고성능 허프만 코덱의 VLSI 구조

최현준* · 서영호** · 김동욱***

VLSI Architecture of High Performance Huffman Codec

Hyun-Jun Choi* · Young-Ho Seo** · Dong-Wook Kim***

이 논문은 2010년도 한국학술진흥재단 교육인적자원부 학술연구조성사업의 일환으로 연구되었음(KRF-2007-331-D00405).

요 약

본 논문에서는 비디오 코덱을 비롯한 멀티미디어 데이터 압축에 주로 이용되는 엔트로피 코딩 방식 중의 하나인 허프만 코딩을 위한 전용 하드웨어를 제안하고 구현하였다. 제안한 허프만 코덱은 허프만 인코더와 디코더로 구성되어 있다. 허프만 인코더는 룩업 테이블을 이용하여 심볼을 허프만 코드로 변환한다. 가변 길이의 허프만 코드는 데이터 패킷화 블록에서 32 비트의 일정한 형식으로 맞추어진 후에 프레임 단위로 직렬로 출력된다. 허프만 디코더는 직렬로 입력되는 비트스트림을 버퍼링 없이 트리 구조의 FSM을 이용하여 디코딩하여 심볼로 변환한다. 제안한 하드웨어는 동작의 유연성을 위해서 인코딩과 디코딩 하드웨어를 프로그래머블하게 동작시킬 수 있도록 하여 프로그래밍 과정을 통해서 다양한 허프만 코딩을 수행할 수 있도록 하였다. 구현한 하드웨어는 Altera사의 Cyclone III FPGA를 이용하여 검증하였고, 3725개의 LUT를 사용하면서 최대 365MHz로 동작이 가능하였다.

ABSTRACT

In this paper, we proposed and implemented a dedicated hardware for Huffman coding which is a method of entropy coding to use compressing multimedia data with video coding. The proposed Huffman codec consists Huffman encoder and decoder. The Huffman encoder converts symbols to Huffman codes using look-up table. The Huffman code which has a variable length is packetized to a data format with 32 bits in data packeting block and then sequentially output in unit of a frame. The Huffman decoder converts serial bitstream to original symbols without buffering using FSM(finite state machine) which has a tree structure. The proposed hardware has a flexible operational property to program encoding and decoding hardware, so it can operate various Huffman coding. The implemented hardware was implemented in Cyclone III FPGA of Altera Inc., and it uses 3725 LUTs in the operational frequency of 365MHz

키워드

허프만 코드, 인코더, 디코더, FPGA, VLSI, 비디오 코딩, 코덱

Key word

huffman code, encoder, FPGA, VLSI, video coding, codec

* 정회원 : 안양대학교
** 종신회원 : 광운대학교 (교신저자,yhseo@kw.ac.kr)
*** 정회원 : 광운대학교

접수일자 : 2010. 08. 17
심사완료일자 : 2010. 11. 05

Ⅰ. 서 론

데이터 압축 방법에는 크게 손실압축과 무손실압축 두 가지로 나눌 수 있으며, 압축에 이용되는 성질 측면에서 엔트로피 기법과 대상 기반 기법의 두 가지로 나눌 수도 있다. 무손실압축이란 압축한 데이터를 복원했을 때 복원한 데이터가 압축전의 데이터와 완전히 일치하는 것을 말한다. 이 기법은 압축할 때 압축할 데이터에 어떤 변경이나 수정도 가하지 않는다. 따라서 멀티미디어 정보에서 정확성이 생명인 데이터의 압축에 사용되는 기법으로 비트 보존 압축기법이라고도 부른다. 무손실압축 알고리즘으로는 RLC(Run Length Coding) 및 VLC(Variable Length Coding)를 주로 사용하는데 최근 고속 영상 처리 응용에서는 VLC 기법인 허프만 부호화를 많이 사용하고 있다[1][2].

