論 文

하드웨어 설계용 언어의 실현

正會員 李 明 浩* 正會員 李 近 萬**

The Implementation of the Hardware Disign Language

Myung Ho LEE,* Keun Man Yi** *Regular Members*

要 約 본 논문에서는 복잡한 디지털 시스템 설계를 위해 하드웨어 기술 언어와 하드웨어 컴파일러에 관해 정의한 후, 레지스터 전송 알고리즘을 이용하여 데이터 부시스템의 테이블을 생성하는 번역기를 실현하였다.

ABSTRACT This paper presents definitions of the hardware description language and hardware compiler for the complex digital system design. In this paper, The translator is implemented for generating the table of data subsystems using the register transfer algorithm.

I. 서 론

디지털 시스템은 2개의 부시스템 (Subsystem)인 데이터 부시스템 (Data Subsystem)과 제어 부시스템 (Control Subsystem)으로 구성된다.

이들 데이터 부시스템과 제어 부시스템은 외부 입력신호인 자료 입력과 제어 입력을 신호로 하여 이에 따른 제어 신호와 상태 신호를 교환함으로써 원하는 데이터 출력 및 제어 출력을 얻음과 동시에 순서에 따른 시스템 동작을 행하게 된다.

기존의 동기식 순차 시스템(Synchronous Sequential System)의 분석과 설계는 표(table), 다이어그램 (Diagram) 그리고 하드웨어 모듈로 쉽게 바뀌어지는 표현식들 (Expressions)을 사용하여 상태 함수(State Function)들로 기술하여 왔다. 이런 방법은 입력과 출력 그리고 상태들(States)의 수가 적은 간단한 시스템에서 유용하였다.

그러나 입, 출력 상태의 수가 많고 표현식들을 쉽게 구할 수 없는 시스템들은 상태 함수를 쉽게 얻을 수 없으므로 이들 시스템을 기술하고 구현하기 곤란하다. 따라서 복잡한 디지털 시스템의 기능을 기술하고 이를 실현하기 위해 알고리즘의 개념이 도입되었다.

하드웨어 모듈로 직접 구현할 수 없는 어떤 복잡한 연산을 보다 간단한 연산들로 변환시켜 기술한 후에 하드웨어 모듈로 실제화하는 방법을 제시하였다[1].

알고리즘에 의한 기술 방법에는 구현할 시스템

*淸州大學校 電子計算學科
Dept. of Computer Science, Chong Ju University.
**淸州大學校 電子工學科
Dept. of Electronic Engineering, Chong Ju University.
論文番號 : 90-46(接受1989. 3. 14)

의 구조에 따라 사용될 특정 세부연산들의 종류와 세부 연산들의 동시성(Concurrency)에 따라 많은 등가 알고리즘들이 존재할 수 있으며, 또한 각각의 알고리즘에서는 여러 레벨(Level)이 존재할 수 있으므로 시스템을 구현하기 위한 알고리즘은 설계의 목적과 요구 사항들인 속도, 가격, 신뢰성, 크기, 전력 소모등을 만족시킬 수 있는 모든 가능한 사항들을 고려하여 선택되어져야 한다.

이런 방법으로 시스템 전체를 하나로 설명한다는 것이 어려울 경우 시스템을 각기 다른 기능을 수행하는 부시스템으로 나누어 이를 각각 기술하는 모듈 접근법 (Module Approach)으로 가능하게 되었다.

시스템이 복잡해지고 기능이 다양해짐에 따라 설계 시간의 단축, 신뢰도의 향상 및 설계 비용(Cost)의 절감을 위해 고안된 것이 컴퓨터를 이용한 설계 자동화(Design Automation) 방법이다[1 2 3].

본 논문에서는 복잡한 디지털 시스템의 데이터부 설계를 자동화하기 위한 것으로 고수준 알고리즘 기술문(High Level Algorithm Description Language)으로 부터 데이터 부시스템의 테이블을 얻기 위해 데이터부 시스템 번역기를 C언어로 실현하였으므로 어휘와 구문 분석시 Unix의 Lex와 Yacc를 사용하였다.

