한국항행학회 논문지 제15권 제2호 2011년 4월

JBIG2 심벌 ID 부호화를 위한 런코드 부호기의 하드웨어 구현

Hardware Implementation of RUNCODE Encoder for JBIG2 Symbol ID Encoding

서석용*, 고형화**

Seok-Yong Seo*, Hyung-Hwa Ko**

요 약

본 논문은 팩시밀리를 위한 이진 영상 압축 표준인 JBIG2의 주요 구성모듈의 하나인 심벌 ID 코드 길이 부호화를 위한 런코드 부호기 IP를 하드웨어로 설계구현에 관한 것이다. VHDL코드 생성 및 하드웨어 합성을 위해서 ImpulseC Codeveloper와 Xilinx ISE/EDK 프로그램을 사용하였다. 합성된 하드웨어는 Xilinx사의 ML410 개발보드의 Virtex-4 FX60 FPGA에 다운로드하여 성능평가를 수행하였다. 합성된 하드웨어가 FPGA에서 차지하는 면적은 전체 slice의 13%를 차지하였다. 동작 검증을 위해 Active HDL 툴을 이용하여 각 IP에 대한 파형 검증을 수행한 결과 정상 동작함을 확인함으로써 하드웨어로의 구현에 적합성을 확인하였다. 아울러 ML410 개발보드 상에서 Microblaze CPU를 이용해 소프트웨어로만 수행한 경우와 동작 속도를 비교 한 결과, 구현된 하드웨어는 40배 이상의 빠른 처리 속도를 나타내었다. 구현된 하드웨어와 연동된 소프트웨어 모듈로 표준 CCITT문서를 압축한 결과 정상적으로 동작함을 확인하였다.

Abstract

In this paper, the RUNCODE encoder hardware IP was designed and implemented for symbol ID code length encoding, which is one of major modules of JBIG2 encoder for FAX. ImpulseC Codeveloper and Xilinx ISE/EDK program are used for the hardware generation and synthesis of VHDL code. The synthesized hardware was downloaded to Virtex-4 FX60 FPGA on ML410 development board. The synthesized hardware utilizes 13% of total slice of FPGA. Using Active-HDL tool, the hardware was verified showing normal operation. Compared with the software operating using Microblaze cpu on ML410 board, the synthesized hardware was better in operation time. The improvement ratio of operation time between the synthesized hardware and software showed about 40 times faster than software only operation. The synthesized H/W and S/W module cooperated to succeed in compressing the CCITT standard document.

Key words : JBIG2, RUNCODE, FPGA, IP/SoCA

I. 서 론

국내의 기상실황 및 예보 사항을 서비스하는 영역 기상방송은 한국연안은 물론 원양을 항해중인 선박,

* 광운대학교 전자통신공학과((Dept. of Electronics and Communication Eng., Kwang Woon University)
** 광운대학교 전자통신공학과((Dept. of Electronics and Communication Eng., Kwang Wwoon University)
· 제1저자 (First Author) : 서석용
· 투고일자 : 2011년 3월 29일
· 심사(수정)일자 : 2011년 3월 30일 (수정일자 : 2011년 4월 22일)
· 게재일자 : 2011년 4월 30일

태평양지역을 비행하는 항공기, 연안 여객선, 어선 및 선박회사 등 기상 정보를 필요로 하는 기관에서 조업, 항해, 운항 등 방재업무에 활용하도록 하는 중요한 무선통신수단이다. 우리나라는 1971년부터 김포 기상 통신소에서 방송하고 있으며, 현재 해상 기상특보, 태풍 정보 등 20종의 기상 정보를 하루 63회 제공한다. 바다에서 항해하거나 조업하는 선박들은 라디오 채널을 선택하듯 수신기 다이얼을 돌려 인근 국가의 영역기상방송에 주파수를 맞춰 들을 수도 있고, 기상 정보를 팩시밀리를 통해 받아볼 수 도 있다.

이러한 기상예보 무선통신에서 적용중인 팩시밀리 통신에 적용되고 있는 영상 압축 표준은 MH(Modified Huffman), MR(Modified READ)[1], MMR(Modified Modified READ)[2], JBIG[3]의 순서로 발전되어져 왔지만, 근래에 들어 더 높은 화질의 자료 송수신을 필요로 함에 따라 기존의 방법인 MMR보다는 3～5배, JBIG보다는 2～4배 우수한 압축 성능을 가지는 JBIG2 표준이 적용되고 있다[4] - [6]. 현재 JBIG2는 전자문서 형식으로 널리 사용되고 있는 PDF 형태의 파일에서 채택하고 있다. Adobe사의 Acrobat[7]에서 문서영상의 압축 알고리즘으로 사용되고 있으며, Lead Technology사의 LEADTOOS 프로그램[8], Verypdf사의 Image2- PDF 프로그램[9] 등 PDF 파일을 지원하는 프로그램이 그 응용 예이다. JBIG2는 PM&S[10]와 SPM[11]에 기반을 두어 표준화된 압축방식으로 주요 구성 블럭은 심벌사전, 텍스트 영역부호기, 심벌 ID부호기 등으로 구성된다. 각 부호기에 사용되는 압축기법은 허프만부호화, MQ 산술부호화 및 MMR부호화 등이다.