허프만 부호화는 무손실압축 부호화 방법이며, 데이터통신, 음성 압축, 비디오 또는 영상 압축에 두루 사용된다. 허프만 부호화 방법은 통계학적으로 발생 확률이 높은 심볼에 대해서는 짧은 비트를 할당하고 발생 확률이 적은 심볼에 대해서는 긴 비트를 할당하여 전체적으로 데이터 크기를 줄이는 방법이다[3][4]. 허프만 코덱에서 원데이터에 대한 압축데이터를 생성하거나, 압축데이터에 대한 원데이터를 복원하기 위해서 Look-up table이 요구되는데 기존 허프만 코덱에 사용되는 Look-up table 방식 중 ROM(Read Only Memory) 또는 PLA (Programmable Logic Array)의 구조를 사용하여 코덱을 구현할 수 있지만, 적용분야가 한정된다는 단점과 비교동작을 수행하기 위해서 Match Logic이 따로 필요하다는 단점을 가지고 있다[5]. 따라서 허프만 테이블의 재구성이 가능하게 하기 위해서는 ROM이나 PLA 방식이 아닌 CAM(Content Addressable Memory)을 사용하는 것이 바람직하다[6][7]. 그러나 기존의 CAM을 이용한 허프만 코덱의 Look-up table 방식에서는 읽기 및 쓰기동작과 매치동작이 분리되어 수행되지 않으므로 제어가 복잡해질 뿐만 아니라 읽기 및 쓰기동작시에 매치라인이 플로팅 상태가 되어 오동작을 유발할 수 있다는 단점을 가지고 있다[8]. 허프만 디코딩은 특성에 따라서 크게 트리-기반 방식[9]과 그룹-기반 방식[10]으로 구분된다.

본 논문에서는 LUT 방식으로 허프만 인코딩을 수행하고 트리-기반 방식으로 디코딩을 수행하는 하드웨어를 구현하고자 한다. 프로그래머블한 하드웨어를 구현하기 위해서 프로그래밍이 가능한 LUT 방식을 적용하였고, 마찬가지로 프로그래머블한 FSM 기반의 트리 구조의 디코더를 적용하였다.

본 논문은 다음과 같이 구성된다. 2장에서는 기본적인 허프만 코딩의 알고리즘을 간략히 설명하고, 3장에서는 제안한 하드웨어의 구조를 설명한다. 4장에서 구현한 결과와 시뮬레이션 결과를 보이고, 5장에서 결론을 맺는다.

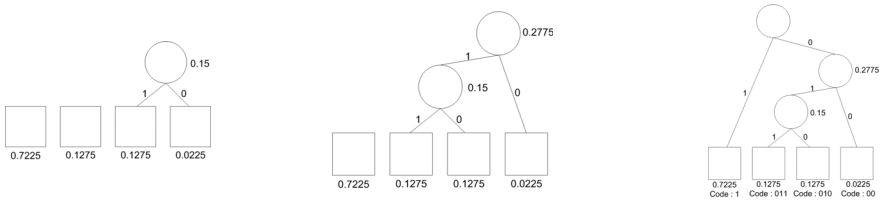
Ⅱ. Huffman coding

2.1 허프만 코딩 알고리즘

1952년에 A. D. Huffman이 무손실 압출을 수행하기 위해 이용되는 새로운 코드 구조를 개발했다[11]. 이 허프만 코딩은 무손실 압축에서의 모델링과 심볼-코드워드 맵핑 함수를 하나의 단일과정으로 처리한다. 입력 데이터는 실질적 모델링 과정을 위해 심벌의 순열로서 분할된다. 이처럼 대부분의 영상압축분야에서 데이터는 심벌로 구성된 알파벳의 크기를 제한하게 된다. 허프만의 코드생성과정은 다음과 같다.

① 각 기호에 대응하는 잎을 만들고, 발생확률을 기록한다.
② 발생확률이 가장 낮은 2개의 잎을 1개의 새로운 잎으로 결합하여 한쪽에는 “0”을 다른 한쪽에는 “1”을 할당하고, 두 발생확률의 합을 새로운 잎에 기록한다.
③ 잎이 1개일 때, 즉 발생확률의 합이 1이 될 때까지 ②를 반복한다.
④ 뿌리에서 각 기호의 잎으로 연결되는 가지에 붙어진 “0”과 “1”의 계열이 그 기호의 부호어가 된다.

허프만 코드는 평균 부호 길이 L을 최소로 하는 컴팩트 부로호서 발생 확률이 높은 기호에 대해서는 짧은 코드를, 발생 확률이 낮은 기호에 대해서는 긴 코드가 할당되는 방식이다[13]. 그러나 2원 정보원과 같이 기호의 수가 작을 때 각 기호를 단독으로 부호화하는 것으로는 효율 좋은 부호를 얻을 수 없다. 따라서 몇 개의 기호를 묶어 부호화 하면, 평균 부호 길이 L을 짧게 하여 효율을 높일 수 있다. 예를 들어, 발생 확률 P(A)=0.85, P(B)=0.15를



(a) (b) (c)

그림 1. 허프만 코드 할당 과정
Fig. 1. Huffman code generation procedure

가지는 정보원 기호 A와 B를 2원 허프만 인코딩하는 과정을 그림1에 나타냈다.

