



(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2009-0077602
(43) 공개일자 2009년07월15일

(51) Int. Cl.

H04N 7/24 (2006.01)

(21) 출원번호 10-2008-0003638

(22) 출원일자 2008년01월11일

심사청구일자 없음

(71) 출원인

주식회사 휴맥스

경기 용인시 유방동 212-1번지 휴맥스빌딩

(72) 발명자

이충구

인천 부평구 부평1동 동아아파트 15동 304호

장의선

서울 성동구 성수1동 강변건영아파트 105-1504

김현규

서울 은평구 응암1동 118-27 501호

(74) 대리인

특허법인이지

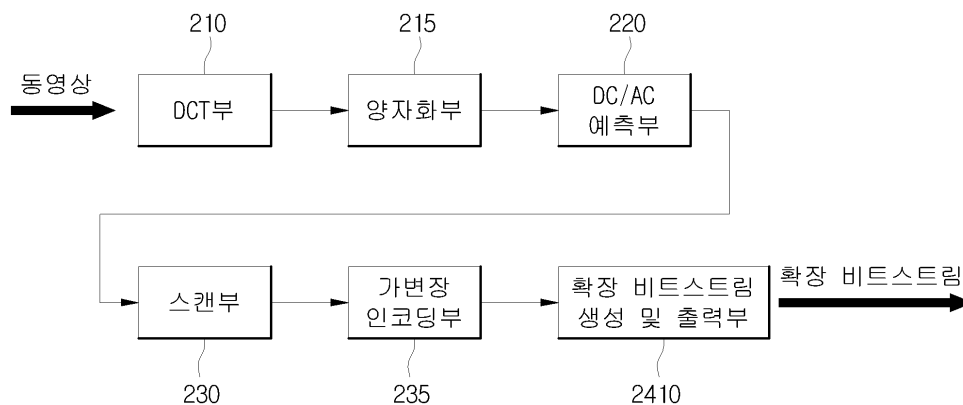
전체 청구항 수 : 총 21 항

(54) 동영상 데이터의 부호화/복호화 방법 및 장치

(57) 요약

동영상 데이터의 부호화/복호화 방법 및 장치가 개시된다. 복호화 장치는 기능부들을 저장하는 톨박스; 압축된 디코더 디스크립션을 입력받아 CDDL 기반 디코더 디스크립션을 복원하는 복원부; 상기 CDDL 기반 디코더 디스크립션을 XML 기반 디코더 디스크립션으로 변환하는 디코더 디스크립션 해석부; 상기 XML 기반 디코더 디스크립션을 이용하여 추상적 복호 모델(ADM)을 생성하는 ADM 생성부; 및 상기 ADM 또는 XML 기반 디코더 디스크립션을 이용하여 상기 톨박스에 저장된 기능부들을 로드(load)하여 입력 데이터를 동영상 데이터로 디코딩하는 디코딩 솔루션을 포함한다. 본 발명에 의해, 디코더 디스크립션을 이용하여 다양한 형태로 복호화기의 재구성 및 제조립이 가능하다.

대표도 - 도3



특허청구의 범위

청구항 1

복수의 기능부들을 저장하는 톨박스;

압축된 디코더 디스크립션을 입력받아 CDDL 기반 디코더 디스크립션을 복원하는 복원부;

상기 CDDL 기반 디코더 디스크립션을 XML 기반 디코더 디스크립션으로 변환하는 디코더 디스크립션 해석부;

상기 XML 기반 디코더 디스크립션을 이용하여 추상적 복호 모델(ADM)을 생성하는 ADM 생성부; 및

상기 ADM 또는 XML 기반 디코더 디스크립션을 이용하여 상기 톨박스에 저장된 기능부들을 로드(load)하여 입력 데이터를 동영상 데이터로 디코딩하는 디코딩 솔루션

을 포함하는 복호화 장치.

청구항 2

제1항에 있어서,

상기 압축된 디코더 디스크립션은,

이진 코드 형식의 비트스트림인 것을 특징으로 하는 복호화 장치.

청구항 3

제1항에 있어서,

상기 CDDL 기반 디코더 디스크립션은,

ADM 또는/및 디코딩 솔루션을 구성하는데 필요한 FU 들의 연결 관계인 FU 네트워킹을 기술하는 FU 네트워킹 테이블 그룹; 및

신택스 파싱을 위한 신택스 파싱 테이블 그룹

을 포함하는 복호화 장치.

청구항 4

제 3 항에 있어서,

상기 FU 네트워킹 테이블 그룹에 포함되는 테이블은,

테이블 시작 코드(Table Start Code), 테이블 식별을 위한 테이블 번호(Table Code), 및 테이블 마침 코드(Table End Code) 필드를 포함하는 것을 특징으로 하는 복호화 장치.

청구항 5

제 3 항에 있어서,

상기 FU 네트워킹 테이블 그룹은,

가상 네트워크 테이블(Virtual Network Table, VNT);

기능부 인스턴스 테이블(Functional Unit Instance Table, FUIT);

네트워크 연결 테이블(Network Connection Table, NCT);

파라미터 테이블(Parameter Table, PT);

표현 테이블(Expression Table, ET) 및

유형 테이블(Type Table, TT)

을 포함하는 복호화 장치.

청구항 6

제 5 항에 있어서,

상기 가상 네트워크 테이블(Virtual Network Table, VNT)은 기본 템플릿(Template)에 해당하는 정보를 기술하는 것을 특징으로 하는 복호화 장치.

청구항 7

제 5 항에 있어서,

상기 기능부 인스턴스 테이블(Functional Unit Instance Table, FUIT)은 VNT 정보인 기본 템플릿(Template)에 기반하여 실제 필요로 하는 Network로 사용되는 객체들을 생성하기 위한 주요 정보가 저장되는 것을 특징으로 하는 복호화 장치.

청구항 8

제 5 항에 있어서,

상기 파라미터 테이블(Parameter Table, PT)은 Bitstream으로부터 생성되는 Syntax를 제외하고 기능부가 구성될 때 필요한 데이터를 생성하기 위해 사용되는 것을 특징으로 하는 복호화 장치.

청구항 9

제 5 항에 있어서,

상기 네트워크 연결 테이블(Network Connection Table, NCT)은, 각각의 기능 유닛들 간에 데이터를 전송할 수 있는 통로인 Port에 관한 정보를 포함하는 것을 특징으로 하는 복호화 장치.

청구항 10

제 5 항에 있어서,

상기 표현 테이블(Expression Table, ET)은, 구문(Syntax)의 표현을 위하여 참조되는 것을 특징으로 하는 복호화 장치.

청구항 11

복수의 기능부들을 저장하는 톨박스;

(a) 압축된 디코더 디스크립션을 입력받아 CDDL 기반 디코더 디스크립션을 복원하는 단계;

(b) 상기 CDDL 기반 디코더 디스크립션을 XML 기반 디코더 디스크립션으로 변환하는 단계;

(c) 상기 XML 기반 디코더 디스크립션을 이용하여 추상적 복호 모델(ADM)을 생성하는 단계; 및

(d) 상기 ADM 또는 XML 기반 디코더 디스크립션을 이용하여 톨박스에 저장된 기능부들을 로드(load)하여 입력 데이터를 동영상 데이터로 디코딩하는 단계

를 포함하는 복호화 방법.

청구항 12

제11항에 있어서,

상기 (a) 단계는,

이진 코드 형식의 비트스트림으로 구성된 압축된 디코더 디스크립션을 입력받는 것을 특징으로 하는 복호화 방법.

청구항 13

제11항에 있어서,

상기 CDDL 기반 디코더 디스크립션은,

ADM 또는/및 디코딩 솔루션을 구성하는데 필요한 FU 들의 연결 관계인 FU 네트워킹을 기술하는 FU 네트워킹 테이블 그룹; 및

신택스 파싱을 위한 신택스 파싱 테이블 그룹

을 포함하는 복호화 방법.

청구항 14

제 13 항에 있어서,

상기 FU 네트워킹 테이블 그룹에 포함되는 테이블은,

테이블 시작 코드(Table Start Code), 테이블 식별을 위한 테이블 번호(Table Code), 및 테이블 마침 코드(Table End Code) 필드를 포함하는 것을 특징으로 하는 복호화 방법.

청구항 15

제 13 항에 있어서,

상기 FU 네트워킹 테이블 그룹은,

가상 네트워크 테이블(Virtual Network Table, VNT);

기능부 인스턴스 테이블(Functional Unit Instance Table, FUIT);

네트워크 연결 테이블(Network Connection Table, NCT);

파라미터 테이블(Parameter Table, PT);

표현 테이블(Expression Table, ET) 및

유형 테이블(Type Table, TT)

을 포함하는 복호화 방법.

청구항 16

제 15 항에 있어서,

상기 가상 네트워크 테이블(Virtual Network Table, VNT)은 기본 템플릿(Template)에 해당하는 정보를 기술하는 것을 특징으로 하는 복호화 방법.

청구항 17

제 15 항에 있어서,

상기 기능부 인스턴스 테이블(Functional Unit Instance Table, FUIT)은 VNT 정보인 기본 템플릿(Template)에 기반하여 실제 필요로 하는 Network로 사용되는 객체들을 생성하기 위한 주요 정보가 저장되는 것을 특징으로 하는 복호화 방법.

청구항 18

제 15 항에 있어서,

상기 파라미터 테이블(Parameter Table, PT)은 Bitstream으로부터 생성되는 Syntax를 제외하고 기능부가 구성될 때 필요한 데이터를 생성하기 위해 사용되는 것을 특징으로 하는 복호화 방법.

청구항 19

제 15 항에 있어서,

상기 네트워크 연결 테이블(Network Connection Table, NCT)은, 각각의 기능 유닛들 간에 데이터를 전송할 수

있는 통로인 Port에 관한 정보를 포함하는 것을 특징으로 하는 복호화 방법.

청구항 20

제 15 항에 있어서,

상기 표현 테이블(Expression Table, ET)은, 구문(Syntax)의 표현을 위하여 참조되는 것을 특징으로 하는 복호화 방법.

청구항 21

복호화 방법을 수행하기 위해 복호화 장치에서 실행될 수 있는 명령어들의 프로그램이 유형적으로 구현되어 있으며, 상기 복호화 장치에 의해 판독될 수 있는 프로그램이 기록된 기록 매체에 있어서,

- (a) 압축된 디코더 디스크립션을 입력받아 CDDL 기반 디코더 디스크립션을 복원하는 단계;
 - (b) 상기 CDDL 기반 디코더 디스크립션을 XML 기반 디코더 디스크립션으로 변환하는 단계;
 - (c) 상기 XML 기반 디코더 디스크립션을 이용하여 추상적 복호 모델(ADM)을 생성하는 단계; 및
 - (d) 상기 ADM 또는 XML 기반 디코더 디스크립션을 이용하여 툴박스에 저장된 기능부들을 로드(load)하여 입력 데이터를 동영상 데이터로 디코딩하는 단계
- 를 실행하는 것을 특징으로 하는 프로그램이 기록된 기록매체.

명세서

발명의 상세한 설명

기술 분야

- <1> 본 발명은 동영상 데이터의 부호화/복호화에 관한 것으로서, 보다 상세하게는 기능부들(Functional Units)의 유기적 결합 또는 선택적 활성화를 통해 동영상 데이터를 부호화/복호화하는 방법 및 장치에 관한 것이다.

배경 기술

- <2> 일반적으로 동영상은 부호화기(encoder, 인코더)에 의해 비트스트림(Bit-stream) 형태로 변환된다. 이때, 비트스트림은 부호화기의 제약 조건을 만족하는 부호화 유형에 따라 저장된다.
- <3> MPEG은 비트스트림의 제약 조건으로서 구문(syntax, 이하 'syntax'라 칭함) 및 의미(semantics, 이하 'semantics'라 칭함)를 요구한다.
- <4> syntax는 데이터의 구조나 형식 및 길이를 나타내며, 데이터가 어떤 순서로 표현되는지를 나타낸다. 즉, syntax는 부호화(encoding)/복호화(decoding) 작업을 위한 문법을 맞추기 위한 것으로, 비트스트림에 포함된 각 요소들(elements)의 순서와 각 요소의 길이, 데이터 형식 등을 정의한다.
- <5> Semantics는 데이터를 구성하는 각 비트가 의미하는 뜻을 나타낸다. 즉, semantics는 비트스트림 내의 각 요소들의 의미가 무엇인지를 나타낸다.
- <6> 따라서, 부호화기의 부호화 조건 또는 적용된 표준(또는 코덱)에 따라 다양한 형태의 비트스트림이 생성될 수 있다. 일반적으로 각 표준(예를 들어 MPEG-1, MPEG-2, MPEG-4, MPEG-4 AVC 등)은 각각 상이한 비트스트림 syntax를 가진다.
- <7> 따라서, 각 표준이나 부호화 조건에 따라 부호화된 비트스트림은 각각 다른 형식(즉, syntax 및 semantics)을 가진다고 할 수 있으며, 해당 비트스트림의 복호화를 위해서는 부호화기에 대응되는 복호화기가 사용되어야 한다.
- <8> 상술한 바와 같이, 종래의 비트스트림 복호화기는 부호화기의 제약 조건을 만족하여야 하는 제한이 있었으며, 이러한 제한은 복수의 표준에 대응되는 통합 복호화기를 구현하기 어려운 원인이 된다.

발명의 내용

해결 하고자하는 과제

- <9> 따라서 본 발명은 상술한 문제점을 해결하기 위한 것으로, 각 표준(예를 들어, MPEG-1, MPEG-2, MPEG-4, MPEG-4 AVC 등)에 따른 다양한 형식(syntax, semantics)으로 부호화된 비트스트림을 동일한 정보 인식 방식으로 복호화(decoding)할 수 있는 비트스트림 디코딩 방법 및 장치를 제공하기 위한 것이다.
- <10> 본 발명은 각 표준에 따른 다양한 형식(syntax, semantics)으로 부호화된 비트스트림을 동일한 정보 인식 방식으로 복호화(decoding)할 수 있도록 디코더 디스크립션(Decoder Description)을 부가한 확장 비트스트림을 생성하거나 비트스트림과 디코더 디스크립션을 독립적으로 생성하는 비트스트림 인코딩 방법 및 장치를 제공하기 위한 것이다.
- <11> 본 발명은 다양한 부호화 방식으로 압축된 비트스트림이 동일한 정보 분석 방법에 의해 파싱(parsing)되고, 파싱된 데이터를 이용하여 복호화를 위한 각 기능부(FU, Functional Unit)들이 유기적으로 제어되는 비트스트림 디코딩 방법 및 장치를 제공하기 위한 것이다.
- <12> 본 발명은 코덱이 가진 계층 구조를 syntax 파싱 및 디코딩 과정에 응용함으로써 보다 효율적으로 디코더 디스크립션을 기술할 수 있도록 하는 비트스트림 인코딩 방법 및 장치를 제공하기 위한 것이다.
- <13> 본 발명은 디코더 디스크립션을 이용하여 각 코덱의 스케줄링(scheduling) 관리와 각 기능부들의 유기적 처리 구조(예를 들어, 병렬 결합 구조, 직렬 병합 구조, 독립 처리 구조, 개별적 처리 구조 등)를 제시할 수 있는 비트스트림 디코딩 방법 및 장치를 제공하기 위한 것이다.
- <14> 본 발명은 다양한 형태의 비트스트림을 복호화하기 위한 syntax 해석 방법을 공통적으로 적용할 수 있는 비트스트림 부호화/복호화 방법 및 장치를 제공하기 위한 것이다.
- <15> 본 발명은 다양한 형태의 비트스트림을 공통된 Syntax 해석 방법으로 파싱할 수 있도록 하기 위한 새로운 명령어들의 집합을 적용할 수 있는 비트스트림 부호화/복호화 방법 및 장치를 제공하기 위한 것이다.
- <16> 본 발명은 syntax 엘리먼트의 변경이나 추가, 삭제 시에도 복호화기가 용이하게 비트스트림을 복호화할 수 있는 비트스트림 디코딩 방법 및 장치를 제공하기 위한 것이다.
- <17> 본 발명은 해석된 syntax의 엘리먼트 정보(element information, 즉 syntax 파싱에 의한 결과물)를 비트스트림 복호화를 위해 이용되는 구성 요소들이 이용할 수 있도록 하는 비트스트림 디코딩 방법 및 장치를 제공하기 위한 것이다.
- <18> 본 발명은 이미 해석된 syntax의 엘리먼트 정보를 후속하는 비트스트림 syntax 엘리먼트의 해석을 위해 이용할 수 있도록 하는 비트스트림 디코딩 방법 및 장치를 제공하기 위한 것이다.
- <19> 본 발명은 여러 표준(코덱)에서 제안하는 다양한 디코딩 과정에 포함된 기능들을 각기 기능부(FU, Functional Unit)대로 분할하여 툴박스(Tool-Box)에 구비하는 비트스트림 디코딩 방법 및 장치를 제공하기 위한 것이다.
- <20> 본 발명은 다양한 형태로 부호화된 비트스트림을 복호화하기 위해 툴박스에서 필요한 기능부들만을 선별적으로 이용하는 비트스트림 디코딩 방법 및 장치를 제공하기 위한 것이다.
- <21> 본 발명은 툴박스에 저장된 기능부의 변경이나 추가, 삭제가 용이한 비트스트림 디코딩 방법 및 장치를 제공하기 위한 것이다.
- <22> 또한, 본 발명은 비트스트림 복호화를 위한 코덱 통합, 비트스트림이 동일한 정보 분석 방법에 의해 처리되도록 하기 위한 디코더 디스크립션의 생성 및 확장 비트스트림의 구현에 대한 국제 표준화를 달성하기 위한 것이며, 그 외의 다른 본 발명의 목적들은 이하에 서술되는 바람직한 실시예를 통하여 보다 명확해질 것이다.

과제 해결수단

- <23> 상술한 목적을 달성하기 위하여 본 발명의 일 측면에 따르면, 다양한 표준에 범용적으로 이용될 수 있는 복호화 장치 및/또는 통합 코덱 장치 및 그 방법이 제공된다.
- <24> 본 발명의 일실시예에 따른 복호화 장치는, 복수의 기능부들을 저장하는 툴박스; 압축된 디코더 디스크립션을 입력받아 CDDL 기반 디코더 디스크립션을 복원하는 복원부; 상기 CDDL 기반 디코더 디스크립션을 XML 기반 디코더 디스크립션으로 변환하는 디코더 디스크립션 해석부; 상기 XML 기반 디코더 디스크립션을 이용하여 추상적 복호 모델(ADM)을 생성하는 ADM 생성부; 및 상기 ADM 또는 XML 기반 디코더 디스크립션을 이용하여 상기 툴박스

에 저장된 기능부들을 로드(load)하여 입력 데이터를 동영상 데이터로 디코딩하는 디코딩 솔루션을 포함하는 것을 특징으로 한다.

