구조를 설계할 때 속도와 면적이라는 측면에서 두가지 요소가 중요하게 고려 되었는데 그중의 하나가 parallelism의 극대화이고 다른 하나가 global connection의 최소화였다. 위의 두 관점에서 GP를 살펴보면 GP의 명령어처리 기능과 여러가지 inferface 기능들이 서로 독립적으로 수행될 수 있으므로 병렬구조에 적합하고 또 각 기능을 수행하는 블럭들이 메모리만을 공유하므로 각 블럭들과 메모리 사이의 데이타 전달을 위해 내부버스를 이용하면 global connection을 줄일 수 있다. 위의 두 사항을 만족시키는 GP의 구조를 그림 1에 나타내었다.

GP는 명령어를 수행하는 CPU, HP(host processor)와의 정보교환을 담당하는 host interface, 비디오 모니터를 리프레쉬시키는 CRT 제어기, VRAM과 DRAM을 리프레쉬시키는 DRAM 리프레쉬 제어기, 그리고 여러가지 메모리 싸이클을 수행하는 메모리 제어기로 구성되어 있다. 각 블럭은 서로 독립적으로 동작하며 메모리를 access할 필요가 있을때만 메

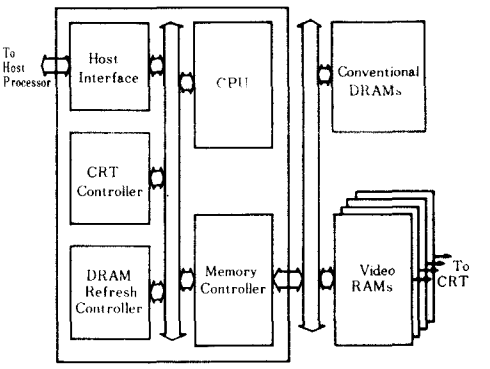


그림 1. GP의 구성도
Fig. 1. Block diagram of GP.

모리 제어기에 메모리 싸이클을 요구한다. 메모리제어기는 이러한 싸이클 요구를 우선순위에 따라 처리해 주며 각 블럭들과 메모리 제어기 사이의 정보는 16bit 주소 버스와 데이타 버스, 그리고 메모리 싸이클의 종류에 관한 정보를 보내기 위한 3bit opbus를 통해 이루어진다.

하드웨어적으로 제공되는 자료구조에는 필드(field), 픽셀(pixel), 이차원 픽셀어레이(2dimensional pixel array)가 있다. 필드는 그림정보를 제외한 나머지 데이타를 처리하기 위한 것으로 1-32bit 크기를 가지며 둘 또는 세개의 워드(word)에 걸쳐서 저장될 수도 있다. 픽셀은 그림정보를 위한 자료구조로 1, 2, 4, 8, 16bit 크기를 가질 수 있으며 필드와는 달리 여러 워드에 걸쳐 있을 수 없다. 위의 두 자료구조는 32 bit 시작주소와 크기에 의해 정의된다. 시작주소중 하위 4bit은 한 워드내에서의 변위를 나타내며 메모리 싸이클 수행시 출력되지 않고 bit addressing에 사용된다. 픽셀 어레이는 이차원 자료구조로 시작주소, 높이와 너비에 의해 정의되며 픽셀블럭전송(pixel block transter) 명령어에 의해 하나의 자료 구조로 취급된다.

Ⅲ. CPU부분

1. 명령어의 집합

그래픽스 명령어를 수행하는 그래픽스 프로세서를 설계하기 위해서는 먼저 명령어를 정하여야 한다. 실제 하나의 프로세서를 설계함에 있어서 명령어를 선택하는 일은 매우 어려운 일인데, 명령어를 정하기 위해서는 많은 경험과 설계하려는 프로세서의 사용용도를 정확히 알아야 한다. 본 논문에서 설계된 CPU는 일반 목적 명령어를 포함해 모두 109개의 명령어를 수행할 수 있으며 또한 많은 일반 목적 명령

표 1. 그래픽스 명령어의 예
Table 1. Example of graphics instructions.

Mnemonic		Operation	OP Code			
ADDXY	Rs, Rd	Add XY mode.	1110	000S	SSSR	DDDD
CMPXY	Rs, Rd	Compare XY mode.	1110	010S	SSSR	DDDD
CVXYL	Rs, Rd	Conversion from XY to linear.	1110	100S	SSSR	DDDD
DRAV	Rs, Rd		1111	011S	SSSR	DDDD
EXGXY	Rd	Exchange X, Y value.	1110	1010	000R	DDDD
FILL	XY	Fill area.	0000	1111	1110	0000
LINE		Draw line.	1101	1111	Z001	1010
MOVX	Rs, Rd	Move X value.	1110	110S	SSSR	DDDD
MOVXI	Rd	Move immediate X value.	0000	1010	000R	DDDD
MOVY	Rs, Rd	Move Y value.	1110	111S	SSSR	DDDD
MOVYI	Rd	Move immediate Y value.	0000	1010	001R	DDDD
PIXBLT	XY, XY	Pixel block transfer.	0000	1111	0110	0000
PIXT	*Rs, XY, *Rd. XY	Pixel transfer.	1111	010S	SSSR	DDDD
SUBXY	Rs, Rd	Subtract XY mode.	1110	001S	SSSR	DDDD

어를 포함하므로 (설계된 프로세서는) 독자적으로도 사용될 수 있다. 표 1에 대표적인 그래픽스 명령어들의 opcode와 그 기능을 나타내었다.

2. 전체적 회로구성

CPU의 전체적 회로구성은 그림 2에 나타나 있다.

