論文97-34C-3-3

2바이트 코드워드 표현방법에 의한 자료압축 알고리듬

(Data Compression Algorithm with Two-byte Codeword Representation)

梁 榮 日 *, 金 導 鉉 *

(Yeong-Yil Yang and Do-Hyun Kim)

요 약

본 논문에서는 *Lempel-Ziv* 자료압축 알고리듬의 하드웨어 구현을 위한 새로운 코드워드 표현방법을 제시하였다. 기존의 표현방법에서는 3 바이트(*last symbol, position*과 *matched length*)로 이루어진 코드워드를 생성한다. *Last symbol*의 *MSB*는 *compression flag*이고, 나머지 7 비트는 *character*를 나타낸다. 본 논문에서는 *matched length*의 크기를 256 대신에 128로 제한하여, *matched length*를 7 비트로 표현하였다. 이 표현방법에서, 코드워드는 2 바이트(*merged symbol*과 *position*)로 구성되어 있다. *Merged symbol* 의 MSB는 자료의 압축여부를 나타내는 *compression flag*이고, *merged symbol*의 나머지 7 비트는 *compression flag*의 값에 따라 *character* 또는 *matched length*를 나타낸다. 제안한 코드워드 표현방법은 기존의 표현방법에 비해 압축률이 5% 증가되었고, 기존에 개발된 압축과 복원 하드웨어구조로 구현이 가능하다. 그리고, 압축률 증가요인을 분석하였다.

Abstract

In this paper, the new data model for the hardware implementation of Lempel-Ziv compression algorithm was proposed. Traditional model generates the codeword which consists of 3 bytes, the last symbol, the position and the matched length. MSB (Most Significant Bit) of the last symbol is the compression flag and the remaining seven bits represent the character. We confined the value of the matched length to 128 instead of 256, which can be coded with seven bits only. In the proposed model, the codeword consists of 2 bytes, the merged symbol and the position. MSB of the merged symbol is the compression flag. The remaining seven bits represent the character or the matched length according to the value of the compression flag. The proposed model reduces the compression ratio by 5% compared with the traditional model. The proposed model can be adopted to the existing hardware architectures. The incremental factors of the compression ratio are also analyzed in this paper.

I. 서 론

원래 자료로부터 작은 크기의 자료로 표현하여 자료를 압축하고, 압축된 자료로부터 원래 자료로 복원하는 방법을 자료압축/복원이라 한다. 자료를 압축함으로써 자료의 저장비용 또는 전송비용을 줄일 수 있다. 압축된 정보표현으로부터 자료가 복원될 때, 원래 자료가 완전히 복원되는 비손실압축방법 (lossless compression technique)과 원래 자료가 완전히 복원되지 않는 손실압축방법 (lossy compression technique)이 있다. 대표적인 비손실자료압축 알고리듬으로는 Huffman coding[1], run-length encoding[2], multi-group compression method[3], Lempel-Ziv(LZ) 알고리듬[4] 등이 있다. Ziv 와 Lempel[4]에의해 제안된 비손실자료압축 알고리듬은 하드웨어 구현에 적합하다. 자료압축시 반복되는 구(phrase)에 대하여 구가

* 正會員, 慶尙大學校 電子材料工學科

(GyeongSang National University, Dept. of Electronic Materials Engineering)

接受日字:1996年11月8日, 수정완료일:1997年3月7日

존재하는 곳에 포인터를 지정하고, 자료복원시 각각의 포인터위치에 이미 존재하는 디코드텍스트의 구조로 대치하는 방법을 개선한 LZ 알고리듬들이 연구되었다. [5,6]

LZ 자료압축 알고리듬을 하드웨어로 구현하는 방법들이[7,8] 연구되었다. 자료압축을 위한 하드웨어는 자료의 저장과 전송비용을 감소시키므로 터미날과 디스크 제어기에 적합하며, 또한 자료압축을 구현하는 VLSI 칩은 분산 네트웍을 위한 통신 제어장치에 매우 적합하다. 기존의 방법에서 제시된 하드웨어 구조는 고속과 고성능 출력을 얻기 위하여 systolic 형태의 pipeline과 병렬구조로 되어있고, 3 바이트 (*last symbol, position* 과 *matched length*)로 이루어진 코드워드를 생성한다. 제안된 코드워드는 2 바이트 (*merged symbol*과 *position*)로 구성되어 있다. *merged symbol* 의 MSB는 자료의 압축여부를 나타내는 *compression flag*이고, *merged symbol*의 나머지 7 비트는 *compression flag*의 값에 따라 *character* 또는 *matched length*를 나타낸다. 이 논문은 LZ 압축 알고리듬의 하드웨어 구현에서 2 바이트로 이루어진 새로운 코드워드 표현방법을 제시하였다. 제안한 코드워드 표현방법에서는 *last symbol*과 *matched length*가 한 바이트로 표현되고, 압축률이 향상된다.