<25> 본 발명의 일실시예에 따른 복호화 방법은, (a) 압축된 디코더 디스크립션을 입력받아 CDDL 기반 디코더 디스크립션을 복원하는 단계; (b) 상기 CDDL 기반 디코더 디스크립션을 XML 기반 디코더 디스크립션으로 변환하는 단계; (c) 상기 XML 기반 디코더 디스크립션을 이용하여 추상적 복호 모델(ADM)을 생성하는 단계; 및 (d) 상기 ADM 또는 XML 기반 디코더 디스크립션을 이용하여 툴박스에 저장된 기능부들을 로드(load)하여 입력 데이터를 동영상 데이터로 디코딩하는 단계를 포함하는 것을 특징으로 한다.

<26> 본 발명의 다른 실시예에 따르면, 복호화 방법을 수행하기 위해 복호화 장치에서 실행될 수 있는 명령어들의 프로그램이 유형적으로 구현되어 있으며, 상기 복호화 장치에 의해 판독될 수 있는 프로그램이 기록된 기록 매체에 있어서, (a) 압축된 디코더 디스크립션을 입력받아 CDDL 기반 디코더 디스크립션을 복원하는 단계; (b) 상기 CDDL 기반 디코더 디스크립션을 XML 기반 디코더 디스크립션으로 변환하는 단계; (c) 상기 XML 기반 디코더 디스크립션을 이용하여 추상적 복호 모델(ADM)을 생성하는 단계; 및 (d) 상기 ADM 또는 XML 기반 디코더 디스크립션을 이용하여 툴박스에 저장된 기능부들을 로드(load)하여 입력 데이터를 동영상 데이터로 디코딩하는 단계를 실행하는 것을 특징으로 하는 프로그램을 기록한 기록매체가 제공된다.

효 과

<27> 상술한 바와 같이 본 발명에 따른 통합 코덱 장치 및 방법은 각 표준(예를 들어, MPEG-1, MPEG-2, MPEG-4, MPEG-4 AVC 등)에 따른 다양한 형식(syntax, semantics)으로 부호화된 비트스트림을 동일한 정보 인식 방식으로 복호화(decoding)할 수 있는 효과가 있다.

<28> 또한, 본 발명은 각 표준에 따른 다양한 형식(syntax, semantics)으로 부호화된 비트스트림을 동일한 정보 인식 방식으로 복호화(decoding)하도록 디코더 디스크립션(decoder description)을 부가한 확장 비트스트림을 생성할 수 있는 효과도 있다.

<29> 또한, 본 발명은 코덱이 가진 계층 구조를 syntax 파싱 및 디코딩 과정에 응용함으로써 보다 효율적으로 디코더 디스크립션을 기술할 수 있도록 하는 효과도 있다.

<30> 또한, 본 발명은 디코더 디스크립션을 이용하여 각 코덱의 스케줄링(scheduling) 관리와 각 기능부들의 유기적 처리 구조(예를 들어, 병렬 결합 구조, 직렬 병합 구조, 독립 처리 구조, 개별적 처리 구조 등)를 제시할 수 있는 효과도 있다.

<31> 또한, 본 발명은 기술된 디코더 디스크립션만으로 다양한 시스템 설계 및 구축이 가능한 효과도 있다.

<32> 또한, 본 발명은 다양한 부호화 방식으로 압축된 비트스트림을 동일한 정보 분석 방법에 의해 파싱(parsing)하고, 파싱된 데이터를 이용하여 복호화를 위한 각 기능부(FU, Functional Unit)들을 유기적으로 제어할 수 있는 효과도 있다.

<33> 또한, 본 발명은 다양한 형태의 비트스트림을 복호화하기 위한 syntax 해석 방법을 공통적으로 적용할 수 있는 효과도 있다.

<34> 또한, 본 발명은 다양한 형태의 비트스트림을 공통된 Syntax 해석 방법으로 파싱할 수 있도록 하기 위한 새로운 명령어들의 집합을 적용할 수 있는 효과도 있다.

<35> 또한, 본 발명은 syntax 엘리먼트의 변경이나 추가시에도 복호화기가 용이하게 비트스트림을 복호화할 수 있는 효과도 있다.

<36> 또한, 본 발명은 해석된 syntax의 엘리먼트 정보(element information, 즉 syntax 파싱에 의한 결과물)를 비트스트림 복호화를 위해 이용되는 구성 요소들이 공유할 수 있도록 하는 효과도 있다.

<37> 또한, 본 발명은 해석된 syntax의 엘리먼트 정보를 후속하는 비트스트림 syntax 엘리먼트의 해석을 위해 이용할 수 있도록 하는 효과도 있다.

<38> 또한, 본 발명은 MPEG-1, MPEG-2, MPEG-4, MPEG-4 AVC 외의 블록 단위의 처리를 하는 동영상, 정지영상 코덱의 통합시에 사용할 수 있는 효과도 있다.

<39> 또한, 본 발명은 여러 표준(코덱)에서 제안하는 다양한 디코딩 방법을 구성하는 기능들을 각기 기능부(FU,

Functional Unit)대로 분할하여 툴박스에 저장할 수 있는 효과도 있다.

<40> 또한, 본 발명은 다양한 형태로 부호화 된 비트스트림을 복호화하기 위해 툴박스에서 필요한 기능부들만을 선별하여 디코딩할 수 있는 효과도 있다.

<41> 또한, 본 발명은 툴박스에 저장된 기능부의 변경이나 추가, 삭제가 용이한 효과도 있다.

발명의 실시를 위한 구체적인 내용

<42> 본 발명은 다양한 변경을 가할 수 있고 여러 가지 실시예를 가질 수 있는 바, 특정 실시예들을 도면에 예시하고 상세한 설명에 상세하게 설명하고자 한다. 그러나, 이는 본 발명을 특정한 실시 형태에 대해 한정하려는 것이 아니며, 본 발명의 사상 및 기술 범위에 포함되는 모든 변경, 균등물 내지 대체물을 포함하는 것으로 이해되어야 한다.

<43> 제1, 제2 등과 같이 서수를 포함하는 용어는 다양한 구성요소들을 설명하는데 사용될 수 있지만, 상기 구성요소들은 상기 용어들에 의해 한정되지는 않는다. 상기 용어들은 하나의 구성요소를 다른 구성요소로부터 구별하는 목적으로만 사용된다. 예를 들어, 본 발명의 권리 범위를 벗어나지 않으면서 제1 구성요소는 제2 구성요소로 명명될 수 있고, 유사하게 제2 구성요소도 제1 구성요소로 명명될 수 있다. "및/또는"이라는 용어는 복수의 관련된 기재된 항목들의 조합 또는 복수의 관련된 기재된 항목들 중의 어느 항목을 포함한다.

<44> 어떤 구성요소가 다른 구성요소에 "연결되어" 있다거나 "접속되어" 있다고 언급된 때에는, 그 다른 구성요소에 직접적으로 연결되어 있거나 또는 접속되어 있을 수도 있지만, 중간에 다른 구성요소가 존재할 수도 있다고 이해되어야 할 것이다. 반면에, 어떤 구성요소가 다른 구성요소에 "직접 연결되어" 있다거나 "직접 접속되어" 있다고 언급된 때에는, 중간에 다른 구성요소가 존재하지 않는 것으로 이해되어야 할 것이다.

<45> 본 출원에서 사용한 용어는 단지 특정한 실시예를 설명하기 위해 사용된 것으로, 본 발명을 한정하려는 의도가 아니다. 단수의 표현은 문맥상 명백하게 다르게 뜻하지 않는 한, 복수의 표현을 포함한다. 본 출원에서, "포함하다" 또는 "가지다" 등의 용어는 명세서상에 기재된 특징, 숫자, 단계, 동작, 구성요소, 부품 또는 이들을 조합한 것이 존재함을 지정하려는 것이지, 하나 또는 그 이상의 다른 특징들이나 숫자, 단계, 동작, 구성요소, 부품 또는 이들을 조합한 것들의 존재 또는 부가 가능성을 미리 배제하지 않는 것으로 이해되어야 한다.

<46> 다르게 정의되지 않는 한, 기술적이거나 과학적인 용어를 포함해서 여기서 사용되는 모든 용어들은 본 발명이 속하는 기술 분야에서 통상의 지식을 가진 자에 의해 일반적으로 이해되는 것과 동일한 의미를 가지고 있다. 일반적으로 사용되는 사전에 정의되어 있는 것과 같은 용어들은 관련 기술의 문맥 상 가지는 의미와 일치하는 의미를 가지는 것으로 해석되어야 하며, 본 출원에서 명백하게 정의하지 않는 한, 이상적이거나 과도하게 형식적인 의미로 해석되지 않는다.

<47> 이하, 첨부한 도면들을 참조하여 본 발명에 따른 통합 코덱 방법 및 장치의 바람직한 실시예를 상세히 설명하기로 하며, 첨부 도면을 참조하여 설명함에 있어 도면 부호에 상관없이 동일하거나 대응하는 구성 요소는 동일한 참조번호를 부여하고 이에 대한 중복되는 설명은 생략하기로 한다.

<48> 도 1은 일반적인 복호화기의 구성을 개략적으로 나타낸 도면이고, 도 2는 일반적인 부호화기의 구성을 개략적으로 나타낸 도면이다.

<49> 도 1에 도시된 바와 같이, 일반적으로 MPEG-4 복호화기(100)는 가변장 디코딩부(Variable Length Decoding, 110), 역 스캔부(Inverse Scan, 115), 역 DC/AC 예측부(Inverse DC/AC Prediction, 120), 역 양자화부(Inverse Quantization, 125), 역 DCT부(Inverse Discrete Cosine Transform, 역 이산 여현 변환부, 130), 동영상 복원부(VOP Reconstruction, 135)를 포함한다. 복호화기(100)의 구성은 적용되는 표준에 따라 상이할 수 있음은 자명하며, 또한 일부 구성요소는 타 구성요소로 대체될 수도 있을 것이다.

<50> 전달된 비트스트림(105)이 syntax 파싱(parsing)되어 헤더 정보 및 인코딩된 비디오 데이터(encoded video data)가 추출되면, 가변장 디코딩부(110)는 미리 저장된 허프만 테이블(Huffman Table)을 이용하여 양자화된 DCT 계수를 만들고, 역 스캔부(115)는 역 스캔을 수행하여 동영상(140)과 동일한 순서의 데이터를 생성한다. 즉, 역 스캔부(115)는 인코딩시 여러 가지 방법으로 스캔된 순서의 역으로, 값을 출력한다. 인코딩 시 양자화(Quantization)를 수행한 후, 주파수 대역 값의 분포에 따라 스캔 방향이 정의될 수 있다. 일반적으로는 지그재그(zig-zag) 스캔 방식이 사용되나, 스캔 방식은 코덱별로 다양할 수 있다.

<51> Syntax 파싱은 가변장 디코딩부(110)에서 통합적으로 수행되거나, 가변장 디코딩부(110)에 선행하여 비트스트림

(105)을 처리하는 임의의 구성 요소에서 수행될 수 있다. 이 경우, Syntax 파싱은 부호화기와 복호화기간에 적용되는 표준이 동일하므로 해당 표준에 상응하도록 미리 지정된 기준에 의해서만 처리된다.

- <52> 역 DC/AC 예측부(120)는 주파수 대역에서 DCT 계수의 크기를 이용하여 예측을 위한 참조 블록의 방향성을 결정한다.
- <53> 역 양자화부(125)는 역 스캔된 데이터를 역 양자화한다. 즉, 인코딩시 지정된 양자화값(QP, Quantization Parameter)을 이용하여 DC와 AC 계수를 환원한다.
- <54> 역 DCT부(130)는 역 이산 여현 변환(Inverse Discrete Cosine Transform)을 수행함으로써 실제의 동영상 픽셀 값을 구하여 VOP(Video Object Plane)를 생성한다.
- <55> 동영상 복원부(135)는 역 DCT부(130)에 의해 생성된 VOP를 이용하여 동영상 신호를 복원하여 출력한다.
- <56> 도 2에 도시된 바와 같이, 일반적으로 MPEG-4 부호화기(200)는 DCT부(210), 양자화부(215), DC/AC 예측부(220), 스캔부(230), 가변장 인코딩부(235)를 포함한다.
- <57> 부호화기(200)에 포함된 각 구성요소는 각각 대응되는 복호화기(100)의 구성 요소의 역 기능을 수행하며, 이는 당업자에게 자명하다. 간단히 설명하면, 부호화기(200)는 동영상 신호(즉, 디지털 영상 픽셀 값)를 이산 여현 변환(Discrete Cosine Transform), 양자화(Quantization) 등을 통해 주파수 값으로 변환하여 부호화를 수행한 후, 이를 정보의 빈도 수에 따라 비트 길이를 차별화하는 가변장 인코딩을 수행하여 압축된 비트스트림 상태로 출력한다.
- <58> 도 3은 본 발명의 일 실시예에 따른 부호화기의 블록 구성도이다.
- <59> 본 발명에 따른 부호화기는 앞서 도 2를 참조하여 설명한 종래의 부호화기(200)에 비해 확장 비트스트림 생성 및 출력부(2410)를 더 포함한다. 확장 비트스트림 생성 및 출력부(2410)는 진단까지의 처리에 의해 생성된 종래 비트스트림(316) 생성 과정에서의 제어 정보(예를 들어, 사용한 기능부들의 목록 및 연결 관계, 해당 기능부들의 입력 데이터, 선택스 정보, 선택스 연결 정보 등)를 이용하여 디코더 디스크립션을 생성한다. 또한, 생성된 디코더 디스크립션(313) 및 종래 비트스트림(316)을 이용하여 확장 비트스트림(305)을 생성하여 복호화기(300)로 전송한다.
- <60> 부호화기는 복수의 부호화 기능부들을 포함하는 툴 박스를 구비하도록 하고, 툴 박스 내에 포함된 기능부들의 순차적 조합 또는 유기적 조합을 통해 하나 이상의 부호화 표준에 따른 비트스트림을 생성할 수 있다.
- <61> 또한, 본 명세서에서 가변장 인코딩부(235)는 부호화기 내에서 종래 비트스트림(316)을 생성하기 위하여 최종적으로 부호화를 수행하는 임의의 구성 요소(예를 들어, 부호화부)를 지칭한 것일 뿐 이에 제한되는 것은 아니며, 또한 이로 인해 본 발명의 권리범위가 제한되지는 않는다.
- <62> 도 3은 디코더 디스크립션 정보 및 종래 비트스트림을 이용하여 생성한 확장 비트스트림이 복호화기로 제공되는 경우를 가정한 도면이다.
- <63> 하지만, 디코더 디스크립션은 별도의 데이터 또는 비트스트림 등의 형태로 복호화기(300)로 전달될 수도 있다. 이 경우는 가변장 인코딩부(235) 후단에 인코딩된 디코더 디스크립션 생성 및 출력부(도시되지 않음)가 위치하여, 종래의 인코딩부와 독립적으로 생성한 인코딩된 디코더 디스크립션을 복호화기(300)로 제공할 수도 있음을 자명하다.
- <64> 도 4는 본 발명에 따른 복호화기의 제1 실시예 블록 구성도이고, 도 5는 도4의 디코딩 유닛에서 비트스트림 처리 과정의 일실시예를 구체적으로 나타낸 도면이다.
- <65> 도 4에 예시된 디코더 디스크립션 및 영상 비트스트림은 예를 들어 부호화기에 의해 생성되어 제공되는 정보일 수 있다.
- <66> 도 4을 참조하면, 복호화기(300)는 디코딩 유닛(305) 및 분리부(310)을 포함한다. 디코딩 유닛(305)은 BSDL 파서(320), 디코더 형성부(330), 툴박스(335) 및 디코딩 솔루션(340)을 포함한다.
- <67> BSDL 파서(320)는 분리부(310)로부터 입력된 BSDL 스키마(schema)를 이용하여 외부로부터 입력된 영상 비트스트림의 구문정보를 해석한다. BSDL 파서(320)로 입력되는 영상 비트스트림은 임의의 부호화 방식(예를 들어, MPEG-4, AVS 등)에 의해 부호화된 데이터이다. 본 명세서를 통해 BSDL 파서(320)가 자체적으로 BSDL Schema를 해석할 수 있거나, 또는 외부 알고리즘에 의해 구성될 수 있음을 당업자는 쉽게 이해할 수 있을 것이다.