회로의 전체적 구조를 설명하면, 명령어의 실행을 위하여 A, B, O버스가 있다. A, B 버스는 각각 레지스터(register) 출력과 연산 부분의 입력에 연결되어 있어서 레지스터의 값을 연산 부분으로 가져가는 버스들이다. O 버스는 연산 부분의 연산 결과를 레지스터의 입력으로 가져가는 기능을 한다. 이와 같은 버스 구조는 한 시스템 클럭(clock) 안에 연산을 행하고 그 결과를 원하는 레지스터에 실을 수 있게 한다. 이들 세개의 버스와는 달리 D버스는 명령어를 미리 가져오는데 사용되는 버스로서 임의의 일을 실행하면서 동시에 명령어를 가져올 수 있게 한다. 이와 같이 명령어를 가져오는 일은 각 명령어의 종류와 FIFO의 상태에 따라 trigger되고 이 일은 다음 명령까지 연장될 수도 있다. 만약 그 명령어가 입출력에 관계되는 일이면 현재 수행중인 명령어의 prefetch가 끝난 후에야 입출력 부분을 사용할 수 있다. 회로를 크게 나누면 연산 부분으로는 ALU, Barrel shifter, 윈도우 비교기(window comparator)가 있고 그 외에 프로그램 번지 계산 부분, 레지스터들과 제어 부분 등이 있다. 다음에서는 이들 각 부분에 대한 설명을 행한다.

1) 연산부분

ALU는 일반 프로세서의 ALU가 가지는 기능외에 GP에 필요한 특수기능을 가지고 있는데, 그 대표적 기능으로 픽셀 단위의 덧셈과 뺄셈을 들 수 있다. 이 기능은 여러 개의 픽셀을 한번에 옮기기 위해 필요한 것으로 포화덧셈(saturated addition)과 포화 뺄셈도 수행할 수 있다. 포화 덧셈을 예로 설명하면 이 덧셈은 두 점의 픽셀 값을 더하였을 때의 값을 가장 큰 값으로 포화시키는 연산으로서 두 색을 더하였을 때의 색이 가장 진한 색보다 진한 색이라면 가장 진한 색으로 포화시켜 나타내는데 이용된다.

Barrel shifter는 임의의 bit 수만큼 빠르게 shift시키거나 rotate시키기 위하여 필요한 연산 부분으로 X, Y좌표 값을 실제 메모리 번지로 환산하거나 픽셀을 워드 경계면에 일치시킬 때 사용한다.

윈도우 비교기는 프레임 버퍼(frame buffer) 상에서 정해진 윈도우의 위치와 지금 연산하는 픽셀의 위치를 비교하는 연산 부분으로 화면상에 그려지지 않는 연산은 행하지 않으므로써 속도를 개선할 수 있다.

이상의 연산 부분들은 제어 회로로부터 제어 신호를 받고 각 출력은 트라이스테이트 게이트(tri-state gate)를 거쳐 출력 버스인 O버스에 연결되어 제어를 받는다.

2) 프로그램 번지 계산부분

다음 번에 가져올 명령어의 번지를 계산하는 부분은 프로그램 번지를 보관하는 프로그램 번지 레지스터와 덧셈 뺄셈을 행하는 ADD-SUB와 하나의 임시 레지스터로 이루어져 있다. 여기서 사용된 프로그램 번지 레지스터는 일반적인 프로그램 계수기(counter)와 달리 다음 번에 가져올 명령어의 번지를 가리키는 값을 가지므로 인터럽트(interrupt)로 갈 때 스택

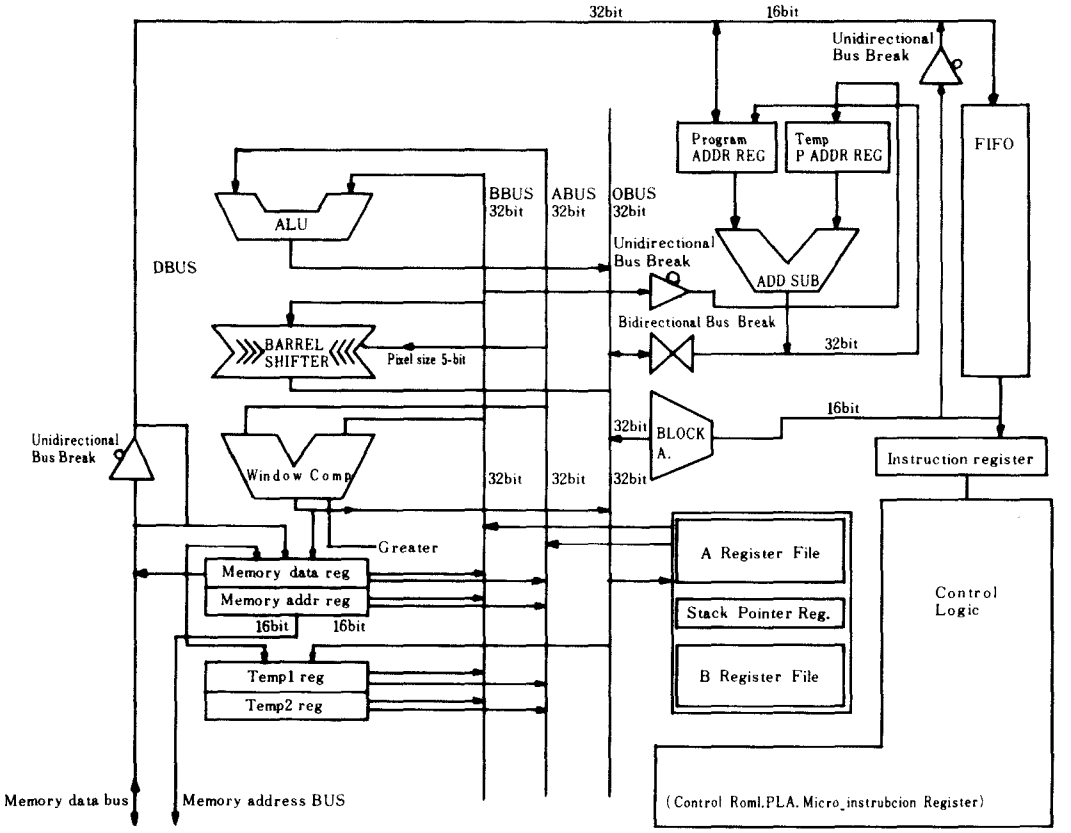


그림 2. CPU의 구성도
Fig. 2. Block diagram of CPU.