제 2 장에서는 논문에서 사용되는 용어에 대하여 정의하였고, 제 3 장에서는 기존의 3바이트 코드워드 표현방법과 제안된 2바이트 코드워드 표현방법을 설명하였다. 2바이트 코드워드 표현방법을 사용하는 압축 알고리듬이 제 4 장에서 기술되었다. 제 5 장에서 실험결과와 압축증가요인을 분석하였고, 제 6 장에서 결론을 내렸다.

Ⅱ. 용어의 정의

이 장에서는 논문에서 사용되는 용어에 대해 정의한다.

정의 1) *character* : 적어도 1개 이상의 요소로 이루어진 *string S*를 구성하는 각 요소

정의 2) 스트링(*string*) : 일련의 연속된 *character*로, 한정된 길이를 가지고 있다.

정의 3) 서브스트링(*substring*) : 스트링의 일부분.

자료압축과정에서 두 *string*이 사용되는데, 그림 1은 *string*을 저장하되는 *buffers, first buffer*와 *second buffer*를 보여준다. *Second buffer*는 압축과정에서 압축이 될 *string*을 저장하고, *first buffer*는 *second buffer*의 *string*과 비교가 이루어질 *string*을 저장하고 있다. 이 예에서, *first buffer*의 크기는 8이고, *second buffer*의 크기는 4이다. 그리고, 두 *buffer*는 이중선으로 분리되어있다. *First buffer*는 *string F = abcbcabb*를 저장하고 있고, *second buffer*는 *string S = bcaa*를 저장하고 있다. *Buffer*위에 표시된 숫자는 각 *buffer* 내에서 주소를 나타낸다.

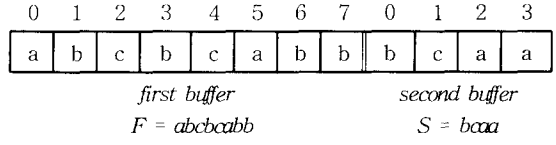


그림 1. 자료압축시 사용되는 *string* 저장을 위한 *buffers, First buffer*는 크기가 8이고, *second buffer*는 크기가 4이다. 두 *buffers* 는 이중선으로 분리되어있다

Fig. 1. *First* and *second buffer* for storing the strings, whose size is 8 and 4, respectively. Two buffers are separated with the double line.

정의 4) *matched string*과 *matched length* : *second buffer*의 주소 0에서 시작하는 *string S*의 *substring*이 *string F*의 *substring*과 일치하는 *substring*들이 존재하면, 이 *substring*들 중에서 가장 많은 *character*로 이루어진 *substring*을 *matched string*이라 하고 *matched string*의 *character* 수를 *matched length*라 한다.

정의 5) *position* : 주어진 *matched string*에 대하여, *first buffer*에서 *matched string*이 시작하는 주소

예) 그림 1에서 *character* set은 { *a* , *b* , *c* } 이고, *string F = abcbcabb* 이며, *string S = bcaa* 이다. *Substring bc* (*first buffer*의 주소 1에서 시작하는 *substring*)와 *bca*(*first buffer*의 주소 3에서 시작하는 *substring*)는 *srting S*의 *substring*과 일치하고, 길이가 긴 *bca*가 *matched string*이 된다. 그러므로, *matched length*는 3 이고, *position*은 3이 된다.

정의 6) *last symbol* : *matched string*이 존재하는 경우, *second buffer*에서 *matched string*의 마

지막 *character* 다음 주소의 *character*이다. 그림 1에서 *matched string*이 *bca*이므로 주소 3 위치의 *character a*가 *last symbol*이 된다.