2.2 허프만 코딩의 하드웨어 구조

데이터의 통계학적 특성을 이용한 허프만 압축 부호화는 영상신호처리, 음성신호처리 및 데이터 저장 등의 다양한 응용에 비손실 압축기법으로 광범위하게 적용되고 있다. 허프만 코덱은 부호화 및 복호화에 허프만 트리를 사용하여 데이터를 처리하는 방법으로 고속의 영상처리 응용에서의 ASIC 구현에 사용되고 있다. 허프만 테이블은 엔코더에서 허프만 트리를 형성할 때 심볼에 부과되는 코드어를 저장하여두는 곳이다. 허프만 부호에 대한 코드 테이블은 고정테이블인 경우 ROM 또는 PLA를 사용하고, 부호의 내용이 가변적인 경우는 RAM이나 CAM을 사용할 수 있다[5][6][7][8].

ROM을 이용한 Look-up table의 구조는 기존 허프만 디코더의 Look-up table 구조는 비트 병렬 방식으로 디코딩을 하는 방법으로 ROM 기반 구조를 사용하고 있다. ROM 기반 구조는 State register, ROM Table, Buffer로 구성되어 있다. 기존 ROM 기반 구조는 간단하여 구현이 용이하다는 장점은 있으나, 피드백 루프에 의한 동작 속도가 느리다는 단점이 있어서 고속 코덱에는 응용이 불가능하다는 문제점이 있다. 그리고 코드어가 가변적인 경우에는 사용이 불가능하다[6].

PLA를 이용한 Look-up table 구조는 ROM 대신에 PLA를 사용하여 구성한다. PLA 기반 구조는 ROM 기반 구조의 단점인 속도문제를 개선할 수 있는 구조로서 약 200Mbit/s 정도의 속도가 가능하다. 하지만 고정구조만을 사용하므로 코드 테이블이 변하는 형태에는 적용이 불가능하다는 단점을 가지고 있다. 즉 허프만 테이블의 재구성이 어렵다는 것을 의미한다. 이러한 단점 때문에 응용분야에 따라 데이터의 통계적 분포에 맞추어 코드 테이블이 재구성될 필요가 있는 부분에는 사용이 불가능하기 때문에 기존의 ROM 이나 PLA 구조는 한계성을 가지고 있다. 따라서 코드 테이블을 재구성하기 위해서는 가변이 용이한 CAM을 사용하는 것이 바람직하다[7].

CAM을 이용한 Look-up table의 구조는 CAM 셀에서 매치신호가 "High"가 되면 RAM 셀의 해당 워드라인을 "High"로 만들어서 RAM에 저장되어있는 데이터를 출력하는 구조로 되어있다.[8,9] 기존 CAM 셀은 6개의 트랜지스터를 사용하는 SRAM과 데이터 매치동작을 위한 4개의 트랜지스터로 구성된 Exclusive-NOR 구조로 되어 있고 읽기나 저장을 할 때의 동작원리는 기존의 RAM과 같으며, 매치동작인 경우 매치선은 3.3[V]로 프리차지되고, 메모리 셀에 "High"가 저장되어 있을 때 bit 선으로 "High"가 입력되면 회로 아래 부분의 4개의 트랜지스터가 형성하는 두 개의 경로는 끊어져서 매치선은 "High"로 유지된다. 만약, 메모리 셀의 내용과 비트선의 내용이 다르면 두 개의 경로 중 하나가 접지로 연결되어 매치선은 "Low"로 된다. 따라서 기존 CAM 셀의 경우 읽기 및 쓰기동작과 매치동작이 분리되어 수행되지 않으므로 제어가 복잡해질 뿐만 아니라 읽기 및 쓰기동작시에 매치라인이 플로팅 상태가 되어 오동작을 유발할 수 있다는 단점을 가지고 있다[6][7][8].

트리-기반 방식은 허프만 코드 생성 트리의 역순으로 비트를 추적하면서 디코딩을 수행한다. 그룹-기반 방식은 고정된 허프만 코드를 사용할 경우 고속처리가 가능하다. 트리-기반 방식은 디코딩 시간이 가변인 단점이 있지만 일반적으로 전체 영상복원 시간에 비해서 허프만 디코딩이 매우 작은 시간동안 이루어지기 때문에 전체적인 동작 시간에는 영향을 주지 않는다.