(stack)에 넣을 프로그램 번지를 보관하고 있는 FIFO 번지 계수기를 따로 가지고 있다. 이 프로그램 번지 계산 부분은 다음 번에 가져올 명령어의 번지를 독자적으로 계산하는 부분으로 연산 부분과는 버스 breaker를 통하여 연결되어 있고 명령어 fetch 순서가 시작될 때 독자적인 제어를 받는다.

3) 레지스터들

두개의 레지스터 화일과 status 레지스터, stack pointer 레지스터, 두 개의 임시 레지스터와 메모리 제어기에 연결된 메모리 데이타 레지스터, 메모리 번지 레지스터가 있다. Status 레지스터는 CPU 내부의 상태를 기억하고 있는 레지스터로서 ALU의 연산이 일어날 때마다 내용이 변하는 N, Z, V, C bit과 인터럽트와 관계되는 bit, 필드의 크기를 나타내는 bit 등으로 이루어져 있다. Stack pointer 레지스터는 외부 stack의 메모리 번지를 보관하고 있는 곳이다. 두 레지스터 화일은 각각 15개의 레지스터들로 이루어져 있고 이중 하나의 화일은 일반 목적 레지스터로 사용되고 다른 하나의 화일은 그래픽스 명령어를 수행할 때 특수 목적 레지스터로 사용되나, 일반 목적 레지스터로도 사용될 수 있다. 이들은 모두 32 bit의 폭을 가지며 전체적 버스와 연결되어 있다.

4) 제어부분

CPU내의 각 블럭의 제어신호를 발생하는 회로는 그림 3과 같다.

명령어는 FIFO를 거쳐서 들어오는데, 이 FIFO는 4단의 깊이를 가지며 각 단에 flag를 가지고 있어 미리 읽어온 데이타를 비어있는 가장 아랫단에 넣을 수 있다. 현재 수행중인 명령어는 명령어 레지스터에 들어있으며 PLA를 통하여 제어 ROM의 번지를 발생하고 그 번지부터 제어 ROM의 micro-program된 내용에 따라 동작하게 된다. PLA를 거쳐 발생된 번지는 micro-instruction counter (M_COUNT)에 실리게 되고 제어 ROM은 그 내용을 증가시키거나 변화시키면서 각 명령어에 정해진 일을 행한다. 본 CPU 부분에는 많은 레지스터가 있어 이들을 제어하는 기능을 제어 ROM에 적을려면 같은 명령어일지라도 레지스터가 다른 경우에는 명령어

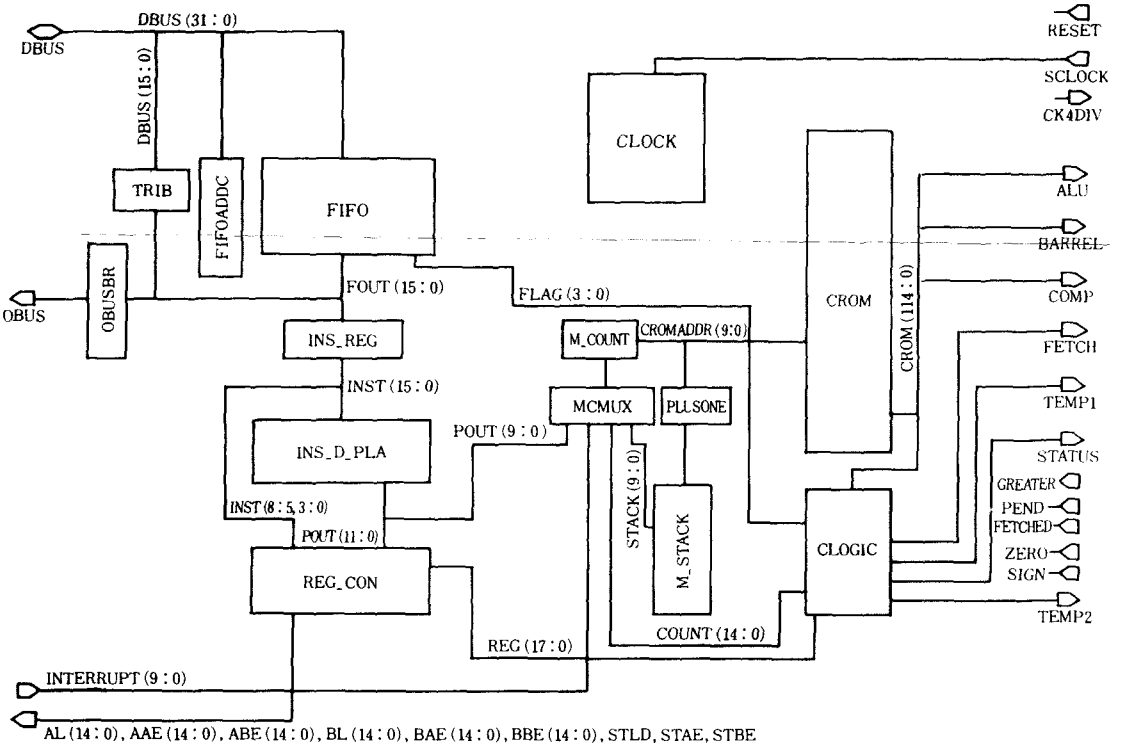


그림 3. CPU 제어기의 구성도
Fig. 3. Block diagram of controller in CPU.

coding을 다르게 하여야하므로 제어 ROM이 많은 면적을 차지하게 된다. 명령어가 같은 경우에는 레지스터가 다르더라도 쉽게 레지스터 제어를 행하기 위하여 레지스터 제어 부분을 특별히 설계하였다. 특히 내부에 stack을 두어서 인터럽트 처리를 그래픽스 명령어같이 시간이 많이 걸리는 명령어 수행중에도 행할 수 있도록 하였을 뿐아니라, 복잡한 명령어 수행중에 단순한 명령어의 수행 루프를 subroutine처럼 사용할 수 있게 하였다. 그림 3에서 보이는 CROM밑의 CLOGIC은 제어 ROM(CROM)에서 나오는 신호에 delay를 주거나 이를 조합하여 새로운 신호를 만드는 곳이다. 이 그림은 Daisy workstation 상에서 schematic 입력으로 사용되는 그림을 데이타의 흐름만 나타나도록 제어신호들을 없애버린 것이다.