Ⅱ. 하드웨어 기술 언어

1. 개요

소프트웨어 설계자들은 알고리즘을 기술하기 위해 고급언어를 사용한다. 반면 하드웨어 설계자들은 하드웨어 동작이나 구조를 기술하기 위해 다른 형태의 언어를 사용하는데 이러한 언어를 하드웨어 기술언어(HDL)라고 한다[2 3 4 6]. 하드웨어 기술언어는 레지스터를 표시하거나 레지스터 내용에 대한 동작 제어 기능을 설명하는 기호이며, 디지털 시스템의 구조와 동작을 기술하기 위해 사용되는 언어로서 주로 학문적인 연구를 목적으로 설계되었다.

이들 기술언어는 하드웨어의 모든 동작특성을 쉽게 기술할 수 있고 구문이 간단하며, 이해 및 수정이 용이한 장점을 지니며 행태적으로 프로그램 언어와 유사하다.

디지털 시스템 설계언어의 개발단계는

a) 시스템을 알고리즘 레벨에서 기술할 수 있는 기술언어의 구조 및 언어를 설계하고

b) 알고리즘 레벨의 기술문을 Register Transfer 의 기술문으로 변환시켜 주기 위한 Translator의 개발

c) 레지스터 전송 레벨의 기술문으로 부터 데이터 부시스템 모듈과 제어 부시스템 모듈을 생성하는 모듈 번역기의 설계

d) 제어 부시스템 모듈로 부터 레지스터 전송 레벨의 설계방식을 결정해 주는 설계 방식 선택기의 설계

e) 시스템의 동작을 Simulation하기 위한 Simulator 개발

f) 제어 부시스템으로 부터의 제어 신호와 데이터 부시스템으로 부터의 조건과의 논리 관계를 결정짓는 논리식 생성용 하드웨어 컴파일러의 개발로 구분된다.

2. 기술언어의 요소[1, 8, 9, 10]

기술언어는 다음 세가지 사항을 고려하여야 한다.

가) 기능적인 측면 :
시스템의 기술목표는 수행하고자 하는 어떤 기능을 효과적으로 표현하는데 있으므로 기술문은 가능한한 많은 내용을 포괄적으로 함유함과 아울러 시스템을 다양하게 실현할 수 있도록 기술되어야만 한다.

나) 구조적인 측면 :
시스템의 구현을 위하여 기술문은 가능한한 구체적으로 구조적인 정보를 포함할 수 있어야 한다.

다) 설계 과정을 지원하는 측면 :
설계 과정은 여러번의 수정이 가해져야 하는 반복적인 과정이며 설계 목적이 여러가지의

제반 사항들을 만족시키는 적절한 시스템을 얻는데에 있으므로 반복 수정하는 과정에 있어서 수정되는 세세한 부분들을 지원할 수 있어야 한다.

3. 기술언어의 구성

(가) Data Objects

Hardware / Function 레벨에서 기본적인 대상물 형태(Object Type)는 비트벡터(Bit Vecter)이며 대문자나 문자에 밑줄을 그어 표시한다. 비트 벡터의 차원(Dimension)은 심볼 "〈"와 "〉" 사이에 기술하며 비트의 수가 된다.

각 비트는 우측에서 좌측으로 증가하며 표기한다.

예로서 A〈5〉 : =(A4, A3, A2, A1, A0)이고 X〈3〉 : =(X2, X1, X0)이다.

여기서, 심볼 ":="은 비트 벡터의 선언(Declare) 또는 재명명(Rename)시 사용된다. C:=B에서 C와 B는 같은 데이터 대상물(Data Object)에 대한 두 개의 다른 이름이다.

(나) 함수(Function)

실행 레벨에서 함수는 비트벡터를 가지고 기술한 연산자는 대문자열의 이름에 의해서 표시되며 인수(Argument)는 조합회로망 (Combinational Circuit)에 의해 구현된다.

(1) 조건적인 선택 함수일 경우 조건에 의해 적어도 한 대상이 선택된다.

A if a : B if b : C if c

여기서 A, B, C는 대상이고 a, b, c는 조건이다.

(2) 연결 (Concatenation) 함수는 큰 벡터를 형성하기 위해 벡터를 연결하며 다음과 같이 표시된다(A, B, C,…)

예로서 B : =(A5, A4,…A0)

B : =(B3, B2,…B0)일 때

D : =A+B

즉 D : =(A5, A4,…A0, B3, B2,…B0)

(3) 추출 (Extraction) : 벡터에서 부벡터(Subvecter) 추출은 요소의 연결로 표기한다. A : =(B4, B3, B2)

(다) 할당문 (Assignment)

할당문은 원시대상물 (Souce Objcets)을 목적대상물 (Destination Object)에 할당할 때 사용된다. 할당문에 대한 언어 구성은 D〈−S와 같이 쓸 수 있으며 레지스터 전송에 의해 실행된다. 조건 할당문은 조건문이 만족될 때만 할당문이 수행된다. 즉 if c then D〈−S에서 c가 만족되면 D〈−S를 수행한다.