- <68> BSDL 파서(320)는 XML 문법으로 기술된 BSDL 스키마를 읽어들이어 BSDL 파서(320)의 구조를 재정의하기 위한 내부 처리부인 BSDL 해석 처리부를 포함한다.
- <69> BSDL 스키마를 이용해 재정의하는 규칙은 제작자가 적용하는 방법에 따라 다양할 수 있으므로, 그 기본적인 목적만을 제시하면 다음과 같다. 첫째, BSDL 스키마 상에 기록되어 있는 비트스트림의 길이 및 의미에 대한 정보를 인식할 수 있도록 하기 위한 것이다. 둘째, BSDL 스키마 상에 정의된 반복 구조 및 조건적 실행 구조를 읽어 들여 같은 반복 또는 조건문에 의해 실제 동작하는 프로그램적인 루틴을 구현하기 위한 것이다. 따라서, 재정의되기 이전의 BSDL 파서(320)는 위와 같은 목적을 달성하기 위한 기능들만이 구현된 상태라고 정의할 수도 있을 것이며, 상술한 파싱 기능을 활용해 실제로 구동하는 BSDL 파서(320)를 구현하는 과정을 재정의 하는 과정이라 할 수 있다.
- <70> BSDL 파서(320)는 BSDL 해석 처리부의 제어에 의해 유동적인 데이터 흐름을 구성할 수 있는 프로그램으로 구현되며, 예를 들어, CAL(Caltrop Actor Language), C, C++, Java 등 프로그램 언어를 이용하여 구현될 수 있다.
- <71> BSDL 내부 처리부(2525) 및 BSDL 파서(320)는 디코더 설계자의 설계 기준에 따라 제한없이 구현될 수 있다. 물론, BSDL 레퍼런스 소프트웨어와 같이 기준에 제시되어 있는 BSDL 운용 프로그램을 응용할 수도 있을 것이다. BSDL 레퍼런스 소프트웨어는 MPEG 표준화 단체에 의해 표준화된 BSDL의 원활한 운용을 위하여 제작된 공식 소프트웨어로서, BSDL 스키마를 입력받는 BSDL 파서(320) 역시 이러한 소프트웨어 자원을 이용하여 보다 용이하게 구현될 수 있음은 자명하다.
- <72> 본 명세서에서 언급되는 바와 같이, BSDL 파서(320)의 기본적인 구조는 디코더 설계자가 선택한 다양한 방법에 의해 설계될 수 있다. 즉, 디코더 설계자는 BSDL 파서(320)의 지정된 기능을 수행하도록 하기 위한 상세한 알고리즘의 적용 및 설계를 자율적으로 선택할 수 있다. 다만, BSDL 파서(320)는 BSDL 스키마를 읽어들이는 결과에 의해 재정의될 수 있으며, 재정의된 결과물이 디코딩 유닛(305)의 다른 구성 요소들과 협업(예를 들어, 통신 등)될 수 있어야 한다.
- <73> BSDL 파서(320)가 입력받는 BSDL 스키마에는 비트스트림에 포함된 구문 정보들에 대한 상세한 내역이 기술되며, 이 내역에는 예를 들어 구문 정보의 길이, 구문 정보의 의미, 구문 정보의 출현 조건 및 반복 출현 횟수 등이 포함될 수 있다. 여기서, 정보의 길이는 비트스트림 상에서 특정 정보가 차지하는 비트 길이를 의미하며, 구문 정보의 의미는 해당 정보가 어떤 의미를 가지는 정보인지를 나타낸다. 예를 들어, 임의의 기능부에서 A라는 정보를 요청하고 있을 경우 어느 것이 정보 A인지를 구별하는 데 필요할 수 있기 때문이다. 또한, 출현 조건이나 반복 출현 횟수의 경우, 하나의 BSDL 스키마를 이용해 동일한 규격의 동영상 비트스트림을 처리할 경우에도 비트스트림의 속성에 따라 일부 구문 정보의 출현 여부나 반복 횟수 등이 달라질 수 있으므로 이러한 경우를 정의하기 위해 BSDL 스키마에 첨부될 수 있는 정보이다. 예를 들어, 출현 조건은 인트라 프레임을 처리할 때는 모션 벡터 정보를 읽어들이지 않도록 하는 데에 필요할 수 있고, 반복 출현 횟수는 해당 매크로블록이 동일한 구조의 블록을 6개 지닌다고 할 경우 해당 블록을 반복시키는 데 사용될 수 있다.
- <74> 도 5에 예시된 바와 같이, BSDL 해석 처리부는 상세한 내역에 관하여 해독한 결과 정보를 BSDL 파서(320)에 전달하여 BSDL 파서(320)가 BSDL 스키마에 지정된 순서에 따라 비트스트림에 포함된 정보를 읽어들이도록 지원한다.
- <75> BSDL 파서(320)는 BSDL 해석 처리부로부터 제공된 결과 정보를 참조하여 입력된 비트스트림의 내용을 의미 있는 데이터로 바꾸어 디코더 형성부(330) 및/또는 디코딩 솔루션(340)에 제공한다. 또한, BSDL 파서(320)가 디코더 형성부 또는/및 디코딩 솔루션(340)으로 제공하는 의미있는 데이터들로는 예를 들어, 미리 지정된 매크로블록 사이즈의 인코딩된 영상 데이터, 인트라 코딩된 매크로블록들에 대한 AC 예측 플래그(ACpred_flag), MCBPC(MB type & coded block pattern for chrominance), CBPY (coded block pattern for luminance) 등이 포함될 수 있다. 이러한 데이터 제공 과정은 디코더 형성부(330)나 디코딩 솔루션(340)의 구동 여부와 무관하게 진행될 수도 있다.
- <76> 본 실시예는 디코더(복호화기)가 디코더 디스크립션을 이용하여 비트스트림을 디코딩하되, 디코더 디스크립션이 BSDL 언어 체계 및 이와 연동 가능한 XML 기반 서식을 사용하는 구조로 구현되도록 하기 위한 것이다. 본 실시예를 통해 디코더 디스크립션이 BSDL, CALML 등의 XML 서식을 가질 수 있고, BSDL 스키마는 Syntax Parsing 과정에, CALML은 기능부간의 연결 제어를 위해 사용되도록 그 역할이 구분될 수 있음을 당업자는 쉽게 이해할 수 있을 것이다.
- <77> BSDL 언어는 비트스트림의 구조와 구성 방식에 대한 정보를 포함한 XML 문서 또는 XML 스키마 형태로 기술된다.

이 언어는 각각이 하나 이상의 영상 비트스트림 구조를 표현할 수 있도록 제작된다. BSDL 언어를 사용함으로써, 디코더는 종래의 MPEG 표준에서 검증되어 사용되고 있는 비트스트림 기술 방식을 그대로 적용할지라도, 타 기술들과 높은 호환성을 획득할 수 있게 된다. BSDL에 관련된 언어 서식 및 문법은 MPEG-B Part 5에 기술되어 있으므로 이에 대한 상세한 설명은 생략한다.

BSDL과 XML을 이용한 BSDL 스키마와 연결 제어 정보의 구성예를 나타내면 아래와 같다. 물론, BSDL 스키마와 연결 제어 정보의 구성 형식이 이에 제한되지 않음은 자명하다.

BSDL 스키마

```
<xsd:element name="VideoObject">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="VOSTartCode"
        type="m4v:StartCodeType"/>
      <xsd:element name="VOL">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="header" type="VOLHeaderType"
              bs2:ifNext="&volSC;" rvc:port="0"/>
            <xsd:element name="VOP"
              type="VideoObjectPlaneType"
              maxOccurs="unbounded"
              bs2:ifNext="&vopSC;" rvc:port="1"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

연결 제어 정보
<Network name="Decoder">
  <Package>
    <QID>
      <ID id="MPEG4 Simple Profile" />
    </QID>
  </Package>
  <Port kind="Input" name="BITSTREAM" />
  <Port kind="Output" name="YUV" />
  <Instance id="1">
```

<110> <Class name="Parser">

<111> <QID>

<112> <ID id="c" />

<113> </QID>

<114> </Class>

<115> <Note kind="label" name="Stream Parser" />

<116> </Instance>

<117> <Instance id="2">

<118> <Class name="VS">

<119> <QID>

<120> <ID id="c" />

<121> </QID>

<122> <Note kind="label" name="Video Session" />

<123> </Class>

<124> </Instance>

<125> <Connection src="" src-port="BITSTREAM" dst="1" dst-port="BITSTREAM" />

<126> <Connection src="1" src-port="CSCI" dst="2" dst-port="CSCI" />

<127> <Connection src="1" src-port="DATA" dst="2" dst-port="DATA" />

<128> <Connection src="2" src-port="YUV" dst="" dst-port="YUV" />

<129> </Network>

<130> 본 실시예에서는 디코더 디스크립션(2560), 인코딩된 비디오 데이터(2580) 등의 정보가 외부로부터 입력되는 경우를 가정하여 설명하나, 그 중 하나 이상의 정보가 디코딩 유닛(305)의 임의의 구성 요소 내에 이미 내장되어 구현될 수도 있음은 자명하다.

<131> 다시 도 4을 참조하면, 디코더 형성부(330)는 분리부(310)로부터 입력받은 연결 제어 정보 또는/및 BSDL 파서(320)로부터 입력받은 비트스트림 데이터의 일부(예를 들어, 미리 지정된 매크로블록 사이즈의 인코딩된 영상 데이터, 인트라 코딩된 매크로블록들에 대한 AC 예측 플래그(ACpred_flag), MCBPC(MB type & coded block pattern for chrominance), CBPY (coded block pattern for luminance) 등 중 하나 이상)을 이용하여 디코딩 솔루션(340)이 구현되도록 제어한다.

<132> 즉, 디코더 형성부(330)는 연결 제어 정보 등을 이용하여 툴박스(335)에 구비된 기능부들 중 일부 또는 전체가 유기적인 연결 관계로서 디코딩 솔루션(340) 내에 로드되어 정렬되도록 제어한다. 여기서, 연결 제어 정보는 CALML(CAL Markup Language)로 작성될 수 있다. CALML이란 MPEG 표준화 단체에서 현재 논의중인 CAL언어(Caltrop Markup Language) 방식의 디코더 구성을 기술할 수 있는 XML 포맷이다. CAL언어는 프로그램 객체인 Actor와 각 Actor간의 연결 관계로 구성되는데, 이러한 CAL언어의 구조를 XML 서식에 의해 표현한 것이다. 이에 대한 표현 예는 앞서 BSDL 스키마와 연결 제어 정보의 표현 예로서 이미 제시하였다.

<133> 구체적으로, 디코더 형성부(330)는 여러 기능부의 집합으로 구성된 툴박스(335)에 접근할 권한을 가지며, 툴박스(335)에 구비된 기능부들간에 입, 출력 연결을 설정하여 그 결과물에 해당하는 디코딩 솔루션(340)을 구성한다. 이 때 기능부들간 입/출력 연결 구조 및 실행 순서는 연결 제어 정보를 참고하여 설정된다. 또한, 입력된 비트스트림의 종류를 구별하기 위한 일부 정보를 BSDL 파서(320)로부터는 전달받아, 기능부 연결 과정에서 참조할 수 있다. 기능부들간의 연결 구조가 모두 확정되면, 그 연결 구조는 외부로부터의 지속적인 데이터 입력이 전제될 경우 해당 디코더 디스크립션 제작자가 의도한 모든 종류의 영상 비트스트림을 해석, 디코딩 할 수 있는 독립적인 디코더로 간주할 수 있다. 이 때 이 완결된 기능부 연결 구조를 디코딩 솔루션(340)이라 명명할 수 있

다.

- <134> 툴박스(335)는 미리 지정된 프로세스를 수행하도록 각각 구현된 복수의 기능부들을 포함한다. 각각의 기능부들은 각각 프로그램 코드들의 조합으로 구현될 수도 있다.
- <135> 툴박스(335)에 포함되는 각 기능부들은 적용되는 용도별 집합으로 구분된 복수의 세부 툴박스로 세분화될 수도 있다. 예를 들어, MPEG-용 기능부들이 포함되는 제1 툴박스, MPEG-용 기능부들 이외의 기능부들이 포함되는 제2 툴박스 등으로 세분화될 수 있다. 또는 MPEG-2용 기능부들의 집합인 제1 툴박스, MPEG-4용 기능부들의 집합인 제2 툴박스, 중국의 디지털 TV 압축 표준인 AVS용 기능부들의 집합인 제3 툴박스 등으로 세분화될 수 있다.
- <136> 물론, 툴박스(335) 자체가 복수로 구현되어 디코더 형성부(330) 및 디코딩 솔루션(340)과 독립된 연결 관계를 가질 수도 있다. 이 경우, 도시되지는 않았으나 상술한 제1 툴박스, 제2 툴박스 등이 독립된 형식의 툴박스로 구현될 수 있을 것이다.
- <137> 다만, 이하에서는 설명의 편의를 위해 하나의 툴박스(335) 내에 복수의 세부 툴박스들이 포함되거나 모든 기능부들이 집합체 구성없이 산재되어 포함되는 경우를 중심으로 설명하기로 한다.
- <138> 툴박스(335)는 각각의 기능(즉, 미리 설정된 프로세스)을 수행하도록 구현된 기능부(FU, Functional Unit)들이 포함되는 영역으로, 각 기능부들은 디코더 형성부(330)의 연결 제어에 의해 디코딩 솔루션(340)에 로드(load)되어 순차적인 연결 동작 관계를 형성함으로써 영상 비트스트림(380)에 포함된 인코딩된 영상 데이터를 디코딩된 영상 데이터로 출력한다.
- <139> 툴박스(335) 내에는 예를 들어, DF(De-blocking Filter) 기능부, VR(VOP Reconstructor) 기능부, FFR(Frame Field Reordering) 기능부, IPR(Intra prediction and Picture Reconstruction) 기능부, IT(Inverse Transform) 기능부, IQ(Inverse Quantization) 기능부, IAP(Inverse AC Prediction) 기능부, IS(Inverse Scan) 기능부, DCR(DC Reconstruction) 기능부 등의 기능부가 포함될 수 있다.
- <140> IT4x4 기능부, IQ4x4 기능부) 및 DCR4x4 기능부는 처리하는 블록 사이즈가 4x4인 것을 특징으로 한다. 이는 MPEG-1/2/4의 경우에는 Transform, Quantization, Prediction 시에 8x8 블록 사이즈로 데이터를 처리함에 비해, MPEG-4 AVC는 4x4 블록 사이즈로 데이터를 처리하는 경우가 존재하기 때문이다.
- <141> 툴박스(335)에는 데이터 디코딩 기능을 수행하기 위한 기능부라면 적용되는 표준에 관계없이 모두 포함될 수 있을 뿐 아니라 기술 발전과정에서 필요한 기능부는 추가될 수 있고, 기존 기능부의 수정도 가능하며, 불필요한 기능부는 제거될 수 있음은 자명하다. 예를 들어, 복호화 처리를 위해 4x4 블록 사이즈로 데이터를 처리하는 IS4x4 기능부 등이 추가로 필요한 경우 해당 기능부들이 툴박스(335)에 추가될 수 있다. 또한, MPEG-4 AVC에서 인트라 예측(Intra Prediction) 수행을 위한 SPR(Special Prediction) 기능부 등이 더 추가될 수도 있을 것이다.
- <142> 툴박스(335) 내에 구비된 각 기능부는 각 표준에 독립적으로 존재하지 않고, 표준에 관계없이 동일한 처리가 가능한 기능부의 경우에는 하나의 기능부로 통합되어 구비될 수도 있음은 자명하다. 각 기능부의 기능은 당업자에게 자명한 사항이므로 간략히 설명하기로 한다.
- <143> DF 기능부는 MPEG-4 AVC의 디-블록킹 필터(de-blocking filter)이고, VR 기능부는 복원된 픽셀값을 저장하는 기능부이다.
- <144> FFR 기능부는 interlaced 모드를 위한 기능부이고, IPR 기능부는 MPEG-4 AVC의 인트라 예측(Intra prediction)을 한 후 복원된 픽셀값을 저장하는 기능부이다. 상술한 바와 같이, MPEG-4 AVC의 인트라 예측은 SPR 기능부에 의해 수행될 수 있을 것이다.
- <145> IT 기능부는 DC값 및 AC값들의 역 변환(inverse transform)을 수행하는 기능부이고, IQ 기능부는 AC 값들을 역 양자화(inverse quantization)하는 기능부이다.
- <146> IAP 기능부는 AC값들을 역 예측(inverse AC prediction)하는 기능부이고, IS 기능부는 AC값들을 역 스캔(inverse scan)하는 기능부이다. DCR 기능부는 DC값들의 역 예측 및 역 양자화를 수행하는 기능부이다.
- <147> 디코딩 솔루션(340)은 디코더 형성부(330)에 의해 생성된 결과물로, BSDL 파서(320)에서 구분 정보 단위로 분리된 비트스트림 데이터(또는 미리 지정된 매크로블록 크기의 인코딩된 비디오 데이터)를 입력 받는다.
- <148> 도 5에 예시된 바와 같이, 입력된 비트스트림 데이터는 데이터를 입/출력하기 위한 유, 무형의 데이터 인터페이

스를 통해 입력될 수 있다. 데이터 인터페이스는 소프트웨어의 경우 특정 메모리 버퍼이거나, 자료의 흐름을 정의하는 가상 포트(Port)이거나, 프로그램 상의 파라미터일 수 있고, 하드웨어의 경우에는 회로 상의 연결선일 수 있으며, 이외에도 다양하게 구현될 수 있다.

- <149> 데이터의 입력은 해당 인터페이스를 통해 지속적으로, 그리고 특정 기능부의 프로세스 수행과 무관하게 지속적으로 입력(예를 들어, 병렬 처리가 가능하도록 입력)될 수 있다. 디코딩 솔루션(340)은 입력되는 데이터를 처리하여 디코딩된 영상 데이터로 출력한다. 도 5에 도시된 바와 같이, 데이터는 데이터 인터페이스로부터 시작해 각 기능부로 전달될 수 있으며, 기능부는 해당 데이터를 가공하여 후속하는 기능부로 전달할 수 있다. 이러한 데이터의 흐름은 모두 디코더 형성부(330)에 의해 사전 정의된 바에 의하여 처리된다.
- <150> 디코딩 솔루션(340)내에는 BSDL 파서(320)로부터 제공받은 데이터(예를 들어, 비트스트림의 syntax 파싱에 의해 추출된 정보 등), 각 기능부의 처리 결과 데이터를 저장하기 위한 저장부가 포함될 수 있다. 디코더 형성부(330)의 제어에 의해 로드된 각 기능부는 BSDL 파서(320)로부터 제공된 데이터 및 선행하여 동작된 기능부의 결과 데이터 중 하나 이상을 이용하여 지정된 프로세스를 수행할 수 있다. 이 경우, 후속하여 프로세스를 수행할 기능부는 선행하는 기능부의 동작이 완료되었음을 인식하여야 한다. 이를 위해, 디코더 형성부(330)는 각 기능부의 동작 완료 여부를 지속적으로 모니터링하여 후속하는 기능부의 동작 개시 여부를 제어할 수 있다. 또한, 저장부 내에 각 기능부별로 독립된 영역이 구비되도록 하고, 선행하는 기능부의 처리 결과 데이터를 디코더 형성부(330)의 제어에 의해 후속하는 기능부를 위한 저장 영역에 저장되도록 하면, 후속하는 기능부는 자신의 저장 영역에 프로세스 수행을 위해 필요한 데이터가 저장되는 즉시 프로세스를 수행할 수도 있을 것이다. 이외에도, 기능부간의 처리 개시 시점을 제어하기 위한 다양한 방법이 추가적으로 고려될 수 있음은 자명하다.
- <151> 물론, 해당 저장부는 디코더 형성부(330) 내에 구비될 수도 있으며, 디코더 형성부(330)는 현재 프로세스를 수행할 기능부로 BSDL 파서(320)로부터 제공받은 데이터(예를 들어, 비트스트림의 syntax 파싱에 의해 추출된 정보 등), 각 기능부의 처리 결과 데이터를 해당 기능부로 제공할 수도 있다.
- <152> 이하, 도 5를 참조하여 디코딩 유닛(305)의 동작 과정을 간략히 설명하면 다음과 같다.
- <153> 외부로부터 입력 영상 비트스트림과 BSDL 스키마가 입력되면(비트스트림의 임의 지점에 정보 A와 정보 B가 존재하는 것으로 가정함), BSDL 파서(320)은 BSDL 스키마를 읽어 들어 정보 A에 해당하는 지점에 5bit의 MB type 데이터가 존재하고, 정보 B에 해당하는 지점에는 2bit의 CBPY 데이터가 존재함을 인식한다.
- <154> 이어서, BSDL 파서(320)는 파서는 인식한 정보를 이용해 각 지점에서 지정된 수 만큼의 비트를 읽어 들이고, 읽어들이는 정보를 부여된 의미에 따라 디코딩 솔루션(340)에 전달한다.
- <155> 디코딩 솔루션(340)은 BSDL 파서(320)로부터 MB Type과 CBPY로 명명된 데이터를 제공받아 처리하게 된다. 디코딩 솔루션(340)은 디코더 형성부(330)의 연결 제어에 의해 각 기능부들이 로딩되어 구현됨은 앞서 설명한 바와 같다.
- <156> 디코딩 솔루션(340)에 존재하는 데이터 인터페이스는 외부로부터 전달된 데이터를 받아들여, 연결 제어 정보에 의해 사전에 구성된 기능부들의 연결 관계를 참조하여, 해당 데이터를 요구하는 기능부들에 전달한다.
- <157> 각 기능부는 역시 미리 지정된 연결 관계(즉, 데이터 처리를 위한 연결 관계)에 따라 디코딩 과정을 수행한다. 모든 데이터 흐름과 기능부 간의 연결 관계는 디코더 형성부(330)가 사전에 구성한 내역에 의한다. 각 기능부들의 순차적 처리에 의해 출력 영상 프레임이 외부로 출력된다.
- <158> 상술한 바와 같이, 디코더 형성부(330) 또는/및 디코딩 솔루션(340) 내에는 저장부가 구비될 수 있다. BSDL 파서(320)로부터 데이터를 제공받음에 있어, 그 전달 과정에 끊김이 없고 또한 디코딩 과정과는 병렬적으로 데이터 제공이 수행될 수 있기 때문이다. 또한, 각 기능부는 필요한 데이터를 저장부로부터 독출하여 사용할 수도 있을 것이다.
- <159> 또한, BSDL 파서(320)는 인코딩된 영상 데이터의 디코딩 처리를 위해 상응하는 데이터를 디코더 형성부(330)로 제공하여 디코더 형성부(330)가 디코딩 솔루션(340)으로 제공하도록 하거나, BSDL 파서(320)가 직접 해당 데이터를 디코딩 솔루션(340)으로 제공할 수도 있을 것이다.
- <160> 다시 도 4를 참조하면, 분리부(310)는 입력된 디코더 디스크립션(2560)을 각각의 정보로 분리하여 디코딩 유닛(305)으로 입력한다. 분리부(310)에 입력된 디코더 디스크립션(2560)은 비트스트림의 구조를 기술하기 위한 BSDL 스키마(2565)와 비트스트림의 디코딩 과정을 기술하기 위한 CALML 데이터(2570)를 포함할 수 있다. 상술한 두 가지 종류의 데이터는 각각 독립적으로 XML 문법에 의해 기술될 수도 있으며, 효율적인 디코더의 운용을 위

하여 두 종류의 데이터가 통합되어 전송될 수 있다.