Ⅲ. 제안한 하드웨어 구조

본 장에서는 허프만 인코딩과 디코딩을 위한 제안한 하드웨어의 구조에 대해서 설명한다.

3.1 허프만 코덱의 전체 구조

허프만 코덱은 허프만 인코더와 허프만 디코더로 구성된다. 제안한 허프만 코덱의 가장 큰 특징은 인코더와 디코더가 프로그래머블하다는 것이다. 허프만 인코더는 입력 인터페이스(Input Interface, II)와 DP-RAM(DR), 데이터 패킷화기(Data Packetizer, DP), 출력 FIFO(Output FiFO, OF), 그리고 비트스트림 생성기(Bitstream Generator, BG)로 구성된다. 허프만 디코더는 FSM 구성기(FSM Configurator, FC), 프로그래머블 FSM(Programmable FSM, PF), 그리고 출력 인터페이스(Output Interface, OI)로 구성된다.

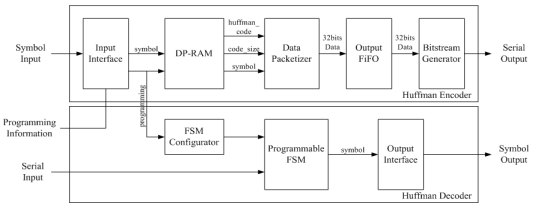


그림 2. 허프만 코덱의 구조
Fig. 2. Architecture of Huffman codec

먼저 허프만 인코더의 구조에 대해서 설명한다. 입력 인터페이스는 입력된 심볼을 버퍼링하는 역할을 담당한다. DP-RAM은 허프만 인코딩을 담당하는 블록으로 입력된 심볼을 허프만 코드(huffman_code)와 허프만 코드의 길이에 대한 정보(code_size)로 변환하는 역할을 담당한다. 또한 예외 조건인 경우에 심볼을 그대로 전송하는 역할도 담당한다. DP-RAM은 제어기와 LUT들로 구성되고, LUT는 SRAM을 사용하여 프로그램이 가능하도록 하였다. 데이터 패킷화기는 허프만 인코더에서 가장 복잡하면서 중요한 기능을 담당하는데 DP-RAM으로부터 출력된 정보들은 각 클록마다 가변적인 특성을 갖는다. 따라서 이러한 가변적인 비트들을 일정한 길이(32비트)로 만들어서 출력하는 것은 매우 어려운 일인데 이 작업을 출력 FIFO가 담당한다. 비트스트림 생성기는 32비트 데이터를 프레임 단위로 모아둔 뒤에 직렬로 출력하는 역할을 수행한다.

다음으로 허프만 디코더의 구조에 대해서 설명한다. 허프만 인코더와 마찬가지로 허프만 디코더도 역시 프로그래머블한 구조를 갖는다. 먼저 초기 프로그래밍을 통해서 허프만 디코딩을 위한 FSM의 트리를 결정한다. 이러한 FSM의 구성은 FSM 구성기를 통해서 수행된다. 프로그래머블 FSM이 허프만 디코딩을 수행하는 블록이다. 앞서 설명한 것과 같이 트리-기반의 디코딩을 수행할 수 있는 다수 개의 FSM으로 구성되어 있고, 이 FSM들은 재구성이 가능한 구조를 갖는다. 그러나 FSM들이 갖고 있는 트리 이상의 허프만 코드에 대해서는 프로그래밍이 불가능한 한계는 갖는다. 따라서 제안한 구조를 이용하여 허프만 디코더를 설계할 경우에는 가장 긴 허프만 코드의 크기에 맞는 FSM을 반드시 내부에 한 개 이상 넣어두어야 한다. 프로그래머블 FSM을 통해서 검출된 심볼은 출력 인터페이스를 통해 버퍼링 된 후에 다음 블록으로 출력된다.