JBIG2의 중요한 프로파일인 FAX 프로파일의 구현을 위해서는 임베디드 시스템으로의 개발이 필수적인데, 이를 위하여서는 SoC/IP 형태로 설계함이 필수적이며, S/W와 H/W의 연동 설계 기술의 개발이 시급하다. 본 논문에서 적용한 연동 설계 기법은 하드웨어와 소프트웨어를 통합하여 동시에 개발, 검증을 하고 전체 시스템에서 하드웨어로 처리할 부분과 소프트웨어로 처리할 부분을 최적으로 분할하여 가격대 성능비를 향상 시킬 수 있으며 하드웨어와 소프트웨어의 설계 시간, 설계비용, 에러의 감소로 인해 임베디드 시스템과 SoC의 설계에 적합하다. 최근에 이러한 요구에 의하여 JBIG2 핵심 모듈들의 하드웨어가 개발되고 있다[12][13].

심벌 ID 부호화는 기존의 텍스트 압축방법과 유사하다고 불 수 있다. 기존의 텍스트 압축방법은 컴퓨터에 의해 생성된 문자의 발생확률에 의거하여 가변장부호화를 수행하는 것이다. 가변장부호화에는 정적 및 동적 허프만부호화가 사용되는 경우가 많다[14]. 본 논문에서 다룬 심벌 ID부호화는 스캔된 문서로부터 추출된 연결된 객체를 사전에 저장하고 인덱싱하는데 사용한다. JBIG2 표준에서는 런코드(RUNCODE)를 이용하여 압축을 시행한다. 본 논문에서는 JBIG2 심벌 ID의 런길이(run-length) 부호화를 위한 런코드 부호기를 설계하고 하드웨어로 최초로 구현하였다. 런코드 부호기는 런길이의 허프만 트리 부호기로 구성되어 있다. 런코드와 유사한 압축 방식으로는 사전 부호화 방식의 일종인 zlib 부호화가 있고, 이 방식의 자세한 내용은 RFC1951 문서에 있다[15].

본 논문의 구성은 2장에서 기존의 텍스트부호화와 JBIG2 심벌 ID 부호기에 대하여 기술하고, 3장에서는 설계 구현된 런코드 부호기의 하드웨어 설계에 대해 기술하였다. 4장에서는 설계 구현된 하드웨어에 대한 실험, 설계 검증과 성능분석을 하였고, 5장에서 결론을 이끌어 내었다.

Ⅱ. 텍스트 압축 방식

2-1 기존의 압축 방식

텍스트는 파일형태로 저장된 것으로 가정하며, 이것을 손실 없이 압축하는 것이 텍스트 압축이다. 텍스트의 형태로는 문서뿐만 아니라 영상, 혹은 다른 종류의 데이터도 가능하다. 이는 파일로 저장될 때, ASCII 코드의 형태로 저장되므로 이것을 활용하여 약 30～50% 정도의 데이터압축을 할 수 있다. 이것의 원리는 ASCII 코드의 발생확률이 코드별로 크게 다르게 나타나는 것을 이용한다. ASCII 코드는 초창기에는 7비트로 표현이 가능했으나 최근에는 1바이트로 확장되었다. 한글의 경우는 완성형(KSC-5601

및 UTF-8), 조합형으로 표현이 가능하다. 완성형은 2바이트로 코딩이 가능하며, 조합형은 2바이트 중에 첫 비트는 구분비트, 나머지 15비트는 5비트씩 초, 중, 종성을 표시하는 것이 가능하다. 한글의 속성상 조합형이 합리적이지만 최근에 윈도우즈에서 완성형을 지원하면서 조합형은 사라지게 되었다. 더욱이 유니코드로 전 세계의 모든 문자가 표현가능하게 됨으로써 완성형도 조만간 사라질 수 있다. 그러나, 이러한 텍스트 표현은 고정길이이므로 좀 더 높은 압축효율을 갖기 위해 가변길이로 압축하는 것이 필요하다.