- <161> 도 6는 본 발명의 바람직한 다른 실시예에 따른 디코더 디스크립션 입력 과정을 나타낸 도면이다.
- <162> 도 6에 예시된 바와 같이, 복호화기(300)는 디스크립션 디코더(510)를 더 포함할 수 있다. 디스크립션 디코더(510)는 입력된 인코딩된 디코더 디스크립션(520)을 디코딩 처리하여 디코더 디스크립션(2560)을 생성하여 분리부(310)로 제공할 수 있다.
- <163> 디코더 디스크립션(2560)을 인코딩하여 송수신함으로써 송수신되는 데이터량을 절감할 수 있는 효과가 있다.
- <164> 도 7은 본 발명의 다른 실시예에 따른 디코더 디스크립션 입력 과정을 나타낸 도면이다.
- <165> 앞서 도 4을 참조하여 디코더 디스크립션(2560)과 영상 비트스트림이 디코딩 유닛(305)으로 입력되는 경우와, 도 6를 참조하여 인코딩된 디코더 디스크립션(520)과 영상 비트스트림(380)이 디코딩 유닛(305)으로 입력되는 경우를 설명하였다.
- <166> 그러나, 도 7에 예시된 바와 같이, 디코더 디스크립션(2560)의 구성 정보들이 원시적으로 분리되어 디코딩 유닛(305)으로 입력될 수도 있음은 자명하다. 이 경우, 앞서 설명한 분리부(310), 디코더 디스크립션(2560) 등이 생략될 수 있음은 자명하다. 도 7 31은 본 발명의 또 다른 실시예에 따른 디코딩 유닛의 구성을 나타낸 도면이다.
- <167> 앞서 도 4 내지 도 7을 참조하여 툴박스(335) 및 디코더 형성부(330)가 분리되어 구현된 디코딩 유닛(305)에 대하여 설명하였다.
- <168> 그러나 도 8 에 예시된 바와 같이, 툴박스(335)가 디코더 형성부(330)의 일 구성 요소로 포함되어 구현될 수도 있음은 자명하다.
- <169> 이 경우, 디코더 형성부(330)는 기능부간의 연결 구조 제어 기능 뿐 아니라 사용될 기능부의 선별 기능까지도 포함할 수 있으며, 이를 통해 구현 가능한 디코딩 솔루션(340)의 유형도 다양해질 수 있을 것이다.
- <170> 도 9는 본 발명의 또 다른 실시예에 따른 BSDL 파서의 구성을 나타낸 도면이다.
- <171> 앞서 도 4을 참조하여, BSDL 해석 처리부를 포함하는 BSDL 파서(320)에 대하여 설명하였다.
- <172> 그러나, 본 발명에 따른 BSDL 파서(320)는 비트스트림의 디코딩을 개시하기 이전에 복호화기(300) 외부로부터 사전 정의되어 제공될 수도 있다. 따라서, 앞서 설명한 BSDL 해석 처리부가 생략될 수 있다. 이때, BSDL 파서 제작기(2610)는 BSDL 레퍼런스 소프트웨어와 같은 기존의 응용 프로그램을 활용하여 구성될 수 있을 것이다.
- <173> 이제까지, BSDL 파서가 독립된 구성 요소로서 지정된 동작을 처리하는 경우를 중심으로 설명하였다. 다만, BSDL 파서는 툴박스 내에 포함되는 하나의 기능부로서 구현되거나, 디코딩 솔루션 내에 독립된 구성 요소로서 미리 포함되도록 구현될 수도 있을 것이다. 만일, BSDL 파서가 툴박스 내에 구비되는 경우 디코더 형성부는 연결 제어 정보를 이용하여 BSDL 파서가 비트스트림 디코딩을 위해 동작하는 기능부들의 동작 이전에 프로세스를 수행하도록 로드 및 제어하여야 할 것이다. 마찬가지로, BSDL 파서가 디코딩 솔루션 내에 미리 포함되는 경우, 디코더 형성부는 로드한 각 기능부들의 프로세스 수행 개시 이전에 BSDL 파서가 먼저 프로세스 수행하도록 제어하여야 할 것이다. 각각의 경우에도 BSDL 파서의 동작 및 기능은 앞서 관련 도면을 참조하여 설명한 바와 동일하므로 상세한 설명은 생략하기로 한다. 다만, BSDL 스키마 또는/및 비트스트림을 초기에 입력받는 주체가 디코더 형성부 또는/및 디코딩 솔루션으로 변경될 필요가 있을 수 있다.
- <174> 이제까지 본 발명에 따른 복호화 장치 및 비트스트림 복호화를 위한 구문 해석 방법을 설명함에 있어 MPEG-4 AVC를 기준으로 설명하였으나, MPEG-1, MPEG-2, MPEG-4, AVS 및 이외의 동영상 부호화/복호화 표준에 아무런 제한없이 동일하게 적용할 수 있음은 당연하다.
- <175> 또한, 연결 제어 정보에 포함되는 정보 역시 하나의 표준에 의한 디코딩 수행을 위한 기능부들의 연결 관계, 해당 기능부에 요구되는 처리 프로세스 등에 관한 정보만으로 기술되지 않고, 복수의 표준에 의한 디코딩 수행을 위한 정보로 기술될 수도 있음은 자명하다.
- <176> 예를 들어, 영상 비트스트림의 초기 복수의 프레임은 MPEG-2로 인코딩되고, 후속하는 복수의 프레임은 MPEG-4로 인코딩되며, 나머지 프레임은 MPEG-1으로 인코딩되었다고 가정하자. 이 경우, 인코딩된 영상 데이터의 디코딩을 위해 연결 제어 정보는 인코딩 방법을 달리하는 각 프레임들이 툴박스(335)에 포함된 각 표준에 따른 기능부들이 유기적으로 결합되어 동작될 수 있도록 정의될 것임은 자명하다.
- <177> 이하, 관련 도면을 참조하여, 본 발명에 따른 복호화기의 다른 실시예를 설명하기로 한다. 다만, 다른 실시예를

설명함에 있어, 전술한 실시예와 동일하거나 극히 유사한 기능을 수행하는 구성에 대한 설명은 그 명칭 및 도면 참조 부호를 동일하게 표시하는 것으로서 중복된 설명을 생략하기로 한다. 예를 들어, 도 11에 도시된 툴박스(335), 디코더 형성부(330) 및 디코딩 솔루션(330)은 기본적으로 전술한 구성과 동일하다.

- <178> 도 10은 본 발명에 따른 복호화기의 제 2 실시예 블록 구성도이고, 도 11은 도 10의 복호화부의 일실시예 구성을 개략적으로 나타낸 도면이다.
- <179> 이하, 설명되는 실시예에서 입력되는 디코더 디스크립션은 테이블 기반 디코더 디스크립션(이하, CDDL(Compact Decoder Description Language) 기반 디코더 디스크립션)으로서, 바이너리 형태로 압축되어 복호기로 전달된다.
- <180> 복호화기는 입력되는 비트스트림의 형식에 따라 필요한 기능부들을 조합하여 디코더를 재구성할 수 있다. 이로 인해, 본 발명에 따른 복호화기는 어떠한 형식으로 인코딩된 비트스트림이 입력되더라도 입력된 비트스트림에 상응하는 디코더를 형성하여 당해 비트스트림을 복호할 수 있다. 이하, 관련 도면을 참조하여 본 발명에 따른 복호화기에 대해 보다 상세히 설명하기로 한다.
- <181> 전술한 바와 같이, 복호화기로 디코더 디스크립션이 비트스트림과 함께 제공된다. 디코더 디스크립션은 비트스트림과 통합적으로 구현된 비트스트림(305)의 형태로 복호화기에 제공되거나, 독립된 데이터 형태로 복호화기에 제공될 수 있다. 물론, 복호화기의 특정 저장부에 디코더 디스크립션에 상응하는 정보가 미리 저장된 경우라면, 디코더 디스크립션의 제공은 생략될 수도 있다. 다만, 이하에서는 해당 데이터가 비트스트림 내에 포함되어 복호화기로 제공되는 경우를 중심으로 설명하기로 한다.
- <182> 본 발명의 다른 실시예에 따른 복호화기는 분리부(310) 및 복호화부(320)를 포함한다. 도시된 복호화기의 구성 요소(예를 들어, 분리부(310), 복호화부(320) 자체 또는 복호화부(320)에 포함된 하나 이상의 구성 요소 등) 중 하나 이상은 하기에서 설명될 기능을 수행하도록 구현된 소프트웨어 프로그램(또는 프로그램 코드들의 조합)으로 구현될 수도 있음은 자명하다.
- <183> 분리부(310)는 입력된 비트스트림(Bitstream)을 압축된 디코더 디스크립션(Compressed Decoder Description)과 인코딩된 비디오 데이터로 분리하여 복호화부(320)로 각각 입력한다.
- <184> 분리부(310)는 압축된 디코더 디스크립션(313)을 복원부(410)로 입력하고, 인코딩된 비디오 데이터(316)는 복호 실행부(450)로 입력할 수 있다. 상술한 바와 같이, 압축된 디코더 디스크립션과 인코딩된 비디오 데이터가 각각 독립된 데이터로 입력되는 경우 분리부(310)는 생략될 수 있다. 또한, 인코딩된 비디오 데이터는 앞서 도 1의 비트스트림(105)과 동일 또는 유사한 형식의 데이터일 수 있다.
- <185> 복호화부(320)는 분리부(310)로부터 입력된 압축된 디코더 디스크립션(313)을 이용하여 인코딩된 비디오 데이터(316)를 복호화하여 디코딩된 영상 데이터(390)를 출력한다. 이하에서 도 11를 참조하여, 복호화부(320)를 구성에 대해 상세히 설명하도록 한다.
- <186> 도 11을 참조하면, 복호화부(1320)는 복원부(Decompression Unit)(1410), 디코더 디스크립션 해석부(1420), 툴박스(335), ADM 생성부(1440), 복호 실행부(1450)를 포함한다. 복호 실행부(1450)는 디코더 형성부(330) 및 디코딩 솔루션(330)을 포함한다.
- <187> 복원부(1410)는 분리부(1310)로부터 입력된 압축된 디코더 디스크립션(313)을 소정의 복원 방식에 따라 복원(decompression)하여 디코더 디스크립션 해석부(420)로 출력한다. 더욱 구체적으로는, 디코더 디스크립션은 소정의 규칙에 따라 바이너리 형식으로 압축되어 있으며, 복원부는 바이너리 형태의 압축된 디코더 디스크립션을 디코딩하여 상응하는 CDDL 기반 디코더 디스크립션을 복원하여 출력한다.
- <188> 디코더 디스크립션 해석부(1420)는 복원된 CDDL 기반 디코더 디스크립션을 XML 형식으로 기술된 XML 기반 디코더 디스크립션으로 변환하여 ADM 생성부(1440)로 출력한다. 물론, 디코더 디스크립션 해석부(420)를 생략하고 복원부(410)가 입력된 바이너리 데이터를 직접 XML 기반 디코더 디스크립션으로 변환하여 출력할 수도 있음은 자명하다.
- <189> ADM 생성부(440)는 하나 이상의 입출력 포트를 갖는 다수의 기능부 정보 및 상기 포트들 간의 연결 정보를 포함하는 추상적 복호화 모델(Abstact Decoding Model, ADM)을 생성한다.
- <190> ADM 생성부(440)는 전송받은 기능부를 문맥 제어 정보, 연결 제어 정보, 파싱 제어 정보에 따라 조합하여 비트스트림을 복호화 할 수 있는 ADM을 생성한다.
- <191> 복호 실행부(450)는 상기 ADM을 이용하여 전술한 바와 같이 디코더형성부(330) 및 디코딩 솔루션(340)을 통해

툴박스에 저장된 해당 기능부들을 이용하여 입력 영상을 디코딩 할 수 있다. 또한, 복호 실행부(450)는 상기 XML 기반 디코더 디스크립션을 직접 입력받아 입력 영상을 디코딩할 수도 있음은 물론이다.

<192> 이하, 본 발명의 일실시예에 따른 CDDL 기반 DD(Decoder Description)를 설명하기로 한다.

<193> CDDL 기반 DD(Decoder Description)은 ADM 또는/및 디코딩 솔루션을 구성하는데 필요한 FU 들의 연결 관계인 FU 네트워킹을 기술하는 FU 네트워킹 테이블 그룹 및 신택스 파싱을 위한 신택스 파싱 테이블 그룹으로 구분된다.

<194> FU 네트워킹 테이블 그룹은 가상 네트워크 테이블(Virtual Network Table, VNT), 기능부 인스턴스 테이블(Functional Unit Instance Table, FUIT), 네트워크 연결 테이블(Network Connection Table, NCT), 파라미터 테이블(Parameter Table, PT), 표현 테이블(Expression Table, ET) 및 유형 테이블(Type Table, TT)을 포함한다.

<195> 신택스 파싱 테이블 그룹은 CSCIT(Control Signal and Context Information Table), SET(Syntax Element Table), SRT(Syntax Rule Table) 및 DVT(Default Value Table)를 포함한다.

<196> 이하, 관련 도면을 참조하여 FU 네트워킹 테이블들에 대하여 먼저 설명한 후, 신택스 파싱 테이블들을 설명한다.

<197> 도 12는 본 발명의 일실시예에 따른 FU 네트워킹 테이블의 구조를 나타낸 도면이다.

<198> CDDL 기반 DD(Decoder Description)에서 사용하는 FU 네트워킹 테이블 은 도 12에 도시된 바와 같은 구조에 따라 이진 표현 비트스트림으로 기술된다.

<199> 각 테이블은 테이블 시작 코드(Table Start Code)와 테이블 식별을 위한 고유한 테이블 번호(Table Code)가 기술된다. 그리고 실질적인 테이블의 내용이 이진화(Binary)되어 비트스트림을 구성한다. 하나의 테이블 내용이 완전하게, 즉, 하나의 테이블 내용 전체가 비트스트림에 기록되면 테이블 마침 코드(Table End Code)가 추가된다.

<200> 테이블 시작 코드(Table Start Code)는 고정된 값이며 24비트의 2진수(1111111111111111111111110)로 이루어져 있고, Table End Code는 고정된 값이며 24비트의 2진수(1111111111111111111111111)로 이루어져 있고, 테이블 코드(Table Code)는 고유한 테이블 번호를 식별하기 위하여 아래의 표 1과 같이 4비트의 2진수로 이루어져 있다.

<201> [표 1]

Table	Code
VNT	0100
NCT	0101
PT	0110
FUIT	0111
TT	1000
ET	1001
reserved	1010 ~ 1111

<203> 도 13는 본 발명의 일실시예에 따른 VNT 테이블의 구조를 나타낸 도면이다.

<204> 도 13에 도시된 바와 같이, VNT 테이블은 FUID, VN 명칭(VN name), 입력포트(INPUT ports), 출력 포트(OUTPUT Ports) 및 QID 명칭(QID name) 필드를 포함한다.

<205> 기능부의 집합인 Network는 구조가 동일하면서 입력이나 출력으로 쓰이는 데이터가 동일하지 않은 경우가 존재할 수 있다. 이렇게 구조가 동일하지만 입/출력이 동일하지 않은 여러 개의 Network(개체들)을 효율적으로 구현하기 위하여 상속 개념을 이용한 기본 템플릿(Template) 객체를 생성하여 두고 이 템플릿(Template)을 모체로 하여 필요한 하위 객체(자손, 상속된 객체)를 만들어내는 형태를 구성한다. VNT는 기본 템플릿(Template)에 해당하는 정보를 기술한다.

<206> FUID는 기능부(FU)가 이미 도구 박스(toolbox)에 존재할 수도 있고 존재하지 않는 새로운 것 일수도 있기 때문에, 기능부가 도구 박스에서 몇 번째 항목인지를 나타내는 인덱스 번호를 사용한다. 도구 박스에 존재할 경우에

는 FUID flag 가 1로 세팅되고 존재하지 않는 경우에는 0으로 세팅된다. FUID flag가 0으로 세팅되는 경우 FUID 가 기술되지 않고 넘어간다.