3.2 허프만 인코더

본 절에서는 앞서 설명한 허프만 인코더를 좀 더 자세히 설명하도록 한다. 먼저 입력된 심볼에 대해서 허프만 코드를 직접 할당하는 역할을 수행하는 DP-RAM의 구조를 살펴본다. DP-RAM은 다수 개의 LUT(look-up table)로 구성되고, LUT_select 신호에 의해서 입력된 심볼을 적절한 LUT를 통해 코드를 할당할 수 있다. 본 논문에서 구현한 하드웨어에는 6개의 LUT로 구성된다. 본 연구팀이 개발하고 있는 압축 분야에 활용하기 위해서 개발한 것으로 이에 대한 자세한 설명은 본 논문의 범위를 벗어나기 때문에 생략하도록 한다. LUT 내부에는 허프만 코드(huffman_code)와 허프만 코드의 길이를 나타내는 코드 크기(code_size)에 대한 정보가 들어있어서 심볼이 입력되면 심볼을 LUT의 주소로 사용하여 LUT 내부에 저장된 허프만 코드와 코드의 길이 정보를 출력한다. 그림 3에 DP-RAM의 간략한 구조를 나타냈다.

데이터 패킷화기는 다양한 길이의 허프만 코드값을 32비트의 일정 길이로 만드는 역할을 한다. 그림 4에서 보이는 것처럼 허프만 코드의 길이에 대한 정보는 비트정보 신호(code_size)를 통해 얻을 수 있고 이를 이용해서 첫 번째 다중 쉬프터(Multi-shifter1)에 허프만 코드를 순서대로 저장하는데, 예외 영역에 해당되는 계수가 발견 되었을 경우는 허프만 코드 외에 심볼(symbol)도 저장한다.

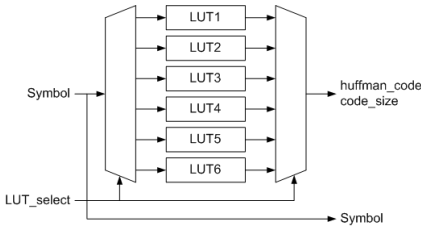


그림 3. DP-RAM의 구조
Fig. 3. Architecture of DP-RAM

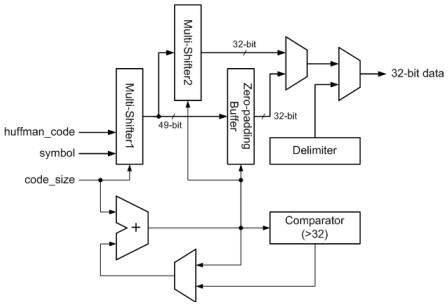


그림 4. Data Packetizer의 구조
Fig. 4. Architecture of Data Packetizer

이와 동시에 비트 정보는 누적되면서 비교기(Comparator)에 의해 32비트 출력이 가능한지 판단한다. 32비트 이상의 허프만 코드가 확인되면 두 번째 다중 쉬프터(Multi-shifter2)에 보내진 후 재 정렬되어 32비트 단위로 출력된다. 전체적인 동작은 부대역별로 이루어지고 32비트에서 부족한 데이터는 영값으로 채워 32비트로 만든다. 영상의 시작과 끝, 영역에 대한 정보, 그리고 필드 및 프레임 정보 등은 구획기(Delimiter)에 저장되어 압축결과와 함께 복원을 위한 정보로 부가된다.

3.3 허프만 디코더

허프만 디코더는 그림 5에 보이는 것과 같이 허프만 디코딩 결과로 16비트의 심볼(Symbol[15:0])를 출력한다. 16비트 두 개의 입력 포트로부터 32비트 단위의 입력 데이터를 받고 코드경계 검출기(Delimiter detector)에 의해서 영역 정보를 비롯하여 영상에 대한 데이터를 추출하고 직렬 쉬프터(HD shifter)를 통해 직렬 데이터를 발생시켜 허프만 디코딩을 수행한다. 디코딩을 하면서 예외 영역에 대한 정보가 검출되면 그 후의 4개의 직렬 데이터를 심볼로 변환하여 출력한다.

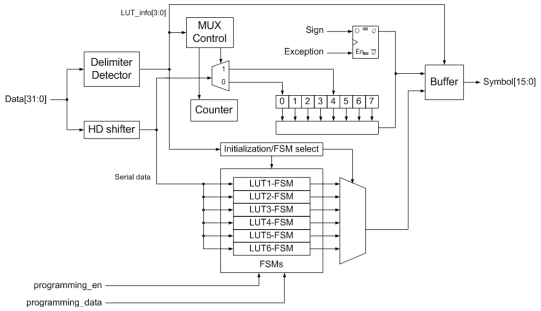


그림 5. Programmable FSM의 구조
Fig. 5. Architecture of Programmable FSM

허프만 디코더의 트리-기반의 FSM은 인코더와 마찬가지로 프로그래머블한 특성을 갖는다. 트리-기반의 FSM을 프로그래밍하는 방법을 아래에 나타냈다. 먼저 표 1에는 간단한 심볼들과 이 심볼들의 발생 확률에 따라서 선택된 허프만 코드의 예를 나타내고 있다. 총 8개의 심볼들이 존재하고 심볼들에 따라서 최소 1비트에서 최대 5비트의 크기를 갖는 허프만 코드가 존재한다.