텍스트 압축에 가장 많이 적용되는 방법은 허프만 부호화가 있다. 이는 최적의 통계적 부호화방식으로 ASCII 코드로 표현된 각 심벌의 발생확률에 의해 부호어를 새롭게 할당한다. 허프만부호화는 프리픽스 코드를 이용한다. 프리픽스코드는 복호가 용이하다는 장점이 있다. 정적 허프만 부호화는 미리 만들어진 고정된 허프만테이블을 이용하여 전체 텍스트를 압축한다. 허프만 테이블은 텍스트의 종류에 관계없이 고정된 것을 이용한다. 이 정적 허프만 부호화는 2가지 단점이 있다. 첫째로, 텍스트의 문자 발생확률을 사전에 알지 못한다면 소스 텍스트는 두 번 읽어야 하는 점이다. 두 번째로, 허프만트리를 압축된 파일에 포함하여 전송해야 한다. 반면, 동적 허프만 부호화는 모든 심벌이 각각 입력될 때마다 허프만 트리가 갱신된다. 허프만트리는 전송하지 않고, 복호기에서 부호기에서와 마찬가지로 트리가 만들어 진다. 이것이 가능한 이유는 허프만 트리의 siblings 성질 때문이다[14].

허프만 부호화보다 성능이 우수한 텍스트 압축방식으로 Ziv와 Lempel이 개발한 LZ77[16], LZ78[17]은 텍스트의 문자로부터 사전을 구성해서 압축하는 부호화 기법의 대표적인 알고리즘이다. 이러한 사전을 이용하는 부호화 방법들은 문자를 하나씩 부호화하지 않고 문자열을 window를 이용하여 부호화하는 방법이다. LZ77 방법은 window와 look-ahead 버퍼를 두어서 look-ahead 버퍼에서 window와 중복되는 연속된 부분열을 찾아 포인터로 치환하여 부호화하는 방법이며, LZ78 방법은 사전을 가지고 텍스트를 입력받아 처음 나오는 문자열은 사전에 저장하고 이미 나왔던 부분열은 사전의 포인터로 치환하여 부호화하는 방법이다. 이러한 방법들은 일치하는 문자열이 연속적이지 않고 한 문자일 경우 부호화를 위해 영문자의 경우 한 바이트를, 한글의 경우 두 바이트를 사용하여야 하는 비효율성이 존재하게 된다. LZ77 계열 중 하나인 LZSS는 LZ77 계열이 가지고 있는 일치된 문자열을 이진 탐색 트리를 사용하여 아주 빠른 탐색이 가능하며 복호화가 간단하고 빠르다. 그리고 LZ77 압축 방법의 포인터 표기 방식을 개선하여 압축률의 향상을 보인다.

2-2 JBIG2 심벌 ID 부호기

그림1은 JBIG2 부호화기 중 본 논문과 관련된 부분에 대한 간략한 블록도이다.

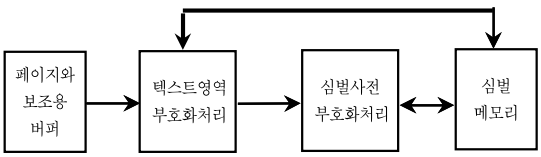


그림 1. 심벌ID 코드길이 부호화 블록도
Fig. 1. Symbol ID Code Length Encoding Block-diagram

부호화할 문서를 스캐닝하여 찾아진 심벌들은 새로운 심벌로 사전에 등록한다. 심벌 사전은 입력과 동시에 구성되고, 새로운 심벌은 이전에 저장된 심벌과 비교하여 추가여부를 결정한다. 심벌 사전에는 가로, 세로크기와 비트맵을 저장하게 된다. 심벌의 저장의 효율을 높이기 위해 추출된 모든 심벌을 높이 순서로 정렬한다. 같은 높이를 갖는 심벌은 하나의 클래스로 묶이게 되고, 같은 클래스에서는 폭이 큰 것부터 작은 순서로 정렬되어 순서에 따라 높이가 같은 심벌 비트맵들을 하나로 통합하여 MMR 혹은 MQ 산술부호화 하게 된다. 동일 클래스 내에서는 폭의 차이 값을, 클래스 간에는 높이 차이 값을 허프만 혹은 정수 MQ 산술부호화하게 된다.