3. 명령어 Coding

명령어 coding이란 외부에서 볼 수 있는 명령어들을 하드웨어의 각 state(micro-instruction 이라 칭함)들로 원하는 동작을 하도록 제어 ROM의 내용을 적는 것을 말한다. 명령어 coding은 하드웨어와 관계되는 것이며 본 논문에서 제시하는 하드웨어에서의 명령어 coding은 그림 4와 같은 구조로 행해져야 한다.

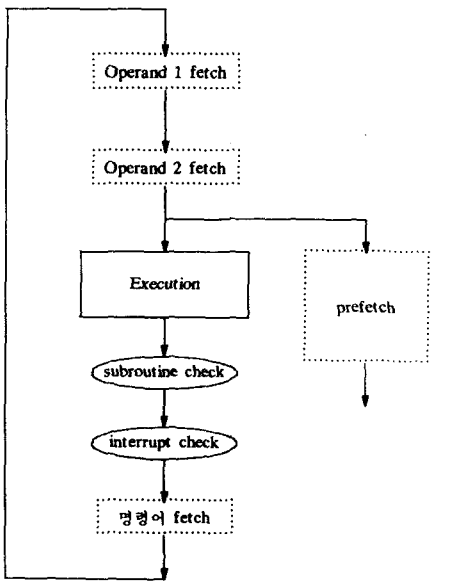


그림 4. 명령어의 구조
Fig. 4. Construction of instructions.

그림 4 에서 operand fetch가 필요없을 때는 행하지 않겠지만, 필요할 때라도 FIFO에 미리 operand가 fetch되어 있을 수 있어 fetch하는데 걸리는 시간을 절약할 수 있다. 미리 읽혀진 operand는 OB-USBR(그림 2. CPU 전체 회로도 참조)을 거쳐 바로 O 버스에 실릴 수 있다. 명령어를 fetch할 때도 FIFO가 비어 있지 않으면 이미 명령어가 fetch되어 있는 것이므로 단지 FIFO에서 shift시켜 명령어 레지스터에 싣고, PLA 출력을 micro-instruction counter에 실으면 된다. 만약 FIFO가 비어 있다면 fetch request를 한 다음 명령어가 fetch될 때까지 기다려야 한다

Ⅳ. 입출력 부분

GP의 주변장치로는 GP를 제어하는 HP와 프로그램 및 일반적인 데이타와 화상정보를 저장하는 국소 메모리(local memory), 결과를 출력시키는 CRT 등이 있다. 입출력 부분은 이러한 주변장치들과 GP와의 interface를 담당하는 부분으로 host interface, CRT 제어기, 메모리 제어기 그리고 GP의 국소 메모리를 구성하고 있는 DRAM 및 VRAM을 주기적으로 리프레쉬 시키는 DRAM 리프레쉬 제어기로 이루어져 있다.

1. Host interface

HP는 host interface가 제공하는 여러가지 제어신호와 4개의 레지스터를 이용해 GP의 국소 메모리를 access할 수 있다. 연속된 메모리 블럭의 access를 쉽게 하기 위해 access할 때마다 주소가 자동으로 증가하는 주소자동증가 기능도 가지고 있으며 8 bit와 16 bit 프로세서 모두가 연결될 수 있도록 설계되었다.

1) Host interface 레지스터의 읽기와 쓰기

Host interface 레지스터의 읽기와 쓰기는 HP에 의해 시작되며 host interface 버스의 각 신호에 의해 제어된다. Host interface bus는 16 bit 양방향 버스인 HD0-HD15, 읽기와 쓰기 싸이클을 나타내는 HREAD, HWRITE, access할 레지스터를 선택하기 위한 HRS0, HRS1, 선택된 레지스터의 상위 또는 하위 바이트중 어느 부분을 access할 것인가를 나타내는 HUB/LB, host interface의 상태를 HP에 알려 주는 HREADY, GP의 인터럽트를 HP에 전달하는 HINT 등으로 이루어져 있다. Host interface 에는 HCTRL, HDATA, HADDRH, HADDRL 등 모두 4개의 레지스터가 있으며 HP는 HRS0, HRS1을 이용해 이중의 하나를 선택할 수 있다.

HADDRH와 HADDRL은 HP가 access할 메모리 주소중 각각 상위 16 bit와 하위 16 bit을 가지고 있으며 HDATA는 HP와 GP사이에 전달될 데이타를 일시적으로 저장하기 위한 레지스터이고 HCTRL은 host interface의 기능을 제어하는 여러가지 field를 가지고 있다.

2) 국소 메모리의 간접 access

HP는 host interface에 있는 특정 레지스터를 access하므로써 메모리 싸이클을 일으킬 수 있다. 싸이클이 일어날 조건은 주소 레지스터에 주소가 모두 실렸거나 HDATA가 모두 access되었을 경우이다. HCTRL의 BM bit과 LBL bit이 이러한 조건을 검사하는데 사용된다. 예를 들어서 BM = 1이고 LBL = 1이면 HP가 8 bit 프로세서이고 하위 바이트를 나중에 access하므로 HADDR의 하위 바이트에 HP가 write를 하면 주소가 다 실린 것으로 간주된다. 반대로 LBL = 0이면 HADDRH의 상위 바이트에 주소가 실려야 읽기 싸이클(read cycle)이 수행된다. HDATA에 대한 access로 위와 같이 동작된다. 읽기 싸이클은 주소가 모두 실렸거나 HDATA를 모두 읽어 갔을 때, 쓰기 싸이클(write cycle)은 HDATA에 데이타가 모두 실렸을 때 각각 일어난다.