- <207> VN name은 기능부(FU)의 이름을 나타낸다. 2진화되어 비트스트림에 기록될 때에는 이름의 문자열 길이를 먼저 기술하고(8비트), 그 이후에 각 영문자를 하나씩 기록한다. 각 영문자는 아스키 코드(Ascii code)로 표현될 수 있다.
- <208> Version은 기능부(FU)의 버전을 나타낸다. 기능부의 버전이 존재하는 경우에는 version flag가 1로 세팅되고 존재하지 않는 경우에는 0으로 세팅된다. 2진화되어 비트스트림에 기록될 때에는 버전의 문자열 길이를 먼저 기술하고(8비트), 그 이후에 각 영문자를 하나씩 기록한다. 각 영문자는 아스키 코드(Ascii code)로 표현될 수 있다.
- <209> 입력 포트는 여러 개일 수가 있다. 하나의 입력 포트 이름을 2진화시켜 기록하기 전에 입력 포트 이름의 길이를 8비트로 표시하고, 그 이후에 입력 포트의 영문자를 하나씩 기록한다. 입력 포트의 타입이 존재할 수도 있고 존재하지 않을 수도 있기 때문에, 입력 포트에 대한 type flag가 존재 여부를 나타낸다. type flag가 1으로 세팅되는 경우 Type 테이블 인덱스가 8비트로 기술되고, 그렇지 않으면 Type 테이블 인덱스가 기술되지 않고 넘어간다. 여러 개의 입력 포트가 존재하는 경우, 하나의 입력 포트에 대한 기술이 끝나면, 전체 입력 포트의 기록이 완료되었는지 안되었는지를 플래그(INPUT ports flag)로 나타낸다. 만약 모든 입력 포트의 기술이 끝났으면 1로 세팅하고, 그렇지 않고 더 기록할 입력 포트가 존재한다면 0으로 세팅한다. 이후에 동일한 방식으로 남아있는 입력 포트를 기록한다.
- <210> 출력 포트 또한 여러 개일 수 있으며, 2진 표현으로 비트스트림에 기록하는 형태는 입력 포트와 동일하다.
- <211> QID 명칭(QID name)은 기능부(FU)의 적절한 식별자로서 사용되는 문자열을 나타낸다. 모든 기능부가 QID 이름을 가지고 있지는 않기 때문에 QID 이름이 존재하는지 여부를 1비트로 기술한다. flag가 0으로 세팅되는 경우 QID 가 기술되지 않고 넘어간다. flag가 1로 세팅되는 경우 QID 이름의 길이를 8비트로 기술한다. 그리고 나서 QID 이름을 하나씩 영문자로 기록한다. 각 영문자는 아스키 코드(Ascii code)로 표현될 수 있다.
- <212> 전술한 바와 같은 구조의 바이너리 비트스트림으로 입력된 VNT 테이블은 복원부(1410) 및/또는 디코더 디스크립션 해석부(1420)에서 소정의 규칙에 따라 XML 기반 디코더 디스크립션으로 변환된다. 즉, 테이블을 나타내는 테이블 약자(영문자)가 이용되고, 테이블 내용은 '{' 로 시작되며 '}' 로 끝이 난다. 각 테이블의 필드는 ',' 로 구분되며, 하나의 행은 '(' 로 시작되며 ')' 로 끝이 나고, 하나의 필드가 여러 개의 값을 가지는 경우 '{' 로 시작되며 '}' 로 끝이 나며, 문자의 표현은 '"' 마크 사이에 기술된다.
- <213> VNT 테이블은 다음과 같은 텍스트형식으로 표현된다.
- <214> {
- <215> ([FU_ID,] (VN Name), (Version),{INPUT PORTS}, {OUTPUT PORTS}, [QID name]),
- <216> ...
- <217> }
- <218> XML 형식을 지원하는 MPEG-4 SP에 대한 VNT 테이블의 실시예는 다음과 같다.
- <219> VNT
- <220> {
- <221> //0
- <222> (-, "decoder", -, "mpeg", -, {"video_Y", -, "video_U", -, "video_V", -}, "mpeg4"),
- <223> (1, "parser", -, "BITS", -, {"BTYPY_Y", -, "BTYPY_U", -, "BTYPY_V", -, "MV_Y", -, "MV_U", -, "MV_V", -, "B_Y", -, "B_U", -, "B_V", -},-),
- <224> (-, "intra_FUs_16x16_Y", -, {"BTYPY", -, "QFS", -}, "f", -, "mpeg4"),
- <225> (-, "intra_FUs_8x8_C", -, {"BTYPY", -, "QFS", -}, "f", -, "mpeg4"),
- <226> (-, "motion_Y", -, {"MV", -, "BTYPY", -, "TEX", -}, "VID", -, "mpeg4"),

<227> //5

<228> (-, "motion_U", -, {"MV", -, "BTYP", -, "TEX", -}, "VID", -, "mpeg4"),

<229> (-, "motion_V", -, {"MV", -, "BTYP", -, "TEX", -}, "VID", -, "mpeg4"),

<230> (-, "byte2bit", -, "in8", -, "BITS", -, -),

<231> (2, "Address16x16", -, {"MV", -, "BTYP", -}, {"WA", -, "RA", -, "halfpel", -}, -),

<232> (3, "Framebuf", -, {"RA", -, "WA", -, "WD", -}, "RD", -, -),

<233> //10

<234> (4, "Interpolate", -, {"RD", -, "halfpel", -}, "MOT", -, -),

<235> (5, "Add", -, {"MOT", -, "TEX", -, "BTYP", -}, "VID", -, -),

<236> (6, "Address8x8", -, {"MV", -, "BTYP", -}, {"WA", -, "RA", -, "halfpel", -}, -),

<237> (7, "DCSplit", -, "IN", -, {"DC", -, "AC", -}, -),

<238> (-, "DCR_16x16_L", -, {"BTYP", -, "QFS_DC", -}, {"QF_DC", -, "PTR", -, "AC_PRED_DIR", -, "SIGNED", -, "QUANT", -}, "cal"),

<239> //15

<240> (8, "InverseScan", -, {"QFS_AC", -, "AC_PRED_DIR", -}, "PQF_AC", -, -),

<241> (9, "IAP_16x16", -, {"PQF_AC", -, "PTR", -, "AC_PRED_DIR", -}, "QF_AC", -, -),

<242> (10, "Dequant", -, {"AC", -, "DC", -, "QP", -}, "OUT", -, -),

<243> (11, "idct2d", -, {"WinW", -, "signed", -}, "out", -, -),

<244> (-, "DCR_8x8_C", -, {"BTYP", -, "QFS_DC", -}, {"QF_DC", -, "PTR", -, "AC_PRED_DIR", -, "SIGNED", -, "QUANT", -}, "cal"),

<245> //20

<246> (12, "IAP_8x8", -, {"PQF_AC", -, "PTR", -, "AC_PRED_DIR", -}, "QF_AC", -, -),

<247> (13, "DCR_addressing_8x8", -, "BTYP", -, {"A", -, "B", -, "C", -}, -),

<248> (14, "DCR_invpred_8x8_C", -, {"BTYP", -, "A", -, "B", -, "C", -, "QFS_DC", -}, {"QF_DC", -, "PTR", -, "AC_PRED_DIR", -, "SIGNED", -, "QUANT", -}, -),

<249> (15, "DCR_addressing_16x16", -, "BTYP", -, {"A", -, "B", -, "C", -}, -),

<250> (16, "DCR_invpred_16x16_L", -, {"BTYP", -, "A", -, "B", -, "C", -, "QFS_DC", -}, {"QF_DC", -, "PTR", -, "AC_PRED_DIR", -, "SIGNED", -, "QUANT", -}, -)

<251> }

<252> 도 14는 본 발명의 일실시예에 따른 FUIT 테이블의 구조를 나타낸 도면이다.

<253> 도 14에 도시된 바와 같이, FUIT 테이블은 VNT index, Parent VNT index, Parameter indices 및 QID name 필드를 포함한다. FUIT에서는 VNT 정보인 기본 템플릿(Template)에 기반하여 실제 필요로 하는 Network로 사용되는 객체들을 생성하기 위한 주요 정보가 저장된다. 실제 필요로 하는 Network는 기본 템플릿(Template)에서 파생된 하위 객체(자손, 상속된 객체, Instance)로 표현할 수 있다.

<254> VNT index는 하나의 기능부를 나타내기 위한 VNT 테이블의 인덱스이다.

<255> Parent VNT index는 하위 객체(Instance)가 필요로 하는 기본 템플릿을 나타낸다. VNT 테이블에서 참조하기 위한 인덱스이다. flag가 0으로 세팅되는 경우 VNT index가 기술되지 않고 넘어간다.

<256> PT indices는 하나의 기능부가 가지는 여러개의 파라미터를 나타내는 인덱스 리스트로서, 기능부가 가지는 파라

미터는 없을 수도 있고, 여러 개 일 수도 있다. 파라미터의 존재 여부를 flag 로 나타내고, 파라미터 정보가 담긴 파라미터 테이블(PT) 인덱스를 이어서 기술한다. 만약 모든 파라미터 정보의 기술이 끝났으면 1로 세팅하고, 그렇지 않고 더 기록할 파라미터 테이블 인덱스가 존재한다면 0으로 세팅한다. 이후에 동일한 방식으로 남아있는 파라미터 테이블 인덱스를 기록한다.

<257> QID name은 기능부(FU)의 적절한 식별자로서 사용되는 문자열을 나타낸다. 모든 기능부가 QID 이름을 가지고 있지는 않기 때문에 QID 이름이 존재하는지 여부를 1비트로 기술한다. flag가 0으로 세팅되는 경우 QID가 기술되지 않고 넘어간다. flag가 1로 세팅되는 경우 QID 이름의 길이를 8비트로 기술한다. 그리고 나서 QID 이름을 하나씩 영문자로 기록한다. 각 영문자는 아스키 코드(Ascii code)로 표현될 수 있다.

<258> FUIT 테이블은 다음과 같은 텍스트형식으로 표현된다.

```
<259> {
<260>     (VNT Index, [Parent VNT Index], {PT Indexes}, [QID name] ),
<261>     ...
<262> }
```

<263> XML 형식을 지원하는 MPEG-4 SP에 대한 FUIT 테이블의 실시예는 다음과 같다.

```
<264> FUIT
<265> {
<266>     (1, 0, {21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37}, "mpeg4"),
<267>     (2, 0, {21, 24, 25, 23, 26, 27, 28, 29, 33, 34, 39}, "mpeg4"),
<268>     (3, 0, {21, 24, 25, 23, 26, 27, 28, 29, 33, 34, 39}, "mpeg4"),
<269>     (3, 0, {21, 24, 25, 23, 26, 27, 28, 29, 33, 34, 39}, "mpeg4"),
<270>     (4, 0, {38, 21, 40, 37, 49, 41, 7, 25, 26, 27, 29, 30, 35}, "mpeg4"),
<271>     ...
<272>     ...
<273>     (24, 14, {52, 21, 24, 25, 23, 26, 27, 28, 29, 33, 34, 39}, "cal")
<274> }
```

<275> 도 15는 본 발명의 일 실시예에 따른 PT 테이블의 구조를 나타낸 도면이다.

<276> 도 15에 도시된 바와 같이, PT 테이블은 Parameter name, Parent VNT index, ET index 및 TT index 필드를 포함한다.

<277> Parameter는 Bitstream으로부터 생성되는 Syntax는 아니지만 기능부가 구성될 때 필요한 데이터를 생성하기 위해 사용된다. Parameter 는 Syntax가 아니기 때문에 데이터를 전송하는 Port를 통해 전달될 수 없다. 기능부의 집합인 Network 에서 그 Network에 포함되어 있는 하위 Network로 Parameter를 전달할 수 있다. 하나의 기능부가 가지는 Parameter에 대한 정보는 FUIT에 PT 인덱스로 기술된다. PT에서는 각 Parameter의 생성 정보를 가진다.

<278> Parameter name에 있어서, 각 파라미터는 기본적으로 그 이름을 가진다. 파라미터 이름의 길이를 8비트로 기술한다. 그리고 나서 파라미터 이름을 하나씩 영문자로 기록한다. 각 영문자는 아스키 코드(Ascii code)로 표현될 수 있다

<279> Parent VNT index에 있어서, 파라미터는 기본 템플릿이 되는 상위의 Network 기능 유닛에서 사용될 수 있고, 기본 템플릿을 통해 생성된 하위 기능 유닛 객체(Instance)에서 사용될 수도 있다. 상위의 Network 에서 사용되는 파라미터일 경우에는 flag가 1으로 세팅되며 VNT 테이블에서 참조하기 위한 인덱스가 기술된다. 그렇지 않고 하위 객체(Instance)에서 사용되는 경우에는 flag가 0으로 세팅되고 VNT 테이블 인덱스가 기술되지 않는다.

<280> ET index에 있어서 파라미터의 형식을 나타내는 표현이 이루어질 수 있다. 이때 파라미터에 대한 표현이 존재하

는 경우 flag 가 1로 세팅되고 존재하지 않는 경우에는 0으로 세팅된다. flag가 0으로 세팅되는 경우 ET 인덱스가 기술되지 않고 넘어간다. 표현에 대한 내용은 ET의 인덱스를 참조하게 된다.

<281> TT index는 파라미터의 타입(파라미터의 종류)을 나타낸다. 파라미터의 타입이 존재하는지를 나타내는 flag가 1로 세팅되면 TT 인덱스가 기술되고, 그렇지 않으면 TT 인덱스가 기술되지 않고 넘어간다. 타입에 대한 내용은 TT의 인덱스를 참조하게 된다.

<282> PT 테이블은 다음과 같은 텍스트형식으로 표현된다.

```
<283> {
<284>     (Parameter Name, [VNT Index], [Expr Index], [Type Index]),
<285>     ...
<286> }
```

<287> XML 형식을 지원하는 MPEG-4 SP에 대한 PT 테이블의 실시예는 다음과 같다.

```
<288> PT
<289> {
<290>     //0
<291>     ("MAXW_IN_MB", 0, 0, -),
<292>     ("MAXH_IN_MB", 0, 1, -),
<293>     ("ADDR_SZ", 0, 2, -),
<294>     ("PIX_SZ", 0, 3, -),
<295>     ("MV_SZ", 0, 3, -),
<296>     //5
<297>     ("SAMPLE_COUNT_SZ", 0, 4, -),
<298>     ("SAMPLE_SZ", 0, 5, -),
<299>     ("MB_COORD_SZ", 0, 4, -),
<300>     ("BTYPE_SZ", 0, 0, -),
<301>     ("NEWVOP", 0, 6, -),
<302>     //10
<303>     ("INTRA", 0, 7, -),
<304>     ("INTER", 0, 8, -),
<305>     ("QUANT_MASK", 0, 9, -),
<306>     ("ROUND_TYPE", 0, 10, -),
<307>     ("FCODE_MASK", 0, 11, -),
<308>     //15
<309>     ("FCODE_SHIFT", 0, 12, -),
<310>     ("ACPRED", 0, 13, -),
<311>     ("ACCODED", 0, 14, -),
<312>     ("FOURMV", 0, 15, -),
<313>     ("MOTION", 0, 4, -),
```

<314> //20
 <315> ("QUANT_SZ", 0, 12, -),
 <316> ("MAXW_IN_MB", -, 16, -),
 <317> ("SAMPLE_COUNT_SZ", -, 17, -),
 <318> ("SAMPLE_SZ", -, 18, -),
 <319> ("MB_COORD_SZ", -, 19, -),
 <320> //25
 <321> ("BTYPE_SZ", -, 20, -),
 <322> ("NEWVOP", -, 21, -),
 <323> ("INTRA", -, 22, -),
 <324> ("INTER", -, 23, -),
 <325> ("QUANT_MASK", -, 24, -),
 <326> //30
 <327> ("ROUND_TYPE", -, 25, -),
 <328> ("FCODE_MASK", -, 26, -),
 <329> ("FCODE_SHIFT", -, 27, -),
 <330> ("ACPRED", -, 28, -),
 <331> ("ACCODED", -, 29, -),
 <332> //35
 <333> ("MOTION", -, 30, -),
 <334> ("FOURMV", -, 31, -),
 <335> ("MV_SZ", -, 32, -),
 <336> ("SEARCHWIN_IN_MB", -, 33, -),
 <337> ("QUANT_SZ", -, 34, -),
 <338> //40
 <339> ("MAXH_IN_MB", -, 35, -),
 <340> ("PIX_SZ", -, 36, -),
 <341> ("BUF_SZ", 4, 37, -),
 <342> ("FLAG_SZ", 4, 15, -),
 <343> ("BUF_SZ", 5, 37, -),
 <344> //45
 <345> ("FLAG_SZ", 5, 15, -),
 <346> ("BUF_SZ", 6, 37, -),
 <347> ("FLAG_SZ", 6, 15, -),
 <348> ("FLAG_SZ", -, 38, -),
 <349> ("ADDR_SZ", -, 39, -),

- <350> //50
- <351> ("BUF_SZ", -, 40, -),
- <352> ("SEARCHWIN_IN_MB", -, 41, -),
- <353> ("DCVAL", -, 7, -)
- <354> }
- <355> 도 16은 본 발명의 일실시예에 따른 NCT 테이블의 구조를 나타낸 도면이다.
- <356> 도 16에 도시된 바와 같이, NCT 테이블은 Src FUIT index, Dst FUIT index, Src port name, Dst port name, Attribute name, Attribute value 및 ET Index 필드를 포함한다.
- <357> NCT에는 각각의 기능 유닛들 간에 데이터를 전송할 수 있는 통로인 Port에 관한 정보를 가지고 있다. 각각의 입력과 출력을 담당하는 Input Port와 Output Port들은 고유한 이름을 가질 수 있다.
- <358> Src FUIT index는 데이터가 하나의 기능 유닛에서 다른 하나로 전송될 때 그 데이터가 어디에서 오는지를 나타내야 한다. 데이터가 전송되기 시작하는 기능 유닛을 FUIT의 인덱스로 참조할 수 있다. 만약 비트스트림의 시작이 입력 파일 파일과 같이 기능 유닛 자체가 아니라면 '-'로 표현될 수 있다. 따라서 FUIT를 참조하는 인덱스인지 아닌지를 flag로 나타내고 FUIT 인덱스를 기록한다. flag가 0으로 세팅되는 경우 FUIT 인덱스가 기술되지 않고 넘어간다.
- <359> Dst FUIT index는 데이터가 도달하는 기능 유닛을 나타내며, 데이터가 도달하는 기능 유닛의 FUIT 인덱스를 나타낸다. 만약 비트스트림의 시작이 출력 파일과 같이 기능 유닛 자체가 아니라면 '-'로 표현될 수 있다. 따라서 FUIT를 참조하는 인덱스인지 아닌지를 flag로 나타내고 FUIT 인덱스를 기록한다. flag가 0으로 세팅되는 경우 FUIT 인덱스가 기술되지 않고 넘어간다.
- <360> Src port name and Dst port name는 데이터가 전송되는 Port의 이름을 나타낸다. 기능 유닛의 출력 포트가 데이터의 전송 시작 포트가 되며, 다른 기능 유닛의 입력 포트가 데이터가 도달하는 포트가 된다. 데이터가 전송되는 출력 포트와 입력 포트의 이름이 동일할 수도 있다. 따라서 두개의 전송 포트 이름이 동일한지 동일하지 않은지를 나타내는 same flag가 기술된다. same flag가 0으로 세팅되는 경우 Dst port 이름이 기술되지 않는다. 2진화되어 비트스트림에 기록될 때에는 이름의 문자열 길이를 먼저 기술하고(8비트), 그 이후에 각 영문자를 하나씩 기록한다. 각 영문자는 아스키 코드(Ascii code)로 표현될 수 있다.
- <361> Attribute name는 네트워크의 연결관계에 있어서 연결 속성 정보가 추가될 수 있다. 그러한 속성 정보의 이름이 존재할 경우에는 flag가 1로 세팅되고 존재하지 않는 경우에는 0으로 세팅된다. flag가 0으로 세팅되는 경우 속성 이름이 기술되지 않고 넘어간다. 연결 속성 정보의 이름이 2진화되어 비트스트림에 기록될 때에는 이름의 문자열 길이를 먼저 기술하고(8비트), 그 이후에 각 영문자를 하나씩 기록한다. 각 영문자는 아스키 코드(Ascii code)로 표현될 수 있다.
- <362> Attribute value는 네트워크의 연결관계에서 연결 속성 정보의 이름이 존재하는 경우 그 값이 기술되는 경우가 있다. flag가 0으로 세팅되는 경우 속성 값이 기술되지 않고 넘어간다. 따라서 속성 값의 존재 여부를 flag로 나타내고, 값을 나타내는 문자열 길이를 8비트로 기술한다. 그 이후에 각 영문자를 하나씩 기록한다. 각 영문자는 아스키 코드(Ascii code)로 표현될 수 있다.
- <363> ET index는 네트워크의 연결관계의 추가적인 표현이 이루어질 수 있다. 이때 추가적인 표현이 존재하는 경우 flag가 1로 세팅되고 존재하지 않는 경우에는 0으로 세팅된다. flag가 0으로 세팅되는 경우 ET 인덱스가 기술되지 않고 넘어간다. 표현에 대한 내용은 ET의 인덱스를 참조하게 된다.
- <364> NCT 테이블은 다음과 같은 텍스트형식으로 표현된다.
- <365> {
- <366> ([Src FUIT Index], [Dst FUIT Index], Src Port, Dst Port, [Attribute Name], [Attribute Value], [Expr Index]),
- <367> ...
- <368> }

<369> XML 형식을 지원하는 MPEG-4 SP에 대한 NCT 테이블의 실시예는 다음과 같다.