텍스트영역 부호기에서는 심벌의 문서상에서의 수평 및 수직 위치와 심벌의 ID를 부호화하게 된다. 텍스트는 소수라인의 스트립(Strip) 단위로 높이를 분할하게 된다. 부호화 효율을 높이기 위해 위치의 절대 값보다 각 스트립 단위 내에서 현재 심벌과 이전

심벌과의 가로 및 세로위치의 차이 값을 부호화하게 된다. 첫 스트립에서 첫 번째 심벌의 가로 위치 증가 값은 심벌영역의 가로 시작 위치를 기준으로 차이 값을 부호화하고 첫 스트립이 아닌 경우에는 이전 스트립의 첫 심벌의 위치를 기준으로 차이 값을 허프만이나 MQ 산술부호화를 하게 된다. 각 심벌의 사용횟수(Used Count)에 따라 각 심벌의 가변 런길이를 구하고, 런길이를 복호기에 전송한다. 아울러 사용된 심벌의 ID는 런길이에 의해 허프만 부호화하여 구한 후 텍스트영역 부호화에 사용한다. 복호기에서는 전송되어 온 런코드 정보로부터 심벌ID를 허프만부호기에 의해 구해 텍스트영역 복호화에 사용된다. 이와 같이 심벌 ID 부호화에 핵심적으로 사용되는 허프만 부호화는 고정된 길이의 입력 정보를 가변 길이의 부호로 변환하기 위한 정보 압축 방법이다. 일반적으로 입력 정보에 대한 허프만 부호화를 실행할 때는 사전 정의된 허프만 테이블을 이용하지만 심벌 ID 코드 길이 부호화에서는 사전 내의 심벌의 개수와 패턴이 유동적이어서 적응적으로 부호 길이와 코드를 생성하게 된다.

런코드의 정의는 표1과 같다. 런코드의 생성은 심벌 ID 코드 길이에 대한 연속된 수를 표현한다. 런(Run)이라 표현된 연속된 동일한 코드 길이의 묶음을 하나 또는 그 이상의 런코드와 첨가 비트로 표현하였다. 예를 들어, 런코드1은 코드길이가 1인 코드이다. 런코드32~34는 이전 심벌 ID 코드 길이가 반복됨을

표 1 . 각 RUNCODE의 정의
Table 1. Definition of each RUNCODE

RUNCODE0	Symbol ID code length is 0
RUNCODE1	Symbol ID code length is 1
RUNCODE2	Symbol ID code length is 2
RUNCODE3	Symbol ID code length is 3
RUNCODE4	Symbol ID code length is 4
:	:
:	:
RUNCODE31	Symbol ID code length is 31
RUNCODE32	Copy the previous symbol ID code length 3-6 times. The next two bits, plus 3, indicate this repeat length.
RUNCODE33	Repeat a symbol ID code length of 0 for 3-10 times. The next three bits, plus 3, indicate this repeat length.
RUNCODE34	Repeat a symbol ID code length of 0 for 11-138 times. The next seven bits, plus 11, indicate this repeat length.

의미하며, 뒤에 추가된 비트는 반복 횟수를 의미한다. 예를 들어, 런코드32(2)는 이전 심벌 ID코드 길이가 반복됨을 의미하며, (2)는 표의 정의에 따라 3을 더하여 이전 심벌 ID 코드 길이를 5회 반복함을 의미한다. 그러므로 결과적으로 동일한 코드 길이를 갖는 6개의 심벌이 발생했음을 의미한다. JBIG2에서 위의 서술에 따르는 심벌 ID 코드 길이 부호화를 위한 런코드 부호화 알고리즘은 그림 2와 같은 순서를 따르며, 예를 들어 설명하기로 한다. 심벌 사전에 32개의 심벌이 있다고 가정하고, 각 심벌이 텍스트영역에서 얼마나 사용되었느냐를 의미하는 사용빈도는 표2의 두 번째 열과 같다고 가정한다. 심벌의 사용빈도에 의해 표준 허프만 트리 알고리즘을 이용하여 코드 길이를 구하면 표2의 세 번째 열과 같이 표현할 수 있다. 그리고 이들 코드길이를 표2의 네 번째 열과 같이 연속 코드 길이끼리 그룹(Runs)으로 묶는다. 32개의 심벌의 코딩에 사용된 런코드의 발생횟수를 계산하면 다음과 같다.

RUNCODE0	1	RUNCODE3	2	RUNCODE4	2
RUNCODE5	1	RUNCODE6	2	RUNCODE7	5
RUNCODE8	1	RUNCODE9	2	RUNCODE32	3
RUNCODE33	1				

런코드의 발생횟수에 따라 표준 허프만 트리 알고리즘을 이용하여 코드 길이를 구하면 다음과 같다.