정의 7) 압축비(compression ratio) 또는 압축률(% compression) : 압축으로 자료가 감소된 정도를 표현하며, 다음과 같이 정의된다.

$$\text{압축률} = \frac{\text{줄어든 } character \text{ 수}}{\text{원래 자료의 } character \text{ 수}} \times 100(\%) \qquad (1)$$

III. 코드워드(codeword) 표현방법

이 장에서는 *matched string*에 대해 코드워드를 생성하는 방법을 설명한다. 코드워드 디자인은 버퍼 크기의 선택과 밀접한 관계가 있다. 또한 압축률은 버퍼 사이즈에 따라 변하는데, 일반적으로 버퍼 사이즈가 클수록 압축률은 증가한다. 버퍼 사이즈를 256으로 할 경우 압축률을 효과적으로 얻을 뿐 아니라 하드웨어 구현에서 적합한 크기이다.[7] 여기서는 버퍼 사이즈가 256일 때 코드워드 표현방법을 기술한다. 3바이트로 표현되는 기존의 코드워드 표현방법을 설명한 후, 제안된 2바이트 코드워드를 설명하며 두 코드워드 표현방법을 비교한다.

1. 3바이트 코드워드 생성

코드워드는 *last symbol, position*과 *matched length*의 3 바이트로 구성된다. 그림 2의 예에 대해 코드워드를 생성하여 보자. 이 예에서 *first buffer*에서 주소 5에서 시작하는 *substring is_the_* 이 (_는 space를 나타낸다.) *matched string*이라 하자. 이 경우 *matched length*는 7 이고 *last symbol*은 *c* 이다. *Buffer* 크기가 256 이므로 *position*과 *matched length*를 표현하기 위하여 각각 한 바이트가 필요하다. ASCII 코드는 7 비트로 되어 있고, parity 비트로 사용되는 MSB는 *compression flag*로 사용된다. MSB가 1이면 압축이 되었음을 나타내고, MSB가 0이면 압축이 이루어지지 않았음을 나타낸다. 그림 2(b)는 생성된 코드워드를 보여준다. 각 바이트는 순서대로 *last symbol, position*과 *matched length*.이다. *last symbol*의 *compression flag*인 MSB는 1로 표시되고, 나머지 7 비트는 *last symbol*의 ASCII 코드 *c*를 나타낸다. *position*과 *matched length*를 나타내기 위하여 5와 7을 나타내는 두 바이트가 생성된다. 만일 *matched string*이 없는 경우, 그림 2(c)처럼 MSB를 0으로 표시되고, *second buffer*의 첫 *character i*의 ASCII 코드 *i*로 이루어진 바이트를 생성한다.

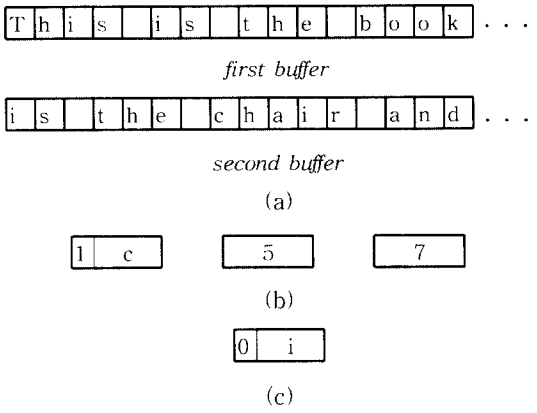


그림 2. 3바이트 코드워드 표현방법 (a) 크기가 256인 first buffer와 second buffer (b) (a)로부터 생성된 코드워드, (c) matched string이 없을 경우 생성된 코드워드

Fig. 2. 3-byte codeword representation (a) first and second buffers whose size is 256, respectively, (b) the codeword representation resulting from (a), and (c) the codeword representation when there is no matched string.

2. 2바이트 코드워드 표현방법

압축이 이루어질 자료를 살펴보면 *matched length*가 125 이상일 가능성은 거의 없다. 제안한 방법에서는 *second buffer* 크기를 *first buffer* 크기의 반으로 설정하였다. 기존의 3바이트 자료압축 알고리듬에서 *matched length* 최대값이 256인 반면, 2바이트 코드워드 표현방법에서는 *matched length* 최대값은 128 이므로 *matched length*는 7 비트로 표현이 가능하다. 코드워드를 이루는 2 바이트는 각각 *merged symbol*과 *position*이라 정의한다. 그림 3(a)에서 *first buffer*에서 주소 5에서 시작하는 *substring is_the_* 이 (_는 space를 나타낸다.) *matched string*이라 하자. *matched length*는 7 이고 *last symbol*은 *c* 이다. 그림 3(b)는 생성된 코드워드를 보여준다. *merged symbol*의 MSB는 *compression flag*로 1로 표시되고, *matched length*는 7 비트로 표현된다. *position*은 7을 나타낸다. 2바이트 표현에서는 *last symbol* (*character c*) 이 코드워드에 사용되지 않아, 다음 압축과정에서 *matched string*의 *character*로 이용될 수