표 1. 심볼과 허프만 코드의 예
Table 1. Example of symbol and Huffman code

심볼	허프만 코드
-3	0010
-2	0000
-1	010
0	1
1	011
2	0001
3	00110
4	00111

표 1과 같은 허프만 코드를 디코딩하기 위해서는 그림 6(a)와 같은 트리 구조의 FSM이 필요하다. 그림 6의 FSM은 간략히 나타내기 위해서 초기 상태로 재귀하는 상태 천이선을 나타내지 않았다.허프만 코드에 따라서 FSM 구성기에 의해 그림 6(a)의 기본 트리 구조는 그림 6(b)와 같이 경로가 선택된다. 이러한 경로만 따로 나태내면 그림 6(c)와 같다.

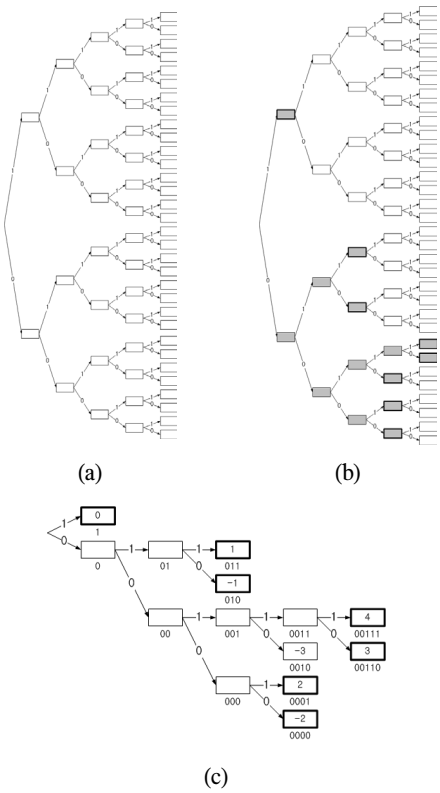


그림 6. Programmable FSM의 프로그래밍 방식
Fig. 6. Programming method of Programmable FSM

Ⅳ. 구현 결과

본 절에서는 제안한 허프만 코덱에 대한 구현결과와 중요한 동작들에 대한 시뮬레이션 결과를 보인다. 먼저 제안한 하드웨어는 Altera사의 Cyclone III EP3C5F256C6의 FPGA를 이용하여 구현하였고, 이 FPGA에서 3725개의 조합함수와 1080개의 레지스터를 사용하며 구현되었고, 내부 메모리는 109,123 비트를 사용하였다. 또한 365MHz의 고속 동작도 가능하였다.

그림 7에는 Altera의 Quartus II 10.0을 사용하여 FPGA 기반의 RTL 합성도를 보여주고 있다. 구현결과인 그림 7를 허프만 코덱의 블록도인 그림 2와 비교해보면 거의 동일한 것을 확인할 수 있고, 한 가지 차이점은 출력 FIFO에 데이터를 쓰고 읽기 위한 제어 블록이 하나 추가된 것이다.

다음으로 시뮬레이션 결과를 보이도록 한다. 허프만 인코더에서 가장 어려운 동작은 데이터 패킷화기이다. 따라서 허프만 인코더에서는 대표로 데이터 패킷화기에 대한 시뮬레이션 결과를 보인다. 그림 8에 데이터 패킷화기에 대한 시뮬레이션 결과 예를 보이고 있다. 임의로 000001111이라는 허프만 코드(hc[8..0])를 고정하여 입력시키고 코드 크기(bi[3..0])를 무작위로 증가 및 선택하여 출력되는 32비트 데이터의 출력값(prob_buff[31..0])을 관찰하였다. 출력값이 유효한 순간은 out_valid 신호가 1인 지점이고, ec_f는 예외 코드가 발생했다는 것을 의미한다. ec_f가 1이 되면 코드 크기에 따른 허프만 코드와 심볼(wc[8..0])를 모두 포함시켜서 데이터를 패킷화한다.