RUNCODE0	5	RUNCODE3	3	RUNCODE4	3
RUNCODE5	5	RUNCODE6	3	RUNCODE7	2
RUNCODE8	5	RUNCODE9	3	RUNCODE32	3
RUNCODE33	5				

위 코드 길이에 대하여 JBIG2 FCD[4]의 부록 B.3의 프리픽스 코드 할당 알고리즘을 이용하여 코드를 할당하면 다음과 같다.

RUNCODE0	11100	RUNCODE3	010
RUNCODE4	011	RUNCODE5	11101
RUNCODE6	100	RUNCODE7	00
RUNCODE8	11110	RUNCODE9	101
RUNCODE32	110	RUNCODE33	11111

JBIG2 부호기는 런코드33→런코드9→런코드6 등

의 순서로 위 프리픽스코드를 전송한다. 한편 32개의 심벌들은 사용빈도에 따라 허프만부호기에 의해 가변길이 부호화를 한다. 텍스트영역 부호기에서는 심벌의 가변부호와 (x,y)의 위치를 전송한다. JBIG2 복호기에서는 전송되어온 런코드로 부터 심벌의 코드길이를 추출한 후, 각 심벌의 가변길이 부호를 구한다. 이 부호를 이용하여 텍스트영역의 심벌 ID를 복호화한다.

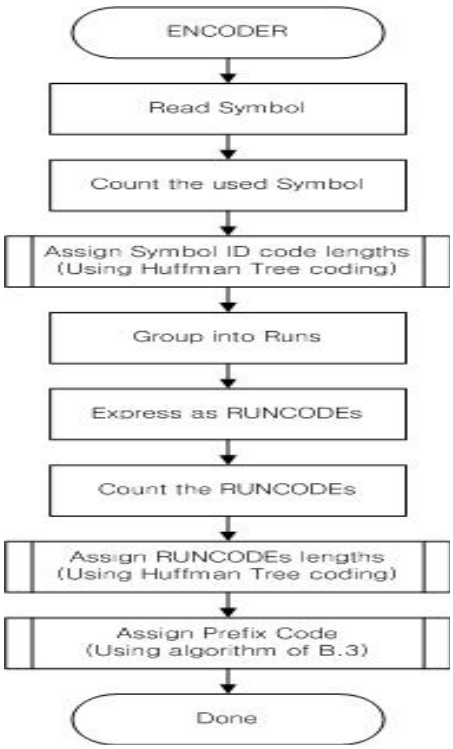


그림 2. 런코드 부호화 과정
FIg. 2. RUNCODE Encoding Procedure

Ⅲ. 런코드 부호기의 하드웨어 설계

임베디드용 JBIG2 부호기중 심벌 ID 부호화를 위한 런코드 부호기의 하드웨어를 설계 구현하기 위해 3단계의 과정을 걸친다. 첫 번째로, Visual Studio를 사용하여 해당 부호기를 C언어로 개발한다. 두 번째로, ImpulseC CoDeveloper[18] 툴을 사용하여 하드웨어 모듈과 소프트웨어 모듈의 분리 설계를 진행한다. 두 모듈 간에 공유메모리를 설치할 수도 있으며, 스트림과 신호를 통해 데이터를 주고받도록 설계하였다. 하드웨어 모듈은 먼저 C로 구현되며, 시뮬레이션이 성공하면 VHDL이나 verilog 코드로 변환된다. 세 번째로, 개발된 하드웨어 모듈과 소프트웨어 모듈은 Xilinx ISE/EDK를 사용하여 비트 스트림으로 합성하여, ML410 임베디드 보드에 하드웨어로 다운로드/추가 한 후, UART 터미널을 통해 동작검증을 수행한다.

표 2. 런코드 부호화의 예
Table 2. The example of RUNCODE Encoding

심 벌	사용빈도 (Used count)	심벌 ID 코드 길이	연속 횟수 (Runs)	런코드 (RUNCODEs)
심벌 #1	0	0	0이 3번 연속	RUNCODE33(0)
심벌 #2	0	0		
심벌 #3	0	0		
심벌 #4	1	9	9가 1번	RUNCODE9
심벌 #5	8	6	6이 4번 연속	RUNCODE6
심벌 #6	8	6		RUNCODE32(0)
심벌 #7	8	6		
심벌 #8	8	6		
심벌 #9	64	3	3이 1번	RUNCODE3
심벌 #10	32	4	4가 6번 연속	RUNCODE4
심벌 #11	32	4		RUNCODE32(2)
심벌 #12	32	4		
심벌 #13	32	4		
심벌 #14	32	4		
심벌 #15	32	4		
심벌 #16	0	0	0이 1번	RUNCODE0
심벌 #17	4	7	7이 1번	RUNCODE7
심벌 #18	1	9	9가 1번	RUNCODE9
심벌 #19	2	8	8이 1번	RUNCODE8
심벌 #20	4	7	7이 1번	RUNCODE7
심벌 #21	16	5	5가 6번	RUNCODE5
심벌 #22	16	5		RUNCODE32(2)
심벌 #23	16	5		
심벌 #24	16	5		
심벌 #25	16	5		
심벌 #26	16	5		
심벌 #27	64	3	3이 1번	RUNCODE3
심벌 #28	8	6	6이 1번	RUNCODE6
심벌 #29	4	7	7이 1번	RUNCODE7
심벌 #30	32	4	4가 1번	RUNCODE4
심벌 #31	4	7	7이 2번	RUNCODE7
심벌 #32	4	7		RUNCODE7