<370> NCT

<371> {

<372> //FUIT 0-7

<373> (-, 7, "mpeg", "in8", -, -, -),

<374> (7, 0, "out", "BITS", -, -, -),

<375> (0, 1, "BTYPY_Y", "BTYPY", -, -, -),

<376> (0, 1, "B_Y", "QFS", -, -, -),

<377> (0, 2, "BTYPY_U", "BTYPY", -, -, -),

<378> (0, 2, "B_U", "QFS", -, -, -),

<379> (0, 3, "BTYPY_V", "BTYPY", -, -, -),

<380> (0, 3, "B_V", "QFS", -, -, -),

<381> (1, 4, "f", "TEX", -, -, -),

<382> (2, 5, "f", "TEX", -, -, -),

<383> (3, 6, "f", "TEX", -, -, -),

<384> (0, 4, "MV_Y", "MV", -, -, -),

<385> (0, 4, "BTYPY_Y", "BTYPY", -, -, -),

<386> (0, 5, "MV_U", "MV", -, -, -),

<387> (0, 5, "BTYPY_U", "BTYPY", -, -, -),

<388> (0, 6, "MV_V", "MV", -, -, -),

<389> (0, 6, "BTYPY_V", "BTYPY", -, -, -),

<390> (4, -, "VID", "video_Y", -, -, -),

<391> (5, -, "VID", "video_U", -, -, -),

<392> (6, -, "VID", "video_V", -, -, -),

<393> //FUIT 8 9 10 11

<394> (-, 8, "MV", "MV", -, -, -),

<395> (-, 8, "BTYPY", "BTYPY", -, -, -),

<396> (-, 11, "TEX", "TEX", -, -, -),

<397> (-, 11, "BTYPY", "BTYPY", -, -, -),

<398> (8, 9, "WA", "WA", -, -, -),

<399> (8, 9, "RA", "RA", -, -, -),

<400> (8, 10, "halfpel", "halfpel", -, -, -),

<401> (9, 10, "RD", "RD", -, -, -),

<402> (10, 11, "MOT", "MOT", -, -, -),

<403> (11, 9, "VID", "WD", -, -, -),

<404> (11, -, "VID", "VID", -, -, -),

<405> //FUIT 12 13 14 15
 <406> (-, 12, "MV", "MV", -, -, -),
 <407> (-, 12, "BTYP", "BTYP", -, -, -),
 <408> (-, 15, "TEX", "TEX", -, -, -),
 <409> (-, 15, "BTYP", "BTYP", -, -, -),
 <410> (12, 13, "WA", "WA", -, -, -),
 <411> (12, 13, "RA", "RA", -, -, -),
 <412> (12, 14, "halfpel", "halfpel", -, -, -),
 <413> (13, 14, "RD", "RD", -, -, -),
 <414> (14, 15, "MOT", "MOT", -, -, -),
 <415> (15, 13, "VID", "WD", -, -, -),
 <416> (15, -, "VID", "VID", -, -, -),
 <417> //FUIT 16 17 18 19
 <418> (-, 16, "MV", "MV", -, -, -),
 <419> (-, 16, "BTYP", "BTYP", -, -, -),
 <420> (-, 19, "TEX", "TEX", -, -, -),
 <421> (-, 19, "BTYP", "BTYP", -, -, -),
 <422> (16, 17, "WA", "WA", -, -, -),
 <423> (16, 17, "RA", "RA", -, -, -),
 <424> (16, 18, "halfpel", "halfpel", -, -, -),
 <425> (17, 18, "RD", "RD", -, -, -),
 <426> (18, 19, "MOT", "MOT", -, -, -),
 <427> (19, 17, "VID", "WD", -, -, -),
 <428> (19, -, "VID", "VID", -, -, -),
 <429> //FUIT 20 21 22 23 24 25
 <430> (-, 20, "QFS", "IN", -, -, -),
 <431> (20, 21, "DC", "QFS_DC", -, -, -),
 <432> (20, 22, "AC", "QFS_AC", -, -, -),
 <433> (-, 21, "BTYP", "BTYP", -, -, -),
 <434> (21, 22, "AC_PRED_DIR", "AC_PRED_DIR", -, -, -),
 <435> (22, 23, "PQF_AC", "PQF_AC", -, -, -),
 <436> (23, 24, "QF_AC", "QF_AC", -, -, -),
 <437> (24, 25, "OUT", "IN", -, -, -),
 <438> (21, 23, "PTR", "PTR", -, -, -),
 <439> (21, 23, "AC_PRED_DIR", "AC_PRED_DIR", -, -, -),
 <440> (21, 24, "QUANT", "QP", -, -, -),

<441> (21, 24, "QF_DC", "DC", -, -, -),
 <442> (21, 25, "signed", "signed", -, -, -),
 <443> (25, -, "out", "f", -, -, -),
 <444> //FUIT 26 27 28 29 30 31
 <445> (-, 26, "QFS", "IN", -, -, -),
 <446> (26, 27, "DC", "QFS_DC", -, -, -),
 <447> (26, 28, "AC", "QFS_AC", -, -, -),
 <448> (-, 27, "BTYP", "BTYP", -, -, -),
 <449> (27, 28, "AC_PRED_DIR", "AC_PRED_DIR", -, -, -),
 <450> (28, 29, "PQF_AC", "PQF_AC", -, -, -),
 <451> (29, 30, "QF_AC", "QF_AC", -, -, -),
 <452> (30, 31, "OUT", "IN", -, -, -),
 <453> (27, 29, "PTR", "PTR", -, -, -),
 <454> (27, 29, "AC_PRED_DIR", "AC_PRED_DIR", -, -, -),
 <455> (27, 30, "QUANT", "QP", -, -, -),
 <456> (27, 30, "QF_DC", "DC", -, -, -),
 <457> (27, 31, "signed", "signed", -, -, -),
 <458> (31, -, "out", "f", -, -, -),
 <459> //FUIT 32 33
 <460> (-, 32, "BTYP", "BTYP", -, -, -),
 <461> (-, 33, "QFS_DC", "QFS_DC", -, -, -),
 <462> (-, 33, "BTYP", "BTYP", -, -, -),
 <463> (32, 33, "A", "A", -, -, -),
 <464> (32, 33, "B", "B", -, -, -),
 <465> (32, 33, "C", "C", -, -, -),
 <466> (33, -, "PTR", "PTR", -, -, -),
 <467> (33, -, "AC_PRED_DIR", "AC_PRED_DIR", -, -, -),
 <468> (33, -, "SIGNED", "SIGNED", -, -, -),
 <469> (33, -, "QF_DC", "QF_DC", -, -, -),
 <470> (33, -, "QUANT", "QUANT", -, -, -),
 <471> //FUIT 34 35
 <472> (-, 34, "BTYP", "BTYP", -, -, -),
 <473> (-, 35, "QFS_DC", "QFS_DC", -, -, -),
 <474> (-, 35, "BTYP", "BTYP", -, -, -),
 <475> (34, 35, "A", "A", -, -, -),
 <476> (34, 35, "B", "B", -, -, -),

<477> (34, 35, "C", "C", -, -, -),

<478> (35, -, "PTR", "PTR", -, -, -),

<479> (35, -, "AC_PRED_DIR", "AC_PRED_DIR", -, -, -),

<480> (35, -, "SIGNED", "SIGNED", -, -, -),

<481> (35, -, "QF_DC", "QF_DC", -, -, -),

<482> (35, -, "QUANT", "QUANT", -, -, -)

<483> }

<484> 도 17은 본 발명의 일실시예에 따른 ET 테이블의 구조를 나타낸 도면이다.

<485> 도 17에 도시된 바와 같이, ET 테이블은 Kind, Literal kind, Literal value, Variable name, Operator, Child ET index, Args ET index 필드를 포함한다.

<486> ET 테이블은 Bitstream으로부터 얻어내는 정보가 아니며 구문(Syntax)의 표현을 위하여 참조되는 테이블이다. 표현을 위한 어떤 수식이나 정보를 나타내기 위하여 기술된다. 이 테이블은 표현에 사용되는 표현 기법의 종류와 값, 변수의 이름 등을 나타낸다.

<487> Kind는 표현식이 나타내는 종류를 의미한다. 이 필드는 비트스트림에 3비트로 기술된다. 표현식은 아래의 표와 같은 5가지 종류가 존재한다.

<488> [표 2]

<489>

Kind	Code
Literal	000
Var	001
Application	010
UnaryOp	011
BinOpSeq	100
Reserved	101 ~ 111

<490> Literal kind는 이전 Kind 필드의 값이 Literal 로 지정되었을 때에만 Literal 이 가질 수 있는 종류를 나타낸다. flag가 0으로 세팅되는 경우 Literal kind 가 기술되지 않고 넘어간다. Literal kind는 아래의 표 3과 같은 종류의 코드를 갖는다.

<491> [표 3]

<492>

Literal kind	Code
Boolean	000
Integer	001
Real	010
String	011
Character	100
Reserved	101 ~ 111

<493> Literal value는 이전 Kind 필드의 값이 Literal 로 지정되었을 때 Literal 이 가지는 값을 나타낸다. flag가 0으로 세팅되는 경우 Literal Value 가 기술되지 않고 넘어간다. flag가 1로 세팅되는 경우 Literal Value 의 길이를 8비트로 기술한다. 그리고 나서 Literal Value 를 하나씩 영문자로 기록한다. 각 영문자는 아스키 코드(Ascii code)로 표현될 수 있다.

<494> Variable name은 이전 Kind 필드의 값이 Var 로 지정되었을 때 Variable 의 이름을 나타낸다. flag가 0으로 세팅되는 경우 Variable name 이 기술되지 않고 넘어간다. flag가 1로 세팅되는 경우 Variable name 의 길이를 8비트로 기술한다. 그리고 나서 Variable name 을 하나씩 영문자로 기록한다. 각 영문자는 아스키 코드(Ascii code)로 표현될 수 있다.

- <495> Operator는 표현식이 연산을 나타내는 경우 연산자를 나타낸다. flag가 0으로 세팅되는 경우 Operator 가 기술되지 않고 넘어간다. flag가 1로 세팅되는 경우 Operator 의 길이를 8비트로 기술한다. 그리고 나서 Operator를 하나씩 영문자로 기록한다. 각 영문자는 아스키 코드(Ascii code)로 표현될 수 있다.
- <496> Child ET index는 표현식이 다른 표현식을 포함하게 되는 경우 하위 표현식에 대한 기술을 할 수 있다. flag가 0으로 세팅되는 경우 하위 표현식에 대한 ET 인덱스가 기술되지 않고 넘어간다. flag가 1로 세팅되는 경우 하위 표현식을 나타내는 ET 인덱스를 8비트로 기록한다.
- <497> Args ET index는 표현식이 인자를 포함하는 경우 인자에 대한 기술을 ET 인덱스로 나타낼 수 있다. flag가 0으로 세팅되는 경우 인자를 표현하기 위한 ET 인덱스가 기술되지 않고 넘어간다. flag가 1로 세팅되는 경우 인자를 표현하기 위한 ET 인덱스를 8비트로 기록한다.
- <498> ET 테이블은 다음과 같은 XML 텍스트형식으로 표현된다.
- <499> {
- <500> (Kind, [Literal-Kind], [value], [name], [Op], [Child-Expr(ET No.)] [Args-Expr(ET No.)]),
- <501> ...
- <502> }
- <503> XML 형식을 지원하는 MPEG-4 SP에 대한 ET 테이블의 실시예는 다음과 같다.
- <504> ET
- <505> {
- <506> //0
- <507> ("Literal", "Integer", "12", -, -, -, -),
- <508> ("Literal", "Integer", "10", -, -, -, -),
- <509> ("Literal", "Integer", "20", -, -, -, -),
- <510> ("Literal", "Integer", "9", -, -, -, -),
- <511> ("Literal", "Integer", "8", -, -, -, -),
- <512> //5
- <513> ("Literal", "Integer", "13", -, -, -, -),
- <514> ("Literal", "Integer", "2048", -, -, -, -),
- <515> ("Literal", "Integer", "1024", -, -, -, -),
- <516> ("Literal", "Integer", "512", -, -, -, -),
- <517> ("Literal", "Integer", "31", -, -, -, -),
- <518> //10
- <519> ("Literal", "Integer", "32", -, -, -, -),
- <520> ("Literal", "Integer", "448", -, -, -, -),
- <521> ("Literal", "Integer", "6", -, -, -, -),
- <522> ("Literal", "Integer", "1", -, -, -, -),
- <523> ("Literal", "Integer", "2", -, -, -, -),
- <524> //15
- <525> ("Literal", "Integer", "4", -, -, -, -),

```

<526>      ("Var", -, -, "MAXW_IN_MB", -, -, -),
<527>      ("Var", -, -, "SAMPLE_COUNT_SZ", -, -, -),
<528>      ("Var", -, -, "SAMPLE_SZ", -, -, -),
<529>      ("Var", -, -, "MB_COORD_SZ", -, -, -),
<530>      //20
<531>      ("Var", -, -, "BTYPESZ", -, -, -),
<532>      ("Var", -, -, "NEWVOP", -, -, -),
<533>      ("Var", -, -, "INTRA", -, -, -),
<534>      ("Var", -, -, "INTER", -, -, -),
<535>      ("Var", -, -, "QUANT_MASK", -, -, -),
<536>      //25
<537>      ("Var", -, -, "ROUND_TYPE", -, -, -),
<538>      ("Var", -, -, "FCODE_MASK", -, -, -),
<539>      ("Var", -, -, "FCODE_SHIFT", -, -, -),
<540>      ("Var", -, -, "ACPREL", -, -, -),
<541>      ("Var", -, -, "ACCODED", -, -, -),
<542>      //30
<543>      ("Var", -, -, "MOTION", -, -, -),
<544>      ("Var", -, -, "FOURMV", -, -, -),
<545>      ("Var", -, -, "MV_SZ", -, -, -),
<546>      ("Literal", "Integer", "3", -, -, -, -),
<547>      ("Var", -, -, "QUANT_SZ", -, -, -),
<548>      //35
<549>      ("Var", -, -, "MAXH_IN_MB", -, -, -),
<550>      ("Var", -, -, "PIX_SZ", -, -, -),
<551>      ("Literal", "Integer", "...", -, -, -, -),
<552>      ("Var", -, -, "FLAG_SZ", -, -, -),
<553>      ("Var", -, -, "ADDR_SZ", -, -, -),
<554>      //40
<555>      ("Var", -, -, "BUF_SZ", -, -, -),
<556>      ("Var", -, -, "SEARCHWIN_IN_MB", -, -, -)
<557>      }
<558>      유형 테이블(Type Table, TT)은 Type Name, Entry Name, Expr index 및 Type index 필드를 포함한다.
<559>      이하, 선택스 파싱 테이블들에 대하여 설명한다.
<560>      CSCIT는 파싱 기능부가 SET 및 SRT를 이용한 프로세스의 결과 정보인 엘리먼트 정보(예를 들어, CSCI)에 대한
      상세 정보가 기술된 것이다. 즉, CSCIT는 종래 비트스트림으로부터 처리되어 CSCI 저장부에 저장되고, 디코딩

```

기능부들에 의해 이용될 모든 의미있는 자료(즉, 엘리먼트 정보)들에 대한 정보를 가진다. CSCIT는 해당 엘리먼트 정보의 고유 번호로서 구분자인 인덱스(C), 플래그(flag), 해당 엘리먼트 정보의 이름(Element Name), 해당 엘리먼트 정보의 자료 구조적인 특성을 지정하기 위한 속성(예를 들어, 해당 엘리먼트 정보의 저장 공간 크기, 해당 엘리먼트 정보가 배열형(Array)인지 여부 등), 해당 엘리먼트 정보가 선택스 파싱 과정에서만 이용되는지 또는 전체적인 디코딩 과정에서 이용되는지를 나타내는 Global/Local 등을 포함한다. CSCIT는 textual description이나 binary description(비트 변환된 바이너리 코드 형태) 등의 기술 방식으로 기술될 수 있을 뿐 아니라, 상기 부분 디코더 디스크립션 중 필요한 최소한의 데이터가 유사 스크립트 언어로 기술될 수도 있다.

<561> SET는 입력된 종래 비트스트림의 선택스(syntax)들에 대한 정보에 의해 구성된 디코더 디스크립션이다. SET는 각 선택스에 대한 인덱스(index), 엘리먼트 명칭(Element Name), 파라미터(Paramter) 및 SET-프로세스(process by SET-PROC) 정보를 포함한다. 여기서 인덱스는 SRT에서 사용되는 각 선택스를 구분하는 구분자(S)이다. 엘리먼트 명칭은 선택스의 이름으로, 선택스의 의미나 역할에 따라 명명될 수 있다. 파라미터는 SRT에 기술된 파싱 과정이 SET의 파싱 알고리즘을 호출할 때 인자 값으로 제공되며, 이 인자 값은 고정된 상수나 비트스트림으로부터 계산된 값, 또는 파싱 과정에서 도출된 자료를 저장하기 위한 버퍼 공간(예를 들어, CSCI 메모리)의 식별자일 수 있다. 파라미터를 통해 전달된 인자 값은 인덱스(P)를 통해 구분되며, 각 인자 값은 엘리먼트 정보(즉, CSCI 정보(C))로서, 획득한 데이터를 저장할 때 사용될 수 있으며, 추후 해당 엘리먼트 정보가 처리 과정에서 데이터로서 필요한 경우 다시 파라미터 인덱스(P)를 이용하여 해당 엘리먼트 정보가 리드(read)될 수 있다. 또한, 파라미터는 SRT에서 지정한 바에 따라 CSCI 정보(C) 뿐 아니라 상수 값을 나타낼 수도 있다. SET-프로세스는 각 비트스트림 선택스를 입력 받아 어떤 가공 절차를 거쳐 출력 데이터인 엘리먼트 정보를 생성할 것인지를 과정을 기술한다.