(라) Comments

설명문은 알고리즘을 설명을 위해 프로그램 어디서나 자유롭게 사용되며 /*과 */ 사이에 기술된다.

Ⅲ. 하드웨어 컴파일러[6, 7, 8, 12]

1. 개 요

컴파일러는 입력으로 원시프로그램 (Source Program)을 받아서 같은 기능을 가진 기계 코우드의 프로그램인 목적프로그램 (Object Program) 으로 출력하게 되는데 보통 여섯단계인 어휘분석 (Lexical Analysis), 중간코우드 생성 (Intermdiate Code Generation), 코우드의 효율화 (Code Optimization), 코우드 생산(Code Generation) 과 이들 각 과정마다 관계하는 에러처리 (Error Handler)와 심볼 테이블 관리(Symbol Table Manager)로 구성된다.

일반적으로 하드웨어 컴파일러는 레지스터 전송 레벨이나 그 이상의 고수준에서 하드웨어 동작을 주 대상으로 기술한 입력을 받아 게이트 레벨의 기술문으로 변환하는 변환기 (Translator) 이다.

하드웨어 컴파일러는 어휘, 구분분석, 심볼테이블 관리, 그리고 에러처리까지는 소프트웨어 컴파일러와 동일하지만 소프트웨어 컴파일러가 중간코드를 생성시켜 최종적으로 기계어로 바꾸는데 반해 하드웨어 컴파일러는 중간코드를 생성시키지 않고 여러 단계의 변환을 행하여 레지스터간의 동작을 기술한 전송 테이블을 얻는다는 점이 다르다.

2. 렉시칼분석과 구문분석

(가) 렉시칼 분석

렉시칼 분석기의 기능은 원시 프로그램의 각 단어를 한 자씩 읽어서 토큰 (Token)이라고 하는 단위로 구성하는 것을 말한다.

입력문장에서 명칭(변수와 지정어), 연산자, 상수등을 분류하여 토큰표를 만들며, 명칭과 상수는 하나의 프로그램에 여러개 나오므로 이들은 구별하기 위해 심볼 테이블을 만드는데 이들 토큰표와 심볼 테이블을 이용하여 토큰을 만든다.

토큰을 만들기 위하여 토큰의 표를 찾는 방법은 많이 있으나 대개 유한 기계(Finite Automata) 을 많이 사용한다. 유한 기계는 유한군의 단계와 입력부호로 인하여 생기는 한 집단의 단계 전위로 구성된다.

(나) 구문 분석

구문분석은 파서(Parser)라고도 하는데 파서란 어떤 스트링 W가 문법 G에 의한 언어가 될 수 있다면 파서는 이 스트링 W를 입력으로 받아들여 출력으로는 나무구조를 만드는 프로그램을 말하는데 이 때 만일 스트링 W가 문법 G에 의한 문장이 아니면 입력 스트링 W는 허락되지 않으며 파서는 착오를 알려주는 일을 한다.

3. 테이블관리 및 에러처리

렉시컬 분석단계에서 원시 프로그램에 나타나는 명칭등에 대한 정보들은 심볼 테이블이라고 하는 데이터 구조로 구성되는데 이 심볼 테이블에 구성된 정보들은 컴파일하는 동안에 여러 단계에 걸쳐서 보완되기도 하며 이용되기도 한다.

테이블을 이용할 때에는 Value 값을 비교하여 이용하는데 만약 테이블에 부합되는 Value 값을 비교하여 이용하는데 이 때에는 에러 형과 기대되는 심볼의 값을 출력한다.

Ⅳ. 데이터부 번역기

알고리즘 기술문을 입력하여 하드웨어의 데이터부 시스템을 설계하기 위한 데이터부 테이블 생성기로서, 각각의 제어점에대한 데이터부의 생성 테이블을 생성토록 설계하였다.

1. 기술 언어의 구조

형태적으로 프로그램 언어와 유사하며, 크게 알고리즘 표제부문과 선언부문, 몸체부분으로 나눌 수 있다.