3-1 하드웨어 설계

각 심벌들의 사용횟수(Used Symbol Count)는 16비트(15:0)의 크기를 가지는 메모리에 저장된다. 예를 들어 설명한, 그림3을 보면 심벌들은 사용횟수에 의해 정렬을 하게 되고 그 값을 자식노드 L,R 메모리에

할당하도록 하였다. 이때 L,R 자식노드 메모리는 16비트(15:0)의 크기를 가지는 2000개의 메모리를 사용 횟수(Used Symbol Count)와 자식노드들이 가지고 있는 UseCount의 개수 합을 저장하는 SubCount 용으로 할당하였다. L,R 자식노드 메모리에 저장된 값들은 허프만 코딩을 통해 생성된 트리에서 0의 값을 제외한 값들을 순차적으로 저장하고 있다. L, R 자식노드 메모리는 UseCount와 SubCount 값을 이용하여 완전한 트리를 생성하고생성된 완전트리를 이용하여 각 심벌들의 트리에서의 레벨(Symbol ID CodeLength)를 구한다. 구해진 Symbol ID CodeLength 값을 이용하여 런길이를 구하고 이어서 런코드를 구하도록 설계하였다.

Ⅳ. 실험 및 결과

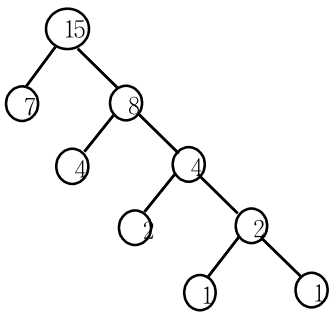
4-1 설계 환경 및 실험 방법

실험에 사용된 개발보드는 Xilinx사에서 만든 ML410 보드로서 Virtex-4 FX60 FPGA칩을 내장하고 있다. 부호기의 하드웨어 구현을 위해 Impulse C Codeveloper를 사용하여 소프트웨어와 하드웨어 모듈, 디바이스드라이버를 구현한 후, Xilinx사의 ISE와 EDK Tool[19]을 사용하여 하드웨어를 합성하였고,

하드웨어의 파형 검증은 Active-HDL[20]을 사용하여 실시하였다. 최종적으로 ML410보드에 Microblaze CPU를 합성한 후, 개발된 하드웨어를 FPGA에 다운로드하여 개발보드에 부속된 Flash 메모리에 저장된 이진문서를 로딩하여 S/W와 연동을 통해 압축하는 실험을 수행함으로써 정상동작여부를 검증하였다. 성능평가는 첫째로, 합성된 하드웨어의 FX-60의 사용가능한 Slice중에서 몇 %를 소모했는지 측정한다. 둘째, 설계된 하드웨어와 소프트웨어의 연동했을 때와 Microblaze 코어를 이용하여 소프트웨어만으로 처리했을 때의 소요되는 처리시간을 비교하였다. Microblaze 코어는 내부에 타이머가 없기 때문에 OPB-Timer를 이용하여 처리 시간을 측정하였다. OPB-Timer는 하드웨어가 동작될 때 클럭을 측정하는 타이머로써 단위는 ticks이고, ticks에 프로세서의 동작 주기를 곱하면 동작시간이 된다. 공정한 동작시간을 얻기 위해 여러번 반복 수행 후 평균 처리 속도를 구했다. 셋째로, FX-60의 IOB를 얼마나 사용하는지 측정하였다.