두 증가제어 bit, INCR과 INCW가 1이면 연속적인 읽기와 쓰기 동안에 주소가 자동으로 증가하는데 HP는 이 기능을 이용해 매 access마다 새로운 주소를 싣지 않고도 연속된 메모리 블럭을 한 워드씩 차례대로 읽거나 쓸 수 있다.

2. CRT 제어기

CRT 제어기는 그래픽스 시스템의 출력장치인 비디오 모니터를 제어하는 블럭으로 모니터 제어신호를 발생시키며 동시에 VRAM의 특수 싸이클을 이용해 화면 리프레쉬를 수행한다. VRAM은 일반적인 DRAM이 갖고 있는 입출력 포트(I/O port) 외에 내부의 쉬프트 레지스터를 위한 포트가 따로 있어서 2포트 메모리라 불리기도 하는 특수 기억소자로 한 row의 내용을 쉬프트 레지스터로 옮기는 MSRC(memory to shift register cycle) 와 그 반대의 기능을 갖는 SRMC(*shift register to memory cycle*)를 가지고 있다. CRT로 출력시킬 데이타를 MSRC를 이용해 내부 쉬프트 레지스터로 옮긴 다음 이를 다시 순차적으로 출력시켜 화면을 리프레쉬시킨다.

CRT 제어신호에는 수평 동기 신호(HSYNC), 수직 동기 신호(VSYNC) 그리고 수평, 수직 blank 신호의 logical ORing에 의해 발생되는 BLNK 신호가 있으며 이 신호들은 여러개의 timing 레지스터와 계수기를 이용해 발생된다. 발생되는 신호들의 timing이 레지스터의 값에 의해 결정되기 때문에 CRT 의

사양이 바뀌더라도 해당 레지스터의 값만 바꾸면 되도록 하였으며, 또한 interlace와 noninterlace video 모두를 support 하도록 설계하였다.

3. 메모리 제어기

메모리 제어기는 GP의 국소 메모리를 구성하고 있는 DRAM이나 VRAM을 제어하기 위한 여러가지 신호들을 발생하고 GP의 각 내부 블럭으로부터 온 여러가지 싸이클 요구를 처리해주는 블럭으로 그 구조는 그림 5와 같다.

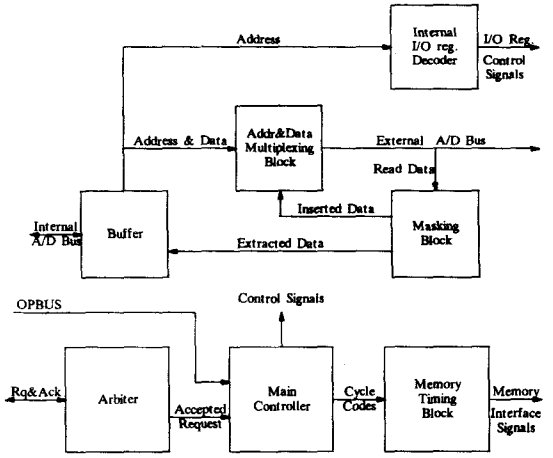


그림 5. 메모리 제어기의 구성도
Fig. 5. Block diagram of memory controller.

각 블럭들이 메모리 싸이클을 요구하는 시간이 일정치 않으므로 한꺼번에 여러 개의 요구가 올 수 있다. 이러한 문제를 해결하기 위해 각 블럭마다 우선순위를 주고 조정자(arbiter)는 여러 개의 메모리 싸이클 요구중에서 우선순위에 따라 하나를 선택해 낸다. 우선순위는 화면 리프레쉬 싸이클 요구, DRAM 리프레쉬 싸이클 요구, HP가 요구한 싸이클, CPU의 메모리 싸이클 요구 순이다. 매스킹(masking) 블럭은 삽입이나 추출 그리고 픽셀 처리를 위한 PM (plane masking)과 TM (transparency masking)을 수행한다. PM은 픽셀내의 특정 bit이 그래픽스 명령어에 의해 변화되는 것을 막아주고 TM은 값이 0인 픽셀이 0이 아닌 픽셀을 변화시키지 못하게 하므로써 물체만을 화면상에 그리고자 할 때 사용된다. 삽입및 추출에 대해서는 아래절에서 자세히 설명하겠다. 내부 입출력 레지스터 디코더(Internal I/O register decoder)는 GP의 메모리 상에 위치한 내부 입출력 레지스터에 대한 access를 검출해 낸다. 메모리 타이밍 블럭은 기억소자들을 제어하기 위한 여러가지 신호들을 발생시키며 이러한 신호들을 이용해 읽기 싸이클(read cycle), 쓰기 싸이클(write cycle), 한 워드를 읽어와서 일부를 고친 다음 다시 메모리에 쓰는 read modify write cycle, VRAM을 제어하기 위한 memory to shift register cycle, shift register to memory cycle, DRAM 리프레쉬를 위한 RAS only refresh cycle, CAS before RAS refresh cycle 등 모두 7종류의 메모리 싸이클을 수행한다. 메모리 제어신호에는 메모리 소자를 직접 제어하기 위한 RAS(row address strobe), CAS(column address strobe), TR/QE(transfer/output enable), W(write)와 사용자를 위한 CAL, DEN, DDOUT이 있다. CAL 신호는 주소와 데이타의 순차출력에 의해 컬럼 주소가 출력되는 시간이 짧기 때문에 버스상의 컬럼 주소를 외부회로에 저장하기 위해 사용되며 DEN이 low이면 버스상의 데이타가 안정되었음을 나타내고, DDOUT은 데이타의 방향을 나타내는 신호로 low일 때 데이타가 입력된다. Pin수를 줄이기 위해 하나의 16 bit 버스를 이용해 주소와 데이타를 순차적으로 출력시키도록 설계하였다.