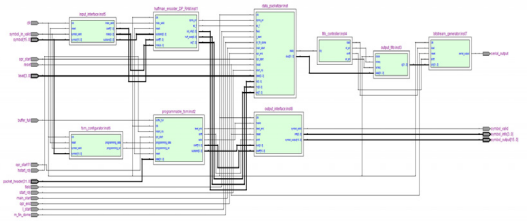


그림 7. 허프만 코덱의 FPGA기반 RTL 합성도
Fig. 7. FPGA-based RTL synthesis of Huffman Codec

허프만 디코딩에 대한 시뮬레이션 결과는 다음과 같다. DP-RAM의 첫 번째 LUT에 대한 내용은 표 2와 같다. 총 32개의 심볼이 존재하고 각 심볼에 대해 최소 1비트에서 최대 9비트의 허프만 코드가 할당되어 있다. 또한 표 1을 이용하여 프로그래밍된 프로그래머블 FSM에 내부의 첫 번째 FSM은 그림 9와 같은 트리 구조를 갖게 된다.

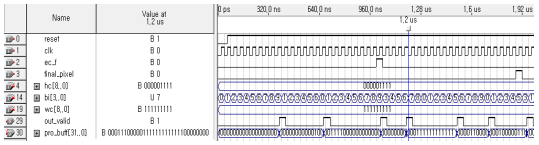


그림 8. Data Packetizer의 시뮬레이션 결과
Fig. 8. Simulation of Data Packetizer

표 2. LUT1의 심볼과 허프만 코드
Table 2. Symbol and huffman code of LUT1

심볼	허프만 코드	심볼	허프만 코드
-15	010010	1	000
-14	00100001	2	01101
-13	00100110	3	00111
-12	01000100	4	010101
-11	01001110	5	010000
-10	01011100	6	0101111
-9	0010001	7	0100110
-8	0011010	8	0011011
-7	0100011	9	0010010
-6	0101100	10	01011101
-5	001100	11	01000101
-4	010100	12	00100111
-3	00101	13	010011110
-2	01100	14	010011111
-1	0111	15	00100000
0	1	16	0101101

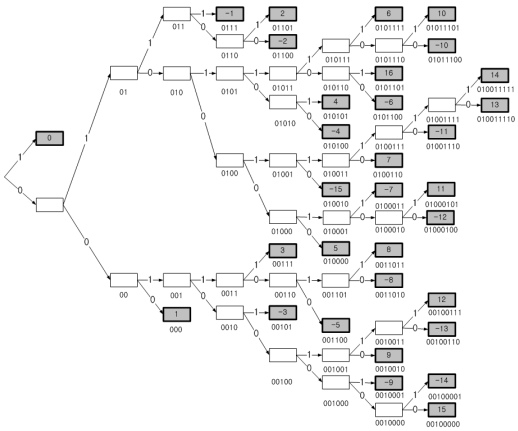


그림 9. 허프만 디코딩을 위한 LUT1의 트리 구조
Fig. 9. Tree architecture of LUT1 for huffman decoding

그림 10에 이러한 전체적인 디코딩 결과를 나타내었다. 1이 입력되면 허프만 디코더는 표 2에서 심볼 0를 출력한다. 그림 10에서 val 신호가 1인 순간에 p 신호의 값이 심볼에 해당한다. 다음으로 0111이 순차적으로 입력되면 -1이라는 심볼을 출력한다. 그리고 01000101이라는 값이 순차적으로 입력되면 11이라는 심볼을 출력한다. 표 2에서 심볼 11이 허프만 코드 01000101로 할당되어 있음을 확인할 수 있다. 그 후에 연속적인 1이 입력되어 허프만 디코더는 연속적인 심볼 0을 검출했음을 출력하고 있다.

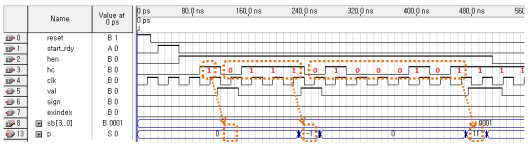


그림 10. 허프만 디코딩의 시뮬레이션 결과
Fig. 10. Simulation of huffman decoding

V. 결론

본 논문에서는 프로그래밍이 가능한 허프만 코덱을 구현하였다. 허프만 코덱은 허프만 인코더와 디코더로 구성이 되고 모두 프로그래밍이 가능하여 다양한 허프만 코드에 대해 적응적인 특성을 갖도록 하였다. 구현한 허프만 인코더는 입력 인터페이스와 DP-RAM, 데이터 패킷화기, 출력 FIFO, 그리고 비트스트림 생성기로 구성하였다. 그리고 허프만 디코더는 FSM 구성기, 프로그래머블 FSM, 그리고 출력 인터페이스로 구성하였다. 구현한 하드웨어는 Altera사의 Cyclone III FPGA를 이용하여 검증하였고, 3725개의 LUT를 사용하면서 최대 365MHz로 동작이 가능하여 고속 및 고성능 압축 분야에서 사용이 가능할 것으로 사료된다.