4-2 실험 결과

4-2-1 설계 모듈의 파형 검증



Used Symbol Count
1
4
1
2
7

L 노드

index	Freq.	SubCount
1	7	0
2	4	0
3	2	0
4	1	0

R 노드

index	Freq.	SubCount
1	8	8
2	4	4
3	2	2
4	1	0

그림 3. 허프만 트리의 생성 예

Flg.3. The example of creating of Huffman Tree

설계된 부호기에 JBIG2 FCD[4] 부록에 있는 테스트 심벌 32개를 클럭 주파수 75MHz에서 입력하여, 출력을 확인함으로써 정상으로 동작함을 알 수 있었다. 그림4는 실험을 통해 출력된 전체 파형이며, 그림5는 입력 심벌 데이터가 정상적으로 입력됨을 나타내고 있다. 그림6에서 부호화된 데이터의 첫 출력이 15.02 μs에서 '0x4'가 출력됨으로써 설계된 하드웨어가 정상적으로 동작하고 있음을 확인하였다.

4-2-1 성능 검증 및 평가

설계된 하드웨어들은 Xilinx ISE/EDK를 이용하 ML410 개발 보드 상의 FPGA에 하드웨어로 추가하였고, 소프트웨어와의 연동으로 이진문서를 압축하는 실험을 수행하였다. 소프트웨어와의 연동 실험을 통하여 소프트웨어에서 데이터를 하드웨어 측으로 전송하고 하드웨어 내에서의 동작을 생성된 데이터 스트림을 소프트웨어 측에서 다시 전송받아 그 결과를 확인하는 방식으로 설계된 하드웨어가 정상 동작됨을 UART 터미널을 통해 확인 검증하였다. 그림 7은 런코드 부호기가 정상적으로 동작됨을 표시하고 있다. 런코드 부호기는 전체 사용가능한 25,280 Slice의 13%인 3,414 Slice를 사용하였다. 런코드 부호기의 성능평가는 표준 테스트 문서(H01_200.bmp 영상)를 소프트웨어만으로 런코드 부호화 할 때와 하드웨어로 런코드를 부호화 할 때 소요되는 시간을 측정함으로 수행하였다. 공정한 시간측정을 위하여 10번의 수행을 한 후 평균시간을 측정 했을 때 소프트웨어로