첫째, 알고리즘 표제 (Algorithm Heading) 부분으로 알고리즘 기술문임을 나타내는 " ALGORITHM" 과 Otherid으로 이루어진다.

둘째, 선언 부분으로 입력분수와 출력 변수의 형태(TYPE)을 정의한다.

셋째, 몸체 부분으로써 표지(Label)와 명령문 집합으로 이루어진다.

2. 알고리즘 기술문의 규칙

알고리즘 표제 (ALGORITHM HEADING) : 프로그램의 시작에 "ALGORITHM"이라 기술하여 일반 프로그램과 다른 언어임을 의미했고, 다음에 프로그램의 식별자인 이름을 기술하였으며, ":"로 프로그램의 표제가 끝났음을 표시했다.

알고리즘의 표제는 다음과 같이 기술한다.

ALGORITHM Otherid :

선언부(Declaration Part) : 선언 부분은 입력변수와 출력변수의 형을 정의한 부분으로서 " inputs" 와 "outputs"로 나누었고, 각 변수와 형의 나열은 ","로 구분하였으며, 입력변수와 출력변소의 끝은 ";"으로 구분하였다.

선언부는 다음과 같이 기술한다.

```
inputs a, b, x : fraction,
         s     : boolean ;
outputs c      : fraction :
```

몸체부(Body part) : 몸체부분은 각 명령을 기술한 집합(Block) 으로서 "begin"으로 시작하여 "end."으로 끝난다.

몸체의 문장은 명령문으로 구성되는데 명령문 사이 ","은 동시 동작을 의미하고 ";"은 순차동작을 의미한다. 몸체부 내부에서 또다른 명령문은 "begin"으로 시작하여 "end;"로 끝난다.

몸체부는 다음과 같이 기술한다.

```
begin
  wait : a=1, b=1, if's then wait ;
  compute : c=0, n=0 ;
  while (n<10) do
  begin
    c=(a*b) / (a+b),
    n=n+1 ;
  end ;
  goto wait ;
end.
```

식별자 (Identifier) : 모듈(Module)의 이름으로써 영문자나 숫자의 조합으로 이루어지며, 첫번째 문자는 반드시 영문자이어야 한다.

명령문 (statement) : 디지털 시스템의 실제 동작을 기술한 것으로서 고수준 언어인 반복문을 포함할 수 있고, 시스템에서의 동시 동작도 포함할 수 있다.

대입문 (Assignments) : 디지털 시스템의 동작을 나타내는 명령문으로써 "="로 나타내었다.

설명문 (Comments) : 알고리즘에 대한 설명문을 삽입하여 프로그램의 이해를 도울 수 있으나, 고 수준의 언어나 알고리즘 기술문에서는 명령문 그 자체가 설명문이 될 수 있으므로 생략할 수 있다.

제어문 (Control Statement) : 일반 프로그램 파스칼과 C의 형태를 사용하였으며 데이터 부 번역기의 입력으로 사용할 때는 While문, Do While문,If문, GoTo문을 고려하였다.

연산자 (Operators) : 연산자로는 산술 연산자와 관계 연산자를 사용하며, 산술 연산자에는 +, −, *, /, ||, ( ) 등을 사용하고, 관계 연산자에는 <, >, =, <=, >=, <> 등을 사용한다.

표 2. 알고리즘 기술문의 문법
Rules of algorithmic description language

```
<program>       : <program heading> <block1>
<program heading> : ALGORITHM Otherid
<block1>        : <declare1> <block2>
<declare>       :
                | <declare11> ';'
<declare11>     : <declare11> ';' <declare12>
                | <declare12>
<declare12>     : INPUTS <declare2>
                | OUTPUT <declare2>
<declare2>      : <declare2> ';' <declare3>
                | <declare3>
<declare3>      : <declare4> ';' <declare5>
<declare4>      : <declare4> ';' Otherid
                | Otherid
<declare5>      : FRACTION
                | BOOLEAN
<block2>        : BEGIN <block21> END '.'
<block21>       :
                | <block22> ';'
<block22>       : <block22> ';' <block3>
                | <block3>
<block3>        : <block33> ';' <block3>
                | LABEL <block33>
<block33>       : <block33> ',' <block4>
                | <block4>
<block4>        : WHILE '('<expression>')' <block5>
                | IF    '('<expression>')' <block5>
                | DO <block5> WHILE '('<expression>')'
                | GOTO Otherid
                | <statement>
<block5>        : <block4>
                | BEGIN <block22> END
<statement>     : Otherid '=' <expression>
```

```
<expression>  : '(' <expression> ')'
              | '|' <expression> '|'
              | <expression> '<' <expression>
              | <expression> '>' <expression>
              | <expression> '=' <expression>
              | <expression> LE <expression>
              | <expression> GE <expression>
              | <expression> '+' <expression>
              | <expression> '-' <expression>
              | <expression> '*' <expression>
              | <expression> '/' <expression>
              | Otherid
              | NUM
```