있다. 만일 *matched string*이 없는 경우, 그림 3(c)처럼 MSB를 0으로 표시되어 압축되지 않았음을 나타내고, *second buffer*의 첫 번째 *character i*의 ASCII 코드로 이루어진 바이트가 생성된다.

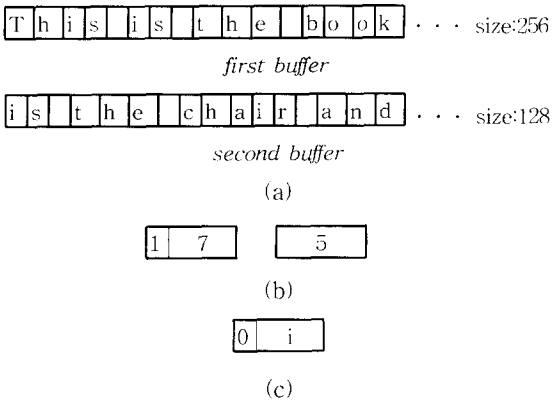
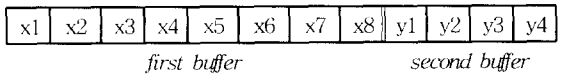


그림 3. 2바이트 코드워드 표현방법 (a) 크기가 256인 *first buffer*와 크기가 128인 *second buffer* (b) (a)로부터 생성된 코드워드, (c) *matched string*이 없을 경우 생성된 코드워드

Fig. 3. 2-byte codeword representation (a) *first* and *second* buffers whose size is 256 and 128, respectively, (b) the codeword representation resulting from (a), and (c) the codeword representation when there is no *matched string*.

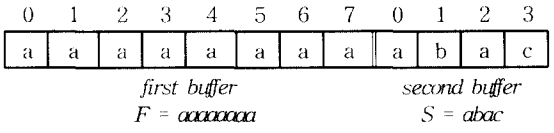
Ⅳ. 압축 알고리듬

이 장에서는 2바이트 코드워드를 생성하는 수정된 Lempel-Ziv 압축 알고리듬을 설명한다. 알고리듬 1과 그림 4는 *first buffer* 크기가 8이고 *second buffer* 크기가 4일 때 압축 알고리듬을 설명하고 있다. *x1, x2, …, x8*은 *first buffer*에 저장된 *string* 이고, *y1, y2, y3, y4*는 *second buffer*에 저장된 *string* 이다. 그림 4는 *character a, b*와 *c* 로 이루어진 *string D = abacbacbba*가 압축될 때 *buffer*의 내용을 보여주고 있다. 압축을 수행하기 위하여 *buffers*를 초기화한다. 그림 4(a)는 초기화된 *buffer*의 내용을 보여준다. *First buffer*를 *a*로 초기화하고 (*a*를 초기화 *character*로 가정한다.), *second buffer* 는 *string D*의 앞 4개의 *character*로 초기화한다. 알고리듬 1에서 **for 루프**는 *matched string*을 구하기 위해 *first buffer*와 *second buffer*를 비교하는 과정이다. 즉, *second buffer*에서 주소 0에서 시작하는 *substring*과 *first buffer*의 *substring*과 비교하여 일치하는 *substring*을 찾고, 이들 중에서 가장 많은 *character*로 이루어진 *substring*을 구한다. 그림 4(a)의 경우 *a*가 *matched string*이 되고, *matched length*는 1이다. *position*은 0, 1, 2, 3, …, 7 인데, 같은 크기의 *matched string* 이 여러개인 경우 가장 큰 주소인 7이 선택된다. 알고리듬 1에서 **if 문**은 비교결과에 따라 코드워드를 생성하고 *buffers*의 내용을 갱신한다. 생성된 코드워드는 2바이트이므로, *matched length*가 3보다 적으면 압축할 필요가 없다. 만일 *matched length*가 2보다 크면 3장에서 설명한 방법으로 코드워드를 생성하고 *buffer*를 갱신한다. 코드워드를 생성할 필요가 없으면 *buffer*의 내용만 갱신된다. *Buffers*는 오른쪽에서 왼쪽으로 정해진 수 만큼 시프트가 일어난다. 시프트가 될 때, *second buffer*에서 주소 0의 *character*는 *first buffer*의 마지막 주소로 시프트된다. 그림 4(a)의 경우 코드워드는 생성되지 않고 시프트만 한 번 이루어진다. 그림 4(b)는 갱신된 다음 *buffer*의 내용을 보여준다. *Second buffer*의 첫 *character a*가 *first buffer*로 시프트되고, *second buffer*의 주소 3에는 *string D*에서 *abac* 다음 *character*인 *b*가 시프트된다. 새로 갱신된 *buffer*에 대하여 위의 과정을 반복한다. 이 과정은 *string D*의 마지막 *character*가 *second buffer*에서 *first buffer*로 시프트가 이루어질 때 까지 반복한다.