감사의 글

이 논문은 2010년도 한국학술진흥재단 교육인적자원부 학술연구조성사업의 일환으로 연구되었음 (KRF-2007-331-D00405).

참고문헌

[1] Khalid Sayood, “Introduction to Data Compression”, 2nd ed., Morgan Kaufmann, 2000.

[2] V. Bhaskaran and K. Konstantinides, “Image and Video Compression Standards, Boston, MA:Kluwer Academic Publishers, 1995.
[3] E. Linzer, “Super efficient decoding of color JPEG image on RISC machines”, Image Communication, Vol. 8, No. 1, pp. 13～24, Jan.1996.
[4] Hashemlan, “Memory efficient and high-speed search Huffman Coding”, IEEE Transactions on Communications, Vol. 43, No. 10, pp. 2576～2581, Oct. 1995.
[5] S. Chang and David Messerschmitt, “Designing High-Throughput VLC Decoder Part Ⅰ-Concurrent VLSI Architecture”, IEEE Trans. Circuits and Systems for Video Technology,Vol. 2, pp. 187～196, Jun. 1992.
[6] Genuhoe Kim et al, “Design of Variable Length Decoder based on CAM”, Proc. JTC-CSCC’94, pp. 950～954, 1994.
[7] E. Komoto et al, “A High-speed and Compact-Size JPEG Huffman Decoder using CAM”, Symp. VLSI ckt, pp. 37～38, 1993.
[8] A. G. Hanlon, “Content Addressable and Associative Memory System”, IEEE Trans. on Electronic Circuits, Vol. Ec 15, No. 4, pp. 509～521, Aug. 1996.
[9] Y. Oi, A. Taniguchi, and S. Demura, ″A 162Mbit/s variable length decoding circuit using an adaptive tree search technique,″ in Proc. IEEE 1994 Custom Integrated Circuits Conf., pp. 107-110, May. 1994.
[10] R. Hashemian, ″Design and hardware implementation of a memory efficient Huffman decoding,″ IEEE Trans. Consumer Electron., vol. 40, pp. 345-352, Aug. 1994.
[11] D. A Huffman, ″A method for the construction of minimum-redundancy codes″ Proc, IRE, vol. 40, pp.1098-1101, 1952.

최현준(Hyun-Jun Choi)

2003년 2월 : 광운대학교 전자재료공학과 졸업(공학사)
2005년 2월 : 광운대학교 일반대학원 졸업(공학석사)
2009년 2월 : 광운대학교 일반대학원 졸업(공학박사)
2010년 3월~현재 : 안양대 정보통신공학과 조교수
※관심분야 : 영상압축, 워터마킹, SoC

서영호(Young-Ho Seo)

1999년 2월 : 광운대학교 전자재료공학과 졸업(공학사)
2001년 2월 : 광운대학교 일반대학원 졸업(공학석사)
2004년 8월 : 광운대학교 일반대학원 졸업(공학박사)
2003년 6월~2004년 6월 : 한국전기연구원 연구원
2005년 9월~2008년 2월 : 한성대학교 조교수
2008년 3월~현재 : 광운대학교 교양학부 조교수
※관심분야 : 2D/3D 영상 및 비디오 처리, 디지털 홀로그램, SoC 설계, 워터마킹/암호화

김동욱(Dong-Wook Kim)

1983년 2월 : 한양대학교 전자공학과 졸업(공학사)
1985년 2월 : 한양대학교 대학원 졸업(공학석사)
1991년 9월 : Georgia 공과대학 전기공학과 졸업 (공학박사)
1992년 3월~현재 : 광운대학교 전자재료공학과 정교수
2009년 3월~현재 : 실감미디어연구소 소장
※관심분야 : 디지털 VLSI Testability, VLSI, CAD, DSP 설계, Wireless Communication