3. 데이터부 번역기의 설계

데이터부 번역기 설계는 구문 분석기 YACC에 입력되는 알고리즘 문법 원시프로그램과 YACC 에서 출력된 프로그램을 포함하는 주 프로그램으로 설계하였으며, 데이터부 번역기의 개략도는 그림 1과 같다.

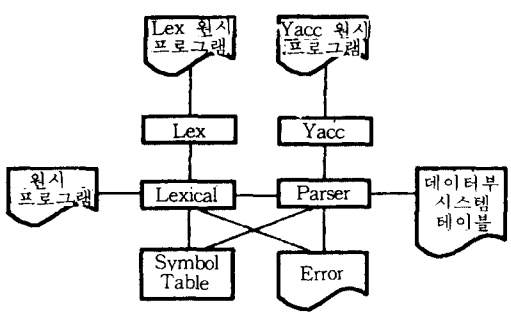


그림 1. 데이터부 번역기의 흐름도
Flowchart of Data Subsystem Translator

4. 데이터부 번역기의 실행 예

데이터부 번역기의 결과를 출력하기 위해 다음과 같은 Newton Rapson iterative algorithm을 입력으로 사용하였다. 이예는 P. G. PAULIN [12]이 제시한 것이다. 데이터부 테이블 번역기의 입력 프로그램은 아래와 같다.

```
ALGORITHM diffeq ;
   input  a, dx, x, u, y : fraction;
   output yy             : fraction;
begin
  a=23, dx=34, x=53, u=5, y=64;
while (x<a)
  {
   x1=x+dx;
   u1=u-5*u*x*dx-3*y*dx:
   y1=y+u*ux;
   x=x1 ; u=u1; yy=y1
  };
end .
```

이 알고리즘의 실행 결과인 최종 출력(데이터부 시스템 테이블)은 아래와 같다.

《** End of Algorithm Interpreting **》

Destination	operator	source1	source2	label
a	<-	23		, stat0
dx	<-	34		, stat1
x	<-	53		, stat2
u	<-	5		, stat3
y	<-	64		; stat4
tcc8	LT	x	a	; stat5
x1	ADD	x	dx	; stat6
tcc4	MUL	5	u	; stat7
tcc4	MUL	tcc4	x	; stat8
tcc4	MUL	tcc4	dx	; stat9
tcc6	SUB	u	tcc4	; stat10
tcc4	MUL	3	y	; stat11
tcc4	MUL	tcc4	dx	; stat12
u1	SUB	tcc6	tcc4	; stat13
tcc4	MUL	u	dx	; stat14
y1	ADD	y	tcc4	; stat15

```
x       <-      x1          ; stat16
u       <-      u1          ; stat17
yy      <-      y1          ; stat18
*****************************************
```

최종 출력 결과를 P. G. PAULIN이 제안한 논문의 그래프〈그림 2〉와 비교하기 위해 그래프를 그리면 그림 3과 같이 된다. 그림 2와 3의 두 그래프는 논리적으로 같음을 볼 수 있다.

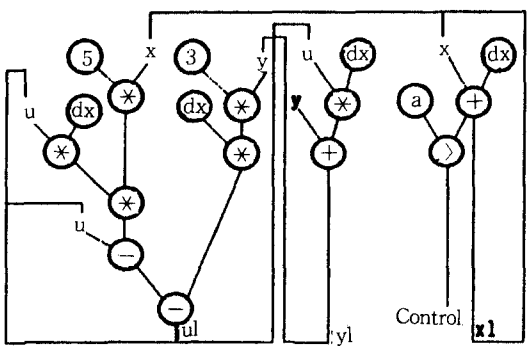


그림 2. 데이터 흐름 그래프(Paulin의 예)
Data Flow Graph(example of the Paulin)