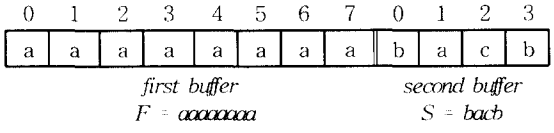

```
maxlength=0 ; pointer=0;
for i:=1 to 8
  index=0; length=0
  for j=1 to 3
     if x_index=y_j then length:=length+1;
                    else break;{exit for loop}
     index=index+1;
  endfor;
  if length>maxlength then
       maxlength=length ; pointer:=i;
endfor ;
if maxlength>2 then
  generate codeword;
  shift out maxlength character
else
   shift out 1 character
endif

String 끝까지 위과정의 반복 수행
```

알고리듬 1 : 2바이트 코드워드 생성 자료압축 알고리듬
Algorithm 1 : Compression algorithm with 2-byte codeword representation.



압축될 *string D = abacbacbba*

(a)



(b)

그림 4. 자료압축과정 예 (a) *buffer*의 초기화 (b) 한 번 시프트가 된 후 *buffer*의 내용
Fig. 4. The example of the data compression (a) shows the initial *buffers* and (b) shows the *buffers* after shifting 1 byte to the left direction.

V. 실험결과

제안한 2바이트 코드워드로 표현했을 때와 3바이트 코드워드로 표현했을 때 압축률을 비교하여 보았다. 표 1은 실험결과를 보여주는데, text1과 text2 실험예제는 UNIX 명령어 설명이고 text3과 text4 실험예제는 일반 교재의 일부분이다. 2바이트로 코드워드 표현했을 경우 압축률이 약 5% 정도 증가함을 알 수 있었다.

표 1. 압축률 비교
Table 1. Comparison of compression ratio.

파일 이름	파일 크기 (Byte)	압축률(%)	
		3-Byte 코드워드	2-Byte 코드워드
TEXT1	3309	35.69	41.16
TEXT2	87368	27.63	34.05
TEXT3	41696	20.47	26.16
TEXT4	29670	21.03	27.73

제안한 코드워드 표현방법에 의해 압축률이 증가되는 요인은 다음 두가지로 분류된다. *Second buffer*는 S_1 = *abcdasdabcd* · · · 이고 *matched string*이 *abc*인 경우를 살펴보자. 2바이트 코드워드로 압축하였을 때, *buffer*의 내용이 갱신된 후의 *second buffer*는 S_2 = *dabcd* · · ·이 된다. 3바이트 코드워드로 압축하였을 때 *last symbol*은 *d*가 되고, 내용이 갱신된 후의 *second buffer*는 S_3 = *asdabcd* · · ·이 된다. 이 때 S_2의 *matched string*이 *dab*이고, S_3의 *matched string*이 *ab*인 경우를 살펴보자. 이 경우는 3바이트 코드워드로 압축방법으로는 압축이 되지 않으나, 2바이트 코드워드 표현방법에서는 압축이 이루어지는 경우로, type 1으로 정의한다. Type 2의 경우를 살펴보자. 2바이트 코드워드로 압축하였을 때, *buffer*의 내용이 갱신된 후의 *second buffer*는 S_2 = *dabcd* · · ·이 된다. S_2의 *matched string*이 *dabcd*이고, S_3의 *matched string*이 *abcd*인 경우를 살펴보자. 이 경우에서 모두 압축이 이루어지나 3바이트 코드워드로 표현방법에서 *matched length* 3이나, 2바이트 코드워드 표현방법에서의 *matched length*는 4이므로, 압축이 한 *character* 더 이루어진 경우이다. 압축율 증가의 90%는 type 1의 요인에 의한 것을 알 수 있었다.