<562> SET는 textual description이나 binary description(비트 변환된 바이너리 코드 형태) 등의 기술 방식으로 기술될 수 있을 뿐 아니라, 상기 부분 디코더 디스크립션 중 필요한 최소한의 데이터가 유사 스크립트 언어로 기술될 수도 있다.

<563> 다음으로, SRT는 종래 비트스트림 내의 각 선택스간의 연결 정보를 나타낸 것이다. 즉, SRT는 각 선택스를 호출하고 다음 선택스로 이동하도록 지시하는 정보를 가진다. 파싱 기능부는 SRT를 이용하여 종래 비트스트림을 읽어들이거나 엘리먼트 정보가 CSCI 저장부에 저장 및/또는 갱신되는 순서를 규정한다.

<564> SRT는 인덱스(R), 파라미터(P), 규칙 정보를 포함한다.

<565> 인덱스(R)은 각 연결 정보(Rule)를 구분하도록 한다. 선택스의 인덱스(S)는 특정 연결 인덱스에서 처리할 선택스를 지정하므로, 파싱 기능부(또는 syntax 파싱을 수행하는 기능부들)는 SET를 이용하여 해당 선택스에 대해 지정된 프로세스를 수행한다. 파라미터는 SRT에 기술된 파싱 과정이 계층적으로 하위 단계에 또 다른 SRT의 파싱 알고리즘을 호출할 때 인자 값으로 제공되며, 이 인자 값은 고정된 상수나 비트스트림으로부터 계산된 값, 또는 파싱 과정에서 도출된 자료를 저장하기 위한 버퍼 공간(예를 들어, CSCI 메모리)의 식별자일 수 있다. 파라미터를 통해 전달된 인자 값은 인덱스(P)를 통해 구분되며, 각 인자 값은 획득한 데이터를 저장할 때 사용될 수 있으며, 추후 해당 엘리먼트 정보가 처리 과정에서 데이터로서 필요한 경우 다시 파라미터 인덱스(P)를 이용하여 해당 엘리먼트 정보를 리드(read)될 수 있다. 또한 파라미터는 SRT에서 지정한 바에 따라 CSCI 정보(C) 뿐 아니라 상수 값을 나타낼 수도 있다. 규칙 정보는 선택스 파싱이 처리되는 과정을 기술한 것으로, 분기와 반복 등 제어 구문을 사용할 수 있으며 하위 계층에 또다른 SRT나 SET의 엘리먼트를 호출하여 선택스 파싱을 처리할 수 있다.

<566> 입력 데이터는 해당 연결 인덱스에서의 연결 제어를 위한 조건 판단에 사용될 엘리먼트 정보의 목록을 나타낸다.

<567> 분기의 수는 후속하는 선택스에 연결되어질 수 있는 경우의 수로서, 해당 연결 인덱스에서 가지는 분기 경로의 총 수를 나타낸다. 분기 정보는 분기의 수 만큼 필요한 분기 정보가 존재(#1, #2, #3... 등)하며, 다음에 어떤 연결 인덱스를 처리할 것인지를 결정하도록 하는 조건 판단 알고리즘이다. 분기 정보에 의해 어떤 순서에 따라 어떤 내용을 읽어 들일지가 직접적으로 판단될 수 있다. 분기의 수가 1인 경우에는 입력 데이터가 존재하지 않으며, 분기 정보에 지정된 연결 인덱스를 처리하기 위해 즉시 진행한다. 그러나, 분기의 수가 2 이상인 경우에는 조건 판단이 수행(조건문 이후에는 다음 번 연결 정보(R)로 구성됨)되고 상응하는 연결 인덱스를 처리하기 위해 진행한다.

<568> 파싱 기능부는 해당 연결 인덱스에서 정의한 선택스를 처리하여 CSCI 저장부를 갱신한 후, 갱신된 CSCI 저장부

(532)의 엘리먼트 정보를 참조하여 읽어들이는 후 분기 조건 판단에 활용한다. 이는 예를 들어, 인덱스 R0의 분기 정보의 분기 조건인 'C0==1'에서의 C0는 선택스 S0를 처리한 후의 엘리먼트 정보 C0이다.

<569> SRT는 textual description이나 binary description(비트 변환된 바이너리 코드 형태) 등의 기술 방식으로 기술될 수 있을 뿐 아니라, 상기 부분 디코더 디스크립션 중 필요한 최소한의 데이터가 유사 스크립트 언어로 기술될 수도 있다.

<570> 마지막으로, DVT는 각 부호화기/복호화기에서 사용하는 허프만 테이블(Huffman table) 정보가 기록된 디코더 디스크립션이다. MPEG-1/2/4/AVC에서는 각 부호화 시 엔트로피 코딩(entropy coding)을 수행한다. 이 때 주로 허프만 코딩(Huffman coding) 방법이 이용되며, 이 경우 이용되는 정보가 허프만 테이블(Huffman table)이다. 통합 코덱을 구현하기 위해서는 각 복호 시 해당 복호화기에서 사용될 허프만 테이블(Huffman table) 정보가 제공되어야 한다. 따라서, 본 발명에 따른 디코딩 디스크립션 내에 신택스 파싱시 각 신택스(syntax)에 해당하는 허프만 테이블(Huffman table) 정보를 포함한다. 물론, 각 표준에 상응하는 허프만 테이블 정보가 이미 디스크립션 저장부에 기록되어 있는 경우 DVT의 전송은 생략되거나 코덱 번호(Codec #, 1120)와 프로파일 및 레벨 번호(Profile and level #, 1130)만이 포함될 수도 있을 것이다.

<571> DVT는 각 허프만 테이블에 대한 이름(name), 허프만 코딩에 의해 압축되어 출력되는 실제 값(value) 및 압축된 실제 값이 종래 비트스트림에 저장될 때 사용되는 코드 값(code)을 포함한다. 예를 들어, MCBPC 값을 압축하여 3이란 실제 값(value)을 얻었다면, 허프만 테이블 매핑(Huffman table mapping) 작업(예를 들어, SET의 PROCESS 부분)에 의해 종래 비트스트림(316)에는 코드 값(code) 011이 기록된다. 다른 예로서, 앞서 예시한 SET의 인덱스 S77의 Process 부분에는 VLD[1]이라 기록되어 있어 VLD라는 함수를 호출하게 된다. 이 함수에 의해 미리 지정된 길이(고정길이 또는 가변 길이)만큼 종래 비트스트림(316)을 읽어 코드 값(code)값을 얻은 후 허프만 테이블 매핑(Huffman table mapping) 작업에 의해 상응하는 실제 값(value)을 얻을 수 있다. 이 때 사용되는 Huffman table은 [1], 즉 1번째 테이블인 CBPY이다.

<572> DVT는 textual description이나 binary description(비트 변환된 바이너리 코드 형태) 등의 기술 방식으로 기술될 수 있을 뿐 아니라, 상기 디코더 디스크립션 중 필요한 최소한의 데이터가 유사 스크립트 언어로 기술될 수도 있다.

<573> 일 예로, DVT는 아래와 같이 textual description될 수 있다.

<574> DVT{((0,1), (1,001), (2,010), (3,011), (4,0001), (5,000001), (6,000010), (7,000011), (8,000000001), (9,NULL)) ((0,0011), (1,00101), (2,00100), (3,1001), (4,00011),(5,0111), (6,000010), (7,1011), (8,00010), (9,000011), (10,0101), (11,1010), (12,0100), (13,1000), (14,0110), (15,11), (16,000000), (17,000001), (18,NULL)) ((0,011), (1,11), (2,10), (3,010), (4,001), (5,0001), (6,00001), (7,000001), (8,0000001), (9,00000001), (10,000000001), (11,0000000001), (12,00000000001), (13,NULL)) ((0,11), (1,10), (2,01), (3,001), (4,0001), (5,00001), (6,000001), (7,0000001), (8,00000001), (9,000000001), (10,0000000001), (11,00000000001), (12,000000000001), (13,NULL))...

<575> 다른 예로서, DVT는 아래와 같이 binary description될 수 있다.

[illegible]

<577> 전술한 바와 같이, 본 발명에 따른 복호화기는 기본적으로 XML 기반, 즉 BSDL 기반 디코더 디스크립션을 이용하여 디코딩 프로세스가 처리되는 한편, 제2 실시예를 통하여 상술한 바와 같이, 바이너리 형태로 압축된 CDDL 기반 디코더 디스크립션을 입력받는 경우에도 이를 XML 기반(BSDL 기반) 디코더 디스크립션으로 변환하여 처리할 수 있다.

<578> BSDL에 기반한 DD는 높은 가독성을 갖고 있기 때문에, DD를 기반으로 다양한 구현(Implementation)을 시도하고자 하는 개발자에게 적합한 방식으로 간주된다. 그러나 XML 기반인 BSDL 서식이 자체적으로 압축 기능을 갖고 있지 않기 때문에, DD의 실시간 전송이 필요한 환경에서는 용량 관계 상 비효율적일 수 있다. 따라서, 전술한 본 발명의 제2 실시예에서와 같이 보다 적은 용량으로 동일한 처리 과정을 표현하는 CDDL 서식을 이용해 BSDL을

표현함으로써, 실시간 환경이 필요로 하는 압축 효과를 이끌어 낼 수 있다.

<579> CDDL과 BSDL 등 서로 다른 종류의 DD를 상호간의 방식으로 변환하기 위하여, 그 중간 형태 (Intermediate form)로서 비트스트림 구문 목록을 활용할 수 있다. 이 목록은 표준 규격 문서에서 제시하는 것과 동일, 또는 유사한 형태로 제공되는 목록으로, 영상 비트스트림이 규격에 따라 내장하게 되면 다양한 구문 정보들의 내역과 비트 길이를 나타낸다. 모든 종류의 DD 서식은 표준에 정의된 규격에 준수하여 작성되었음이 자명하기 때문에, 표준 규격 문서의 형태로 정보를 추출해 냄으로서 서로 다른 종류의 DD 규격 간 변환을 보다 손쉽게 할 수 있다.

<580> 도 18은 비트스트림 구문 목록을 활용한 CDDL/BSDL 간 상호 변환 관계를 나타낸다.

<581> CDDL로부터 구문 정보를 추출해 내기 위하여, 다음과 같은 간결한 규칙을 CDDL 기반 DD에 적용할 수 있다.

<582> 1. SRT의 구문 정보를 읽어들인다

<583> A. 하위 SRT 요소 호출 (Rn();) : 비트스트림 상에서의 하위 구문 요소 집합 호출로 정의한다.

<584> B. SET 요소 호출 (Sn();) : 각각의 개별 구문 요소로 판단한다.

<585> C. 분기 및 반복문 : 그 조건을 그대로 목록 상에 보존한다.

<586> 2. SET 요소를 판독하여 비트스트림 관련 구문을 식별한다

<587> A. READ 명령어: 고정된 비트 길이를 가진 구문 요소로 판단한다. 읽어들이는 비트의 길이는 해당 요소에 정의된 바, 또는 해당 요소를 호출하는 상위 SRT 명령어에서 정의한 바에 따른다. 만일 Byte-align 파라미터가 사용되었을 경우, 해당 구문을 start/end code로 간주한다.

<588> B. SEEK 명령어: When be followed by IF condition in parent SRT element, it will be considered as nextbits() function.

<589> C. 분기/반복문이 포함된 경우: 가변 길이를 가진 구문 요소로 판단한다.

<590> D. VLD (Huffman/Golomb...) 명령어가 쓰인 경우 : 가변 길이를 가진 구문 요소로 판단한다.

<591> E. 그 밖에 다수의 명령어가 결합된 경우: 가변 길이를 가진 구문 요소로 판단한다.

<592> F. READ/SEEK/VLD 등 비트스트림을 읽는 명령어가 없는 경우: 해당 SET 요소는 비트스트림의 구성과 관계가 없는 것으로 간주되어, 처리 과정에서 무시된다.

<593> 3. 모든 CSCI 요소들은 비트스트림 상에서 출현하는 개별 구문 정보를 저장하기 위한 공간으로 간주한다.

<594> 상기 규칙을 적용하여 하기 표 4의CDDL 구문을 아래와 같이 XML 구문으로 변환할 수 있다.

<595> [표 4]

<596>

Element name	Mnemonic	Bits
do {		
CSCI_0	bslbf	32bit
CSCI_1	uimsbf	8bit
while (next_bits()==00 00 01 B2) {		
R3();		
}		
R1();		
} while (next_bits()==433) {		
CSCI_2	bslbf	32bit

<597> SET

<598> { ("READ P1 > P2;"),

<599> ("READ P1 B > P2;"),

<600> ("SEEK P1 > P2;"),

```

<601>         ("SEEK P1 B > P2;")
<602>     }
<603>     SRT
<604>     {("
<605>     do {
<606>         S1(32, C0);
<607>         S0(8, C1);
<608>         S3(32,V0);
<609>         while(V0==434){
<610>             R3();
<611>             S3(32,V0);
<612>         }
<613>         R1();
<614>         S3(32,V0);
<615>     }while(V0!=433);
<616>     S1(32, C2);
<617>     ")}

```

<618> 이제까지 본 발명에 따른 복호화 장치 및 비트스트림 복호화를 위한 구문 해석 방법을 설명함에 있어 MPEG-4를 중심으로 설명하였으나, MPEG-1, MPEG-2 및 이외의 동영상 부호화/복호화 표준에 아무런 제한없이 동일하게 적용할 수 있음은 당연하다.

<619> 또한, 각 디코더 디스크립션들에 포함되는 정보 역시 하나의 표준에 의한 디코딩 수행을 위한 기능부들의 연결 관계, 해당 기능부에 요구되는 처리 프로세스 등에 관한 정보만으로 기술되지 않고, 복수의 표준에 의한 디코딩 수행을 위한 정보로 기술될 수도 있음은 자명하다.

<620> 예를 들어, 확장 비트스트림에 포함된 인코딩된 비디오 데이터의 초기 복수의 프레임은 MPEG-2로 인코딩되고, 후속하는 복수의 프레임은 MPEG-4로 인코딩되며, 나머지 프레임은 MPEG-1으로 인코딩되었다고 가정하자. 이 경우, 인코딩된 비디오 데이터의 디코딩을 위해 디코더 디스크립션에 포함되는 디코더 디스크립션 정보들은 인코딩 방법을 달리하는 각 프레임들이 톨 박스(510)에 포함된 각 표준에 따른 기능부들이 유기적으로 결합되어 동작될 수 있도록 구현될 것임은 자명하다.

<621> 상기에서는 본 발명의 바람직한 실시예를 참조하여 설명하였지만, 해당 기술분야에서 통상의 지식을 가진 자라면 하기의 특허 청구의 범위에 기재된 본 발명의 사상 및 영역으로부터 벗어나지 않는 범위 내에서 본 발명을 다양하게 수정 및 변경시킬 수 있음을 이해할 수 있을 것이다.

도면의 간단한 설명

<622> 도 1은 일반적인 복호화기의 구성을 개략적으로 나타낸 도면.

<623> 도 2는 일반적인 부호화기의 구성을 개략적으로 나타낸 도면.

<624> 도 3는 본 발명의 일 실시예에 따른 부호화기의 블록 구성도.

<625> 도 4는 본 발명에 따른 복호화기의 제1 실시예 블록 구성도.

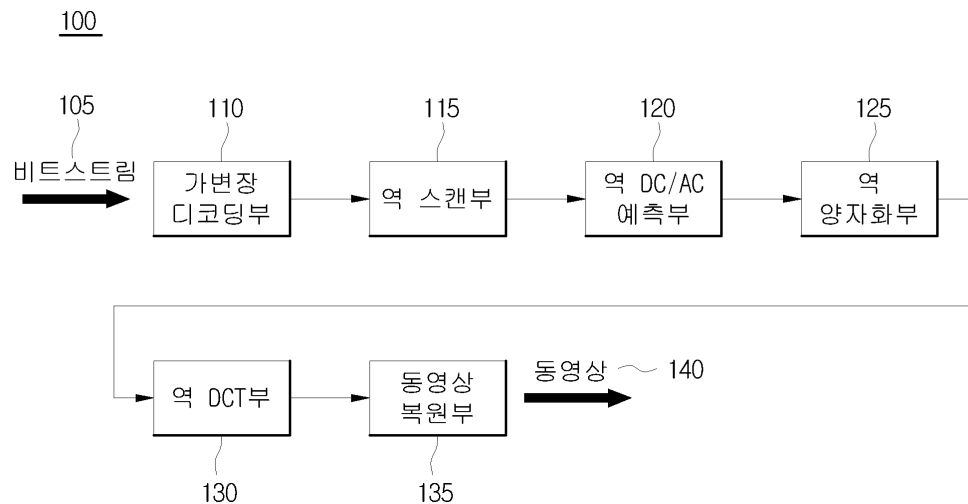
<626> 도 5는 본 발명의 일 실시예에 따른 디코딩 유닛에서 비트스트림 처리 과정을 구체적으로 나타낸 도면.

<627> 도 6은 본 발명의 바람직한 다른 실시예에 따른 디코더 디스크립션 입력 과정을 나타낸 도면.