1) 삽입 및 추출

GP의 워드 크기가 16 bit인데 반해 CPU가 처리하는 자료구조들의 크기가 가변적이므로 이들을 효과적으로 처리하기 위해서는 bit 단위의 access가 가능해야 한다. 즉 한 워드안에서 일부 bit만 추출해 내거나 삽입할 수 있어야 한다. 이것은 삽입 및 추출 매스크를 이용한 매스킹에 의해 이루어지는데 매스크는 EPT(edge painting tree)[5]를 이용해 발생된다. EPT는 원래 다각형을 채우는 하드웨어로 왼쪽 위치와 오른쪽 위치의 두 위치 데이타를 입력으로 받아 두 점사이에 있는 각 점을 1로 만들어 준다. EPT를 이용한 삽입 및 추출 매스크 발생 회로를 그림 6에 나타내었다.

LPOS 레지스터에는 주소의 하위 4 bit이 들어오고 RPOS 레지스터에는 LPOS의 값에 픽셀이나 필드의 크기를 더한 값이 실린다. 픽셀이나 필드 크기는 LPOS 값에 더해지기 전에 1씩 감소되는데 이유는 필드나 픽셀은 그 크기가 1일 때 이진수 1로 표시되나(*필드 크기 32는 0으로 표시됨) EPT는 왼쪽과 오른쪽 위치 입력이 각각 (0,0)일 때 그 출력의 첫번째 하위 bit 이 1이 되기 때문이다. 즉, LPOS가 0이고 데이타의 크기가 1일 때 하위 첫 bit만 1인 매스크를 얻기 위해서는 EPT 입력이 (0,0)이 되어야 하므로 데이타 크기를 1만큼 감소시켜 0

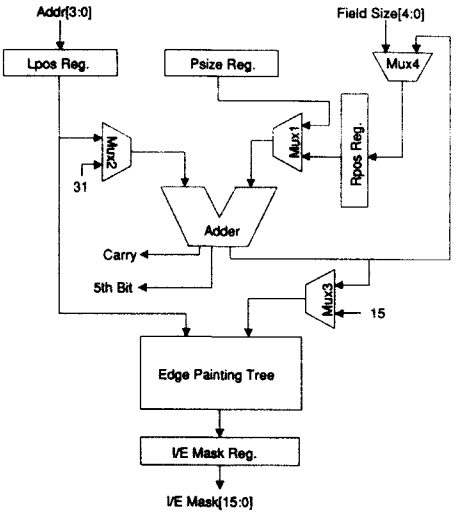


그림 6. 삽입및 추출 매스크 발생회로
Fig. 6. Insertion and extraction mask generation circuit.

으로 만들어야 한다. MUX1은 픽셀이나 필드 크기를 1감소시킬 때 사용된다. 필드가 2 또는 3워드에 걸쳐있을 경우 두번 또는 세번 읽어와야 하는데 맨 처음 읽어올 때 매스크의 출력은 왼쪽 위치에 해당하는 bit 부터 bit 15까지 모두 1이 되어야 한다. 따라서 RPOS에는 15가 있어야 하는데 이 값은 가산기를 이용해서는 얻을 수 없으므로 MUX3를 이용해 15를 EPT의 오른쪽 위치 입력으로 넣어준다. 추출을 할 때는 매스크와 데이타를 ANDing하고 삽입할 때는 삽입될 데이타와 매스크를 ANDing해서 나온 출력과 메모리에서읽어온 데이타와 inverted-mask와 ANDing한 결과를 ORing한다.

2) 주 제어기(Main controller)

주 제어기는 메모리 제어기를 총괄해서 제어하는 회로로, PLA를 이용한 유한 상태 머쉰(finite state machine)으로 설계되었다. 그림 7의 CLK1과 CLK 2는 입력 클럭의 8배의 주기를 가지며 서로 90도의 위상차가 있는 신호들로 시스템 클럭과 함께 전체 chip의 동작을 제어하기 위한 신호들이다.

CLK1과 CLK2의 위상에 따른 주 제어기의 동작은 다음과 같다. 시간 T0에서 조정자를 이용해 싸이클 요구를 검사한다. 요구가 없으면 요구가 들어올 때까지 CLK1과 CLK2가 모두 low일 때마다 검사를 행한다. 요구가 있으면 조정자는 요구가 받아들여진 블럭에 인지(acknowledge) 신호를 보낸다. 인지신호를 받은 블럭에서는 주소 버스를 통해 하위16 bit 주소를 메모리 제어기에 보내고 이 주소는 메모리 제어기의 주소 레지스터에 저장된다. Time T1에

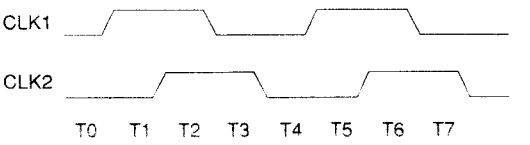


그림 7. CLK1과 CLK2
Fig. 7. CLK1 and CLK2.

서 상위 16 bit 주소와 데이타가 각각 주소 레지스터와 데이타 레지스터에 실리고 동시에 주 제어기는 블럭에서 보내온 opcode를 해석하여 싸이클 시작 신호와 함께 메모리 싸이클의 종류를 coding한 신호를 메모리 타이밍 블럭에 보낸다. Time T2에서 메모리 버스에 row 주소가 출력되면서 메모리 싸이클이 시작된다. T0에서 요구를 받아들임으로써 시작된 메모리 싸이클은 CLK1신호의 3주기동안 수행되는 read modify write cycle을 제외하고는 모두 T7에서 끝난다. 이때 수행되었던 싸이클이 읽기 싸이클인 경우 데이타가 내부 데이타 버스를 통해 싸이클을 요구했던 블럭으로 보내지고 동시에 새로운 싸이클을 시작하기 위해 싸이클 요구가 들어왔는 지를 검사한다.