표 2. 요인에 따른 압축증가율
Table 2. Analysis of the increment of the compression ratio according to incremental factors.

파일 이름	압축증가율(%)		
	Type 1	Type 2	증가압축률
TEXT1	4.403	1.021	5.462
TEXT2	5.857	0.355	6.426
TEXT3	5.418	0.368	5.786
TEXT4	4.702	0.600	5.302

VI. 결 론

이 논문에서는 LZ 자료압축 알고리듬을 하드웨어로 구현할 때 2 바이트 (*merged symbol*과 *position*)로 이루어진 새로운 코드워드 표현방법을 제시하였다. *Second buffer*의 크기를 128로 제한하여, *matched length*를 7 비트로 표현하였다. *Merged symbol*의 MSB인 *compression flag*가 1이면 나머지 7 비트는 *matched length*를 나타내고, 0이면 나머지 7 비트는 ASCII *character*를 나타낸다. *Position*은 *first buffer*의 주소를 나타내는 바이트이다.

제안한 표현방법을 컴퓨터 시뮬레이션을 UNIX명령

어 설명등의 파일에 대해 수행한 결과, 제안된 알고리듬은 3바이트로 표현방법에 비해 압축률이 5% 증가되었고, 기존에 개발된 압축과 복원 하드웨어구조로 구현이 가능하다.

참 고 문 헌

[1] D. Huffman, "A method for the construction of minimum redundancy codes," *Proc. IRE*, vol. 40, pp. 1098-1101, 1952.

[2] S. Golomb, "Run-length encoding," *IEEE Trans. Inform. Theory*, vol. IT-12, pp. 399-401, July, 1966.

[3] M. Bassiouni, A. Mukherjee, and N. Ranganathan, "Encoding arithmetic and tree-based coding," *J. Inform. Processing. Manage.*, vol. 25, no. 3, pp. 293-305, 1989.

[4] J. Ziv and A. Lempel, "A Universal Algorithm for Sequential Data Compression," *IEEE Trans. Inform. Theory*, vol. 23, pp. 337-343, 1977.

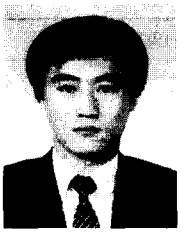
[5] T. C. Bell, I. H. Witten, and J. G. Cleary, "Modeling for Text Compression," *ACM Comput. Surveys*, vol. 21, no. 4, pp. 557-591, Dec. 1989.

[6] E. Fiala and D. Greene, "Data Compression with Finite Windows," *Commun. ACM*, vol. 32, no. 4, pp. 490-505. Apr. 1989.

[7] N. Ranganathan and S. Henriques, "High-speed VLSI designs for Lempel-Ziv-based data compression," *IEEE VLSI Algorithms and Architectures; Advenced Concepts*, pp. 255-265, 1993.

[8] Tae Young Lee, Seung Hyun Nam, Moon Key Lee and Yong Surk Lee, "A High Throughput VLSI Architecture For LZ Algorithm," *J. of the Research Institute of ASIC Design,* vol. 2, pp. 48-57, no. 1, Feb. 1995.

저 자 소 개



梁 榮 日(正會員)

1983년 2월 경북대학교 전자공학과 졸업(학사). 1985년 한국과학기술원 전기 및 전자공학과 졸업(석사), 1989년 한국과학기술원 전기 및 전자공학과 졸업(박사). 1990년 ~ 현재 경상대학교 전자재료공학과 부교수. 1994년 1월 ~ 1995년 1월 UC, Irvine 교환교수. 경상대학교 항공기부품기술연구센터 연구원. 주관심분야는 VLSI& CAD, 영상신호처리등.

金 導 鉉(正會員)

1970년 4월 11일생. 1996년 2월 경상대학교 전자재료공학과 졸업(학사). 1996년 3월 ~ 현재 경상대학교 전자재료공학과 석사과정. 주관심분야는 영상신호처리임