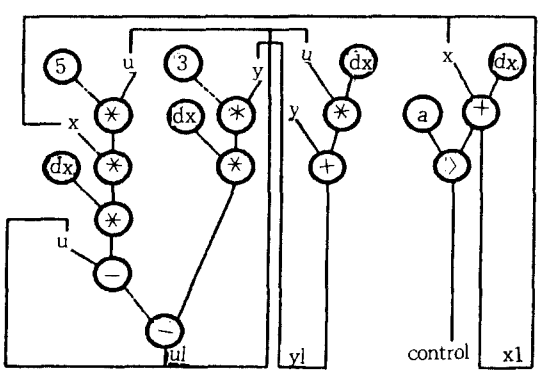


그림 3. 데이터 흐름 그래프(실행 예)
Data Flow Graph(example of the execution)

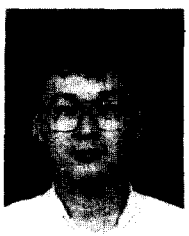
V. 결 론

본 연구에서는 데이터부 번역기를 C－언어로 설계하였으며, 이를 이용하여 알고리즘으로 작성된 기술문을 입력하여 데이터 부시스템에 대한 테이블을 얻었다. 디지털 시스템의 데이터 부시스템을 설계하였는데, 데이터부 번역기를 이용함으로서 데이터 부시스템을 설계할 수 있는 테이블을 얻을 수 있음을 알 수 있었고, 또한 본 논문의 결과를 이용하여 그래픽 라이브러리(Graphic Library)를 연결하면 데이터부의 회로도 얻을 수 있을 것이고, 제어부 번역기의 제어신호와 연결되면 하나의 완전한 디지털 시스템의 회로도를 그려낼 수 있는 방법에 도달할 수 있을 것으로 본다.

參 考 文 獻

1. Milos D. Ercegovac and Tomas Lang, "Digital Systems and Hardware / Firmware Algorithms", John Willey & Sons, 1985, pp. 159~169, pp. 393~439, pp. 469~596.
2. John P. Hayes "Digital System Design and Microprocessors", Mcgraw-Hill, 1984, pp. 301~322.
3. Subrata Dasgupta, "The Design and Description of computer Architectures", John Willey & Sons, 1984, pp. 40~61.
4. Alice C. Parker, "Automated Aynthesis of Digital System", IEEE D & T. 1984, pp. 75~81.
5. A. K. Singth, "Descriptive Technique For Digital System Contaning Complex Hardware Component", Ph. D. Dissertation, Kansas Stat Univ. 1981.
6. Michell Waite, "Stephan Prata and Donald Martin, "C Primer Plus", Howard W. Sans & Co. 1985.
7. Roland C. Backhouse, "Syntax of Programming Languages", C. A. R. HOARE, 1978, pp. 158~172.
8. M. R. barbacci, "A Comparison of Registe Transfer Language for Dicribing computers and Digital System", IEEE Trans. Comput. Vol. C-24, 1975, pp. 137~150.
9. Sajjan G. Shiva, "Computer Hardware Description Language- A Tutorial", Proccedings of IEEE, Vol. 67. No. 12, 1979.
10. Sajjan G. Shiva, "Computer Description and Architecture", Little brown and Company, 1985.

11. Axel T. Schreiner, "Introduction to Compiler Construction with Unix", Prentice-Hall.
12. P. G. Paulin, J. P. Knight, E. F. Circzyc, "A Multi-Paradigm Approch to Automatic Data Synthesis", IEEE, 1986.



李 明 浩(Myung Ho LEE) 正會員
1956年 7 月 2 日生
1975年 3 月～1979年 2 月：光云工科大學 電子通信學科卒
1979年 3 月～1981年 2 月：延世大學校 大學院 電子工學科(工學碩士)
1981年 3 月～1987年 8 月：延世大學校 大學院 電子工學科 博士課程修了
1983年 3 月～現在：清州大學校理工大學 電子計算學科 助教授

李 近 萬(Keun Man YI) 正會員
1949年 9 月 18日生
1969年 3 月～1973年 8 月：漢陽大學校 電子工學科卒
1978年 3 月～1980年 2 月：漢陽大學校 大學院 電子工學科卒(工學碩士)
1984年 3 月～1987年 8 月：漢陽大學校 大學院 電子工學科 博士課程修了
1980年 3 月～現在：清州大學校理工大學 電子工學科 副教授