필드 읽기와 쓰기는 필드의 크기와 시작 주소에따라 최대 3개의 싸이클을 요구하는데 주 제어기는 삽입 및 추출 매스크 발생 회로의 가산기 출력을 이용해 이 요구를 개별적인 메모리 싸이클로 바꾸어 수행한다. 필드 읽기때 캐리(carry)가 발생하면 필드가 3워드에 걸쳐 있는 경우이므로 읽기 싸이클이 세번 수행되고 가산기의 다섯번째 출력 bit이 1이면 두번, 캐리도 발생하지 않고 다섯번째 출력 bit도 0이면 한번 수행된다. 필드 쓰기일 경우도 필요한 싸이클 수는 읽기와 같은 방법으로 계산되나 시작주소와 가산기의 출력을 검사해 자료구조가 쓰여질 위치가 메모리상의 워드 위치와 일치하면 쓰기 싸이클이 일치하지 않으면 read modify write cycle이 수행된다. 필드가 둘 또는 세 워드에 걸쳐있을 때 2개 혹은 3개의 메모리 싸이클이 수행되는데 첫번째 싸이클에서 수행될 싸이클 수가 계산되고 또 싸이클이 끝나고 나면 삽입 및 추출 매스크 발생 회로의 LPOS에는 0이, RPOS에는 가산기 출력의 하위 4bit이 기억되므로 다음에 수행될 싸이클의 종류를 앞서 계산된 싸이클 수와 가산기의 출력을 이용해 알 수 있다.

필드 읽기나 쓰기 싸이클은 더 높은 우선순위를 갖는 싸이클 요구에 의해 지연될 수 있다. 예를 들어 필드 읽기가 두 개의 싸이클을 요구할 때 첫번째 싸

이클이 수행되는 도중에 화면 리프레쉬 싸이클 요구가 들어오면 첫번째 싸이클이 끝난 다음에 화면 리프레쉬 싸이클이 수행되고 그 다음에 지연되었던 두번째 읽기 싸이클이 수행된다. 이렇게 함으로써 우선순위가 높은 요구들의 latency time을 줄일 수 있다. Latency time은 싸이클 요구가 일어나서 그 싸이클 수행이 시작될 때까지 걸리는 시간이다. 픽셀은 필드의 특수한 경우로 취급된다. 단, 픽셀은 워드 경계에 걸쳐 있는 경우가 없고 최대 크기가 16bit이므로 필드와 같이 2~3 싸이클이 걸리는 경우가 없다.

V. 논리검증 결과

CPU 부분의 논리검증에는 Daisy workstation을 사용하였는데 메모리 제어 부분과 프로그램 메모리를 하드웨어 디스크립션 언어를 사용하여 간단히 모델링하여 LINE이란 명령어를 부르는 프로그램을 돌려보았다. 이 때 점(1,1)에서 점(30,50)을 잇는 금을 긋기위해 프로그램 메모리상에 특별목적 레지스터의 값을 주는 일과 LINE이란 명령어를 수행하도록 프로그래밍하였다.[6] LINE의 코딩은 Bresenham의 알고리듬을 이용하였다.[7] 109개의 정한 명령어 중 전체적인 동작을 검사할 수 있는 10여개를 선택하여 코딩하였으며 각각에 대해서 논리검증을 행하였다. 적은 수의 명령어를 코딩하였지만 제시한 회로구조로부터 원하는 그래픽스 프로세서의 동작을 얻었다.

입출력 부분의 논리검증은 GENESIL에서 제공하는 functional simulator를 이용해 행해졌으며 VRAM의 특수 싸이클 중 화면 리프레쉬에 사용되는 MSRC(*Ⅳ 2절 참조)의 검증결과를 그림 9에 나타내었다. 모든 싸이클은 CLK1과 CLK2의 timing을 기준으로 수행되는데 이 싸이클은 일반적인 DRAM 싸이클과는 달리 RAS가 high일 때 TR/QE신호가 low로 떨어진다. 이러한 timing은 VRAM에 특수 싸이

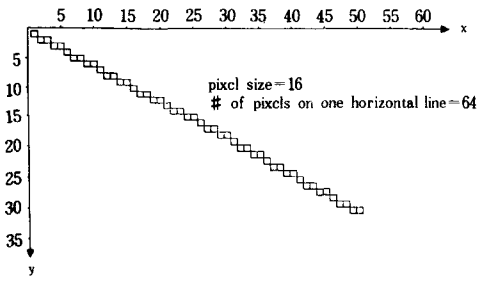


그림 8. LINE 명령어의 검증결과
Fig. 8. The simulation result of LINE instruction.

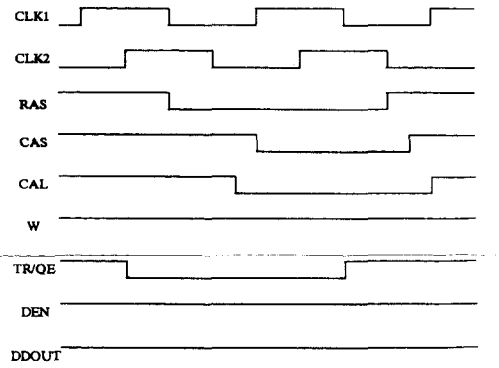


그림 9. MSRC에 대한 검증결과
Fig. 9. The simulation result of MSRC.

클이 시작되었음을 알리고 이 때 VRAM은 W가 high이면 MSRC로, low이면 SRMC로 인식하게 된다. TR/QE가 low로 떨어질 때 RAS와 W가 모두 high이므로 싸이클이 올바르게 수행되고 있음을 알 수 있다. 이 싸이클에서는 메모리 버스를 통한 데이타의 입출력이 없으므로 DDOUT과 DEN은 high 상태를 유지한다. 입출력부분의 다른 블럭들도 이와같은 시뮬레이션을 통하여 우리가 의도했던 대로 동작함을 확인하였다.