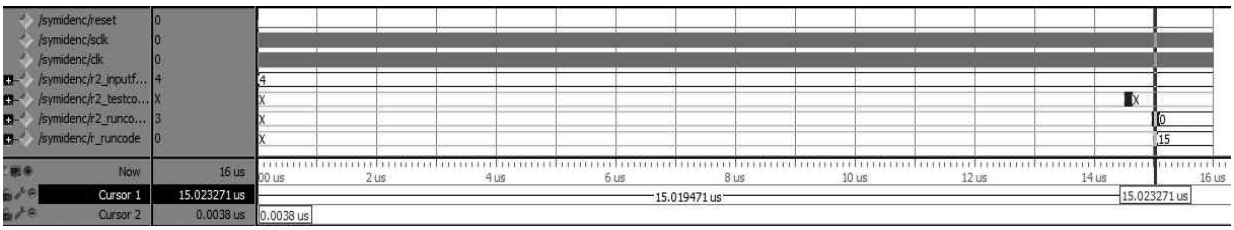


그림 4. 전체 파형
Fig. 4. Full Waveform

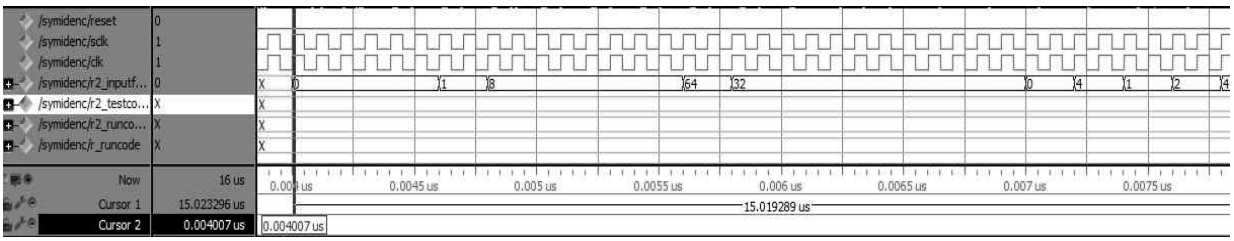


그림 5. 입력 심벌 데이터 파형
Flg. 5. Input Symbol Data Waveform

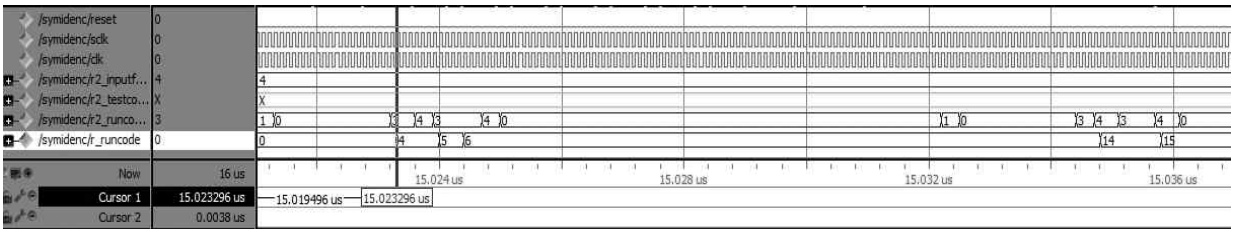


그림 6. 출력 데이터 파형
Fig. 6. Output Data Waveform

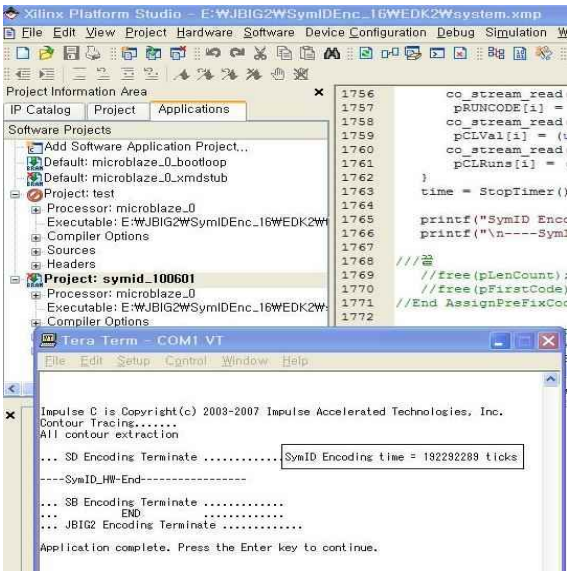


그림 7. 부호기의 하드웨어/소프트웨어 연동 실험 결과

Flg. 7. Hardware/Software co-operation test result of the Encoder

수행한 경우 평균 78억3천1백만 ticks가 소요되었고, 하드웨어로 수행했을 경우 평균 1억 9천 2백만 ticks가 소요되었다. 클럭이 75MHz에서 시간으로 환산하면 각각 약 104.41 sec와 2.56 sec가 된다. 하드웨어를 이용할 경우 소프트웨어만으로 동작시켰을 때보다 약 40배 이상 빠른 수행을 보여 주었다. 또한 그림 8에서 보듯이 사용된 IOB는 전체의 11%만을 사용하여 하드웨어로의 구현에 적합함을 확인하였다.

SYMBOLID_16_0518 Partition Summary

No partition information was found.

Device Utilization Summary (estimated values)

Logic Utilization	Used	Available	Utilization
Number of Slices	3414	25280	13%
Number of Slice Flip Flops	1864	50560	3%
Number of 4 input LUTs	6437	50560	12%
Number of bonded IOBs	40	352	11%
Number of FIFO16/RAMB16s	57	232	24%
Number of GCLKs	1	32	3%

그림 8. 구현된 하드웨어 블록 전체의 FPGA 사용량

Fig. 8. FPGA Usage of the Implemented Hardware Total Block

V. 결 론

본 논문에서는 영역기상방송을 비롯한 다양한 분야에서 널리 사용되고 있는 팩시밀리의 성능개선을 위해 제정된 새로운 이진 압축표준인 JBIG2 하드웨어시스템의 주요 부분인 심벌 ID 코드 길이 부호화를 위한 런코드 부호기를 하드웨어로 설계 및 구현하였다. 현재 JBIG2 표준화는 완료되어 있는 상태로 소프트웨어를 이용한 여러 응용기술들은 PDF 파일에서 채택하여 사용하고 있지만 하드웨어의 구현된 사례는 국내외적으로 거의 찾아 볼 수 없다. 본 논문은 먼저 하드웨어와 소프트웨어의 연동설계를 위하여 ImpulseC CoDeveloper 툴을 사용하여 각 모듈을 설계하였다. 그리고 ALDEC사의 Active-HDL 툴을 사용하여 입력 테이터에 대한 파형 검증을 거친 후 Xilinx사의 ISE/EDK를 사용하여 하드웨어로 합성하였다. 실험 및 검증에는 Xilinx사의 ML410 실험보드상의 Virtex-4 FX60 FPGA를 이용하였다. 실험 결과, 설계된 런코드 부호화는 3,414 Slices를 사용하여 전체 FPGA Slice 용량의 13%를 사용하였고, 런코드 부호화를 위해 소요된 시간은 2.56sec로 소프트웨어로만 처리한 경우보다 약 40배 이상 빠른 처리 속도를 보였다. 또한 각 모듈은 JBIG2 소