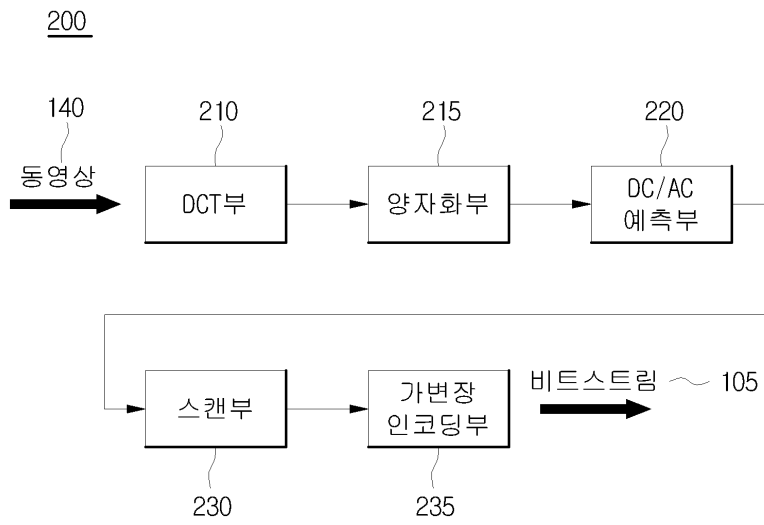
- <628> 도 7은 본 발명의 또 다른 실시예에 따른 디코더 디스크립션 입력 과정을 나타낸 도면.
- <629> 도 8은 본 발명의 다른 실시예에 따른 디코딩 유닛의 구성을 나타낸 도면.
- <630> 도 9은 본 발명의 다른 실시예에 따른 BSDL 파서의 구성을 나타낸 도면.
- <631> 도 10은 본 발명에 따른 복호화기의 제 2 실시예 블록 구성도,
- <632> 도 11은 도 10의 복호화부의 일실시예 구성을 개략적으로 나타낸 도면.
- <633> 도 12는 본 발명의 일실시예에 따른 FU 네트워킹 테이블의 구조를 나타낸 도면.
- <634> 도 13는 본 발명의 일실시예에 따른 VNT 테이블의 구조를 나타낸 도면.
- <635> 도 14는 본 발명의 일실시예에 따른 FUIT 테이블의 구조를 나타낸 도면.
- <636> 도 15는 본 발명의 일실시예에 따른 PT 테이블의 구조를 나타낸 도면.
- <637> 도 16은 본 발명의 일실시예에 따른 NCT 테이블의 구조를 나타낸 도면.
- <638> 도 17은 본 발명의 일실시예에 따른 ET 테이블의 구조를 나타낸 도면.
- <639> 도 18은 비트스트림 구문 목록을 활용한 CDDL/BSDL 간 상호 변환 관계를 나타내는 예시도

도면

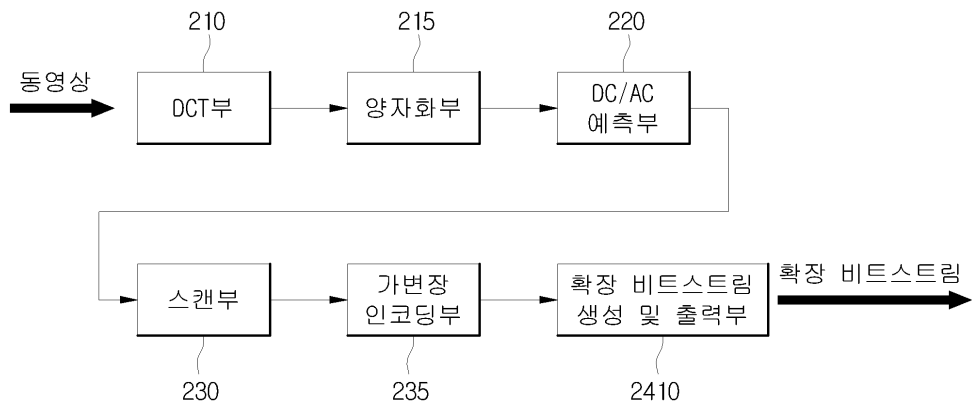
도면1



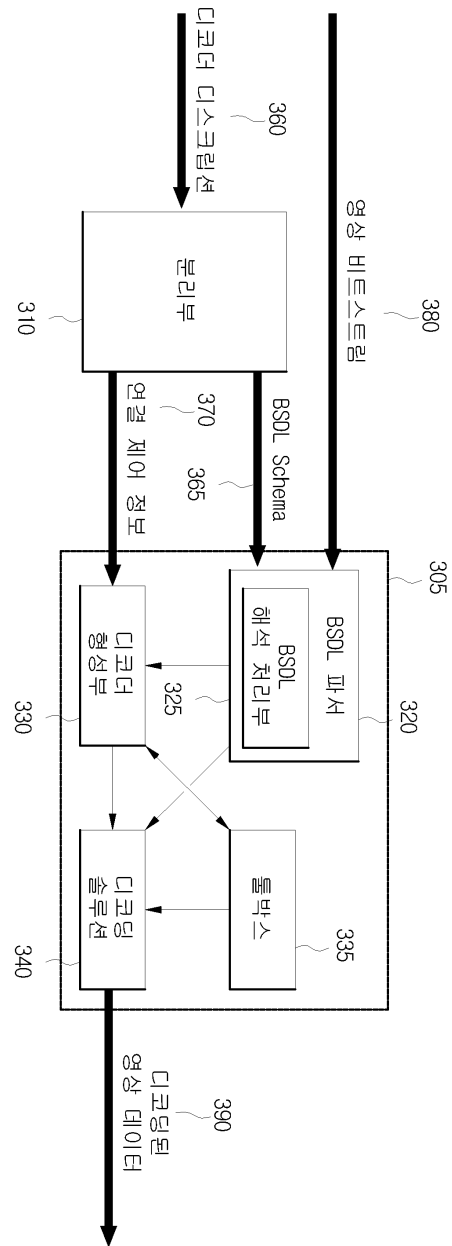
도면2



도면3

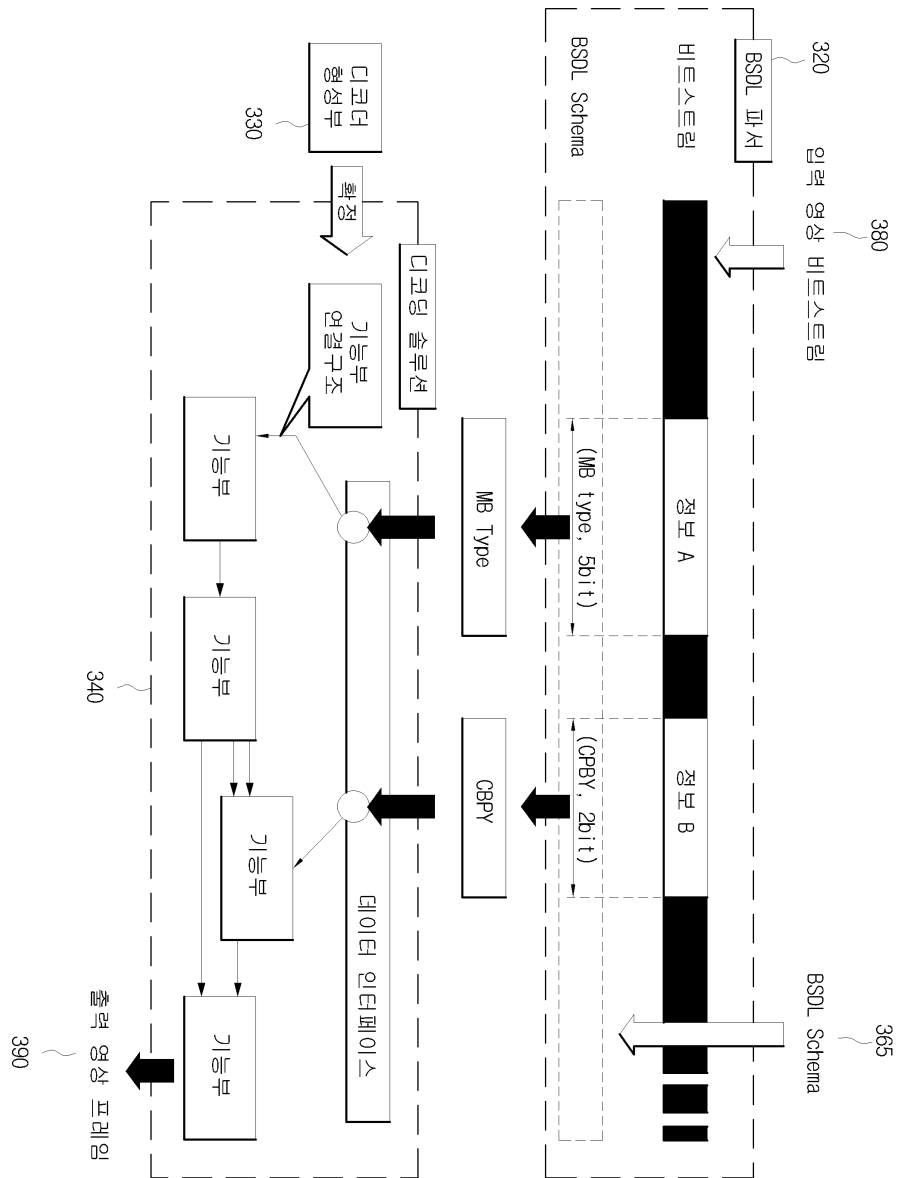


도면4

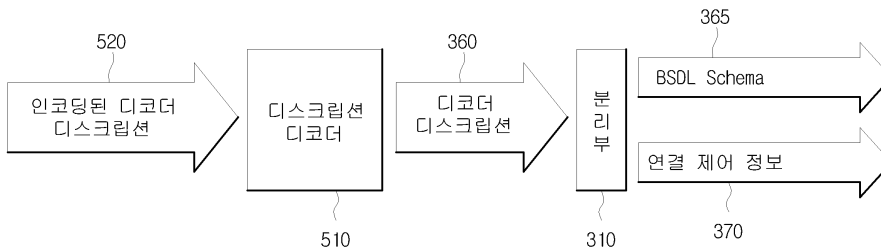


300

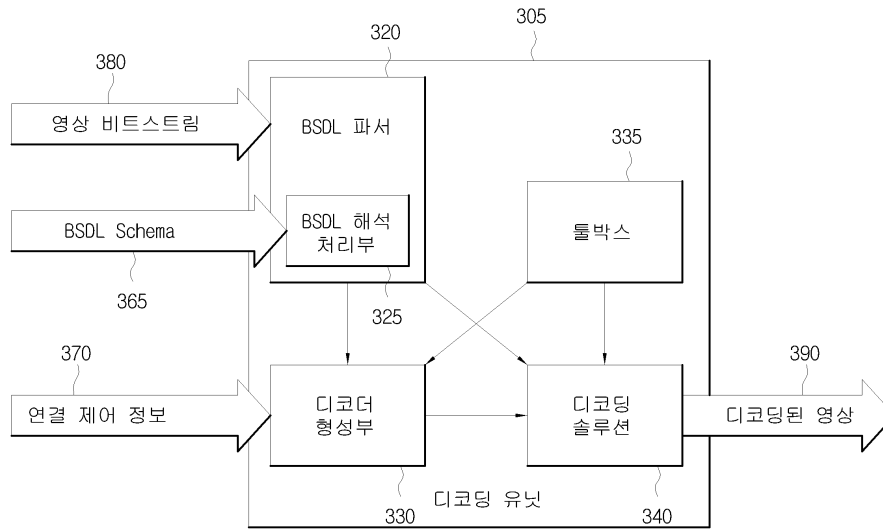
도면5



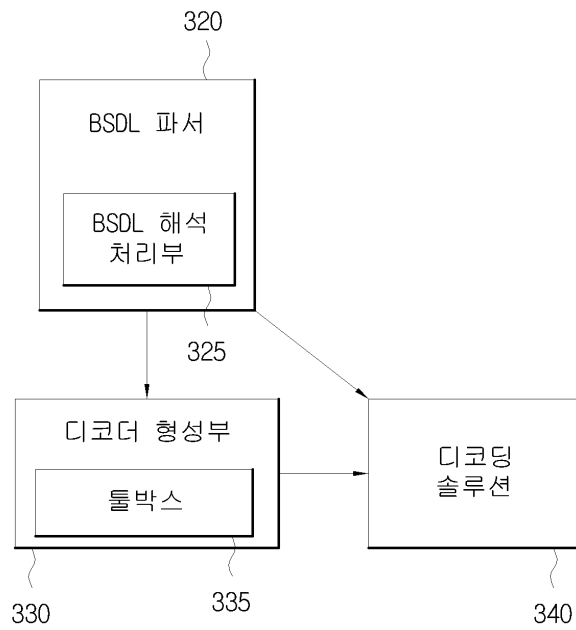
도면6



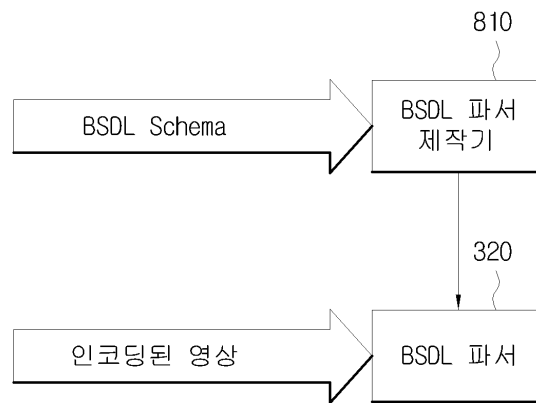
도면7



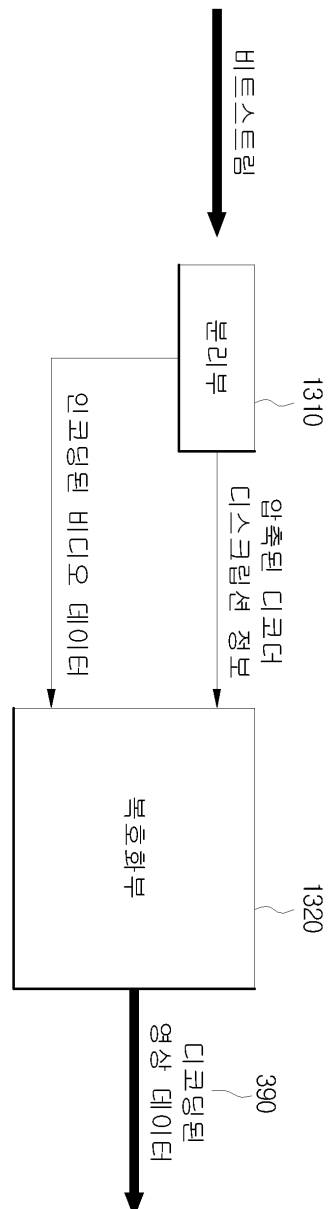
도면8



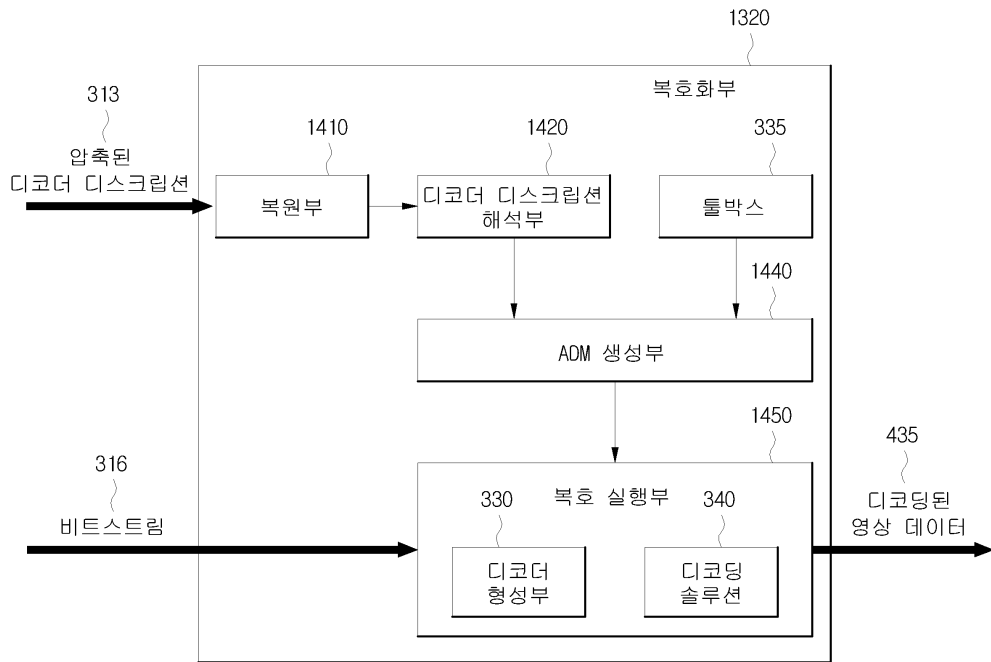
도면9



도면10



도면11



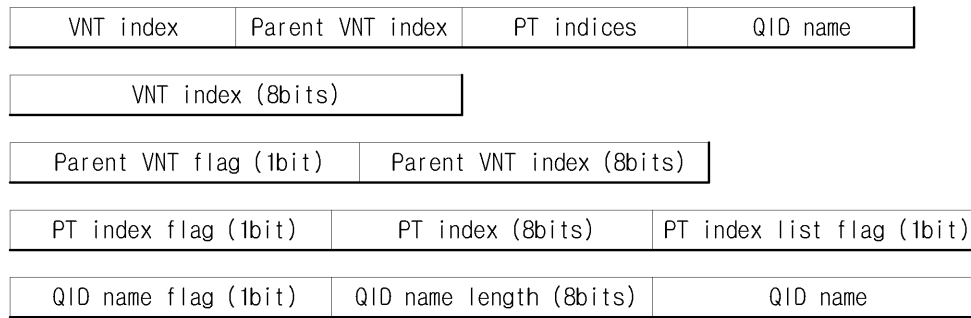
도면12

Table Start Code	Table Code	Content	Table End Code
------------------	------------	---------	----------------

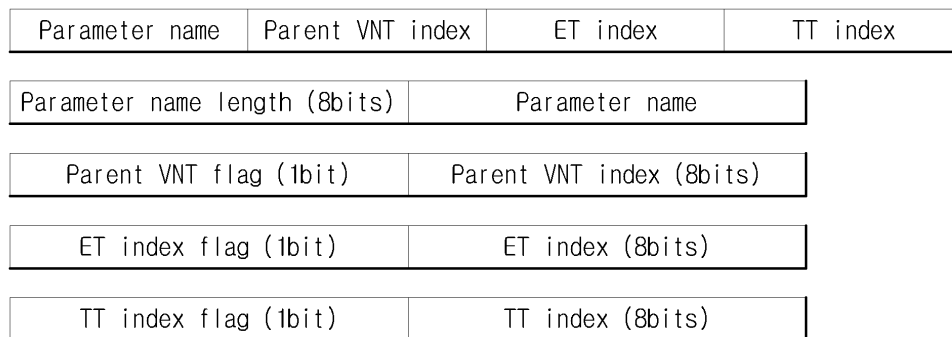
FUID	VN name	Version	INPUT ports	OUTPUT ports	QID name
FUID flag (1bit)		FUID (8bits)			
VN name length (8bits)		VN name			
INPUT port length (8bits)	INPUT port name		INPUT ports flag (1bit)		
OUTPUT port length (8bits)	OUTPUT port name		OUTPUT ports flag (1bit)		
QID name flag (1bit)	QID name length (8bits)		QID name		

도면13

도면14



도면15



Src FUIT index	Dst FUIT index	Src port name	Dst port name	
Attribute name	Attribute value	ET index		

Src FUIT flag (1bit)	Src FUIT index (8bits)			
----------------------	------------------------	--	--	--

Dst FUIT flag (1bit)	Dst FUIT index (8bits)			
----------------------	------------------------	--	--	--

Src and Dst same flag (1bit)	Src port name length (8bits)	Src port name	Dst port name length (8bits)	Dst port name
------------------------------	------------------------------	---------------	------------------------------	---------------

Attribute name flag (1bit)	Attribute name length (8bits)	Attribute name		
----------------------------	-------------------------------	----------------	--	--

Attribute value flag (1bit)	Attribute value length (8bits)	Attribute value		
-----------------------------	--------------------------------	-----------------	--	--

ET index flag (1bit)	ET index (8bits)			
----------------------	------------------	--	--	--

도면16

도면17

Kind	Literal kind	Literal value	Variable name	Operator	Child ET index	Args ET index
Kind (3bits)						
Literal kind flag (1bit)		Literal kind (3bits)				
Literal value flag (1bit)		Literal value length (8bits)	Literal value			
Variable name flag (1bit)		Variable name length (8bits)	Variable name			
Operator flag (1bit)		Operator length (8bits)	Operator			
Child ET index flag (1bit)		Child ET index (8bits)	Args ET index flag (1bit)	Args ET index (8bits)		

도면18