Ⅵ. 결 론

여러종류의 그래픽스 시스템에서 효율적으로 사용될 수 있는 GP의 기능과 전체적인 구조를 정의하고 이를 functional-level에서 설계, 검증하였다. CPU 부분은 빠른 그래픽스 명령수행을 위한 여러가지 하드웨어 및 독특한 버스시스템과 명령어를 미리 읽어오기 위한 FIFO를 가지고 있으며 기본적인 마이크로코드를 여러 명령어 수행 루프들이 공통적으로 사용할 수 있도록 마이크로 스택을 사용하므로써 제어 ROM의 용량을 줄이도록 노력하였다. 입출력 부분은 주변장치인 HP와의 interface와 비디오 모니터, VRAM 및 DRAM 등을 제어하기 위해 필요한 기능들을 가지고 있으며 제어신호발생에 레지스터를 도입하므로써 입출력 장치들의 사양이 바뀌더라도 단지 레지스터값만 바꾸면 되게 설계하여 programmability를 높였다. 그래픽스 시스템은 여러크기의 픽셀을 제공하여야 하는데 이를 위해 꼭 필요한 것이 bit addressing을 가능케 하는 삽입 및 추출기능이다. 삽입 및 추출을 위한 매스크 발생은 쉬프트레지스터를 통해 할 수도 있으나 이 경우에 한 워드안에서 데이타의 크기와 번위에 따라 매스크의 발생시간이 달라지

는 문제가 생기는데 본 논문에서는 image rasterization 하드웨어인 EPT가 삽입 및 추출 매스크 발생에 효과적으로 사용될 수 있음을 보이고 새로운 매스크발생회로를 제시하였다. 또한 이를 기반으로 새로운 형태의 read modify write cycle을 도입하여 워드 경계면에 일치하지 않는 자료구조들의 write time을 줄였다. 기존 chip[1] 에서는 개별적인 읽기와 쓰기 싸이클을 이용해 위의 동작을 수행하나 본 chip에서는 매스크 발생시간이 항상 일정하므로 이를 하나의 싸이클로 만들 수 있었다. 이 회로는 앞으로가변크기 자료구조를 제공해야 하는 모든 프로세서에 효과적으로 사용될 수 있다고 생각된다.

參 考 文 獻

[1] Mike Asal, Graham Short, Tom Preston, Richard Simpson, Derek Roskell, Karl Guttag: Texas Instrument, "The Texas Instruments 34010 Graphics System Processor," *IEEE CG & A*, pp. 24-39, October 1986.

[2] Glen Shires: Intel Corporation, "A New VLSI Graphics Coprocessor The Intel 82786," *IEEE CG & A*, pp. 49-55, October 1986.

[3] Charles Carinalli, John Blair: National Semiconductor, "National's Advanced Graphics Chip Set for High-Performance Graphics," *IEEE CG&A*, pp. 40-48, October 1986.

[4] Ray Pinkham, Mark Novak, Karl Guttag, "Video RAM excels at fast graphics," Electronic Design pp. 161-171, August 1983.

[5] 최상길, 김성수, 어길수, 경종민, "Image rasterization을 위한 Edge Painting Machine의 설계 및 simulation," 1987 전기 전자공학 학술대회 논문집(Ⅱ) pp. 1492-1494.

[6] 이희철, "Design of A Graphics Processor : CPU part," 1988 KAIST 석사학위 논문.

[7] Computer science series "Principles of Interactive Computer Graphics," McGraw Hill Book Company pp. 17-27.

[8] Jeffrey L. Wise, Henryk Szejnwald, "Display controller simplifies design of sophisticated graphics terminals," Electronics pp. 153-157, April 1981.

[9] 배성욱, "Design of A Graphics Processor : I/O part," 1988 KAIST 석사학위 논문.

[10] Mary C. Whitton, "Memory Design for Raster Graphics Displays." *IEEE CG & A*, pp. 48-65, March 1984.

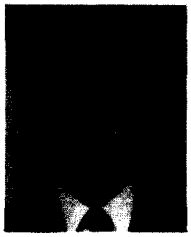
[11] "TMS34010 User's Guide," Texas Instruments.

[12] Jack N. Fenner, Jeffrey A. Schmidt, Houssam A. Halabi: Purdue University, "MASCO: The Design of a Microporgrammed Processor," *IEEE COMPUTER*, pp. 41-43, March 1985.

[13] Jhon DeRosa, Richard Glackemeyer, Tomas Knight, "Design and Implementation of the VAX 8600 Pipeline," *IEEE COMPUTER* pp. 38-48, May 1985.

[14] Subhash Bal, Asher Kaminker, Yoav Lavi, "The NS16000 Family Advances in Architecture and Hardware," *IEEE COMPUTER* pp. 58-67, June 1982. *

著 者 紹 介

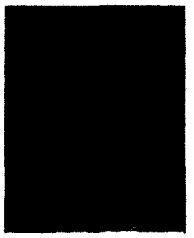


裵 成 玉(準會員)

1964年 5月 15日生. 1986年 2月 충남대학교 전자공학과 공학사학위 취득. 1988年 2月 한국과학기술원 전기 및 전자공학과 석사학위 취득. 1988年 3月～현재 한국과학기술원 전기 및 전자공학과 박사과정. 주관심분야는 Computer Graphics 등임.

李 熙 轍(準會員)

1964年 5月 29日生. 1986年 2月 영남대학교 전자공학과 공학사학위 취득. 1988年 2月 한국과학기술원 전기 및 전자공학과 석사학위 취득. 1988年 3月～현재 한국전자통신연구소 연구원. 주관심분야는 Computer Graphics 등임.



慶 宗 旻(正會員)

1953年 6月 21日生. 1975年 서울대학교 전자공학과 공학사학위 취득. 1977年 한국과학기술원 전기 및 전자공학과 석사학위 취득. 1981年 한국과학기술원 전자공학과 박사학위 취득. 현재 한국과학기술원 전기 및 전자공학과 부교수. 주관심분야는 CAD, Computer Graphics, Neural Net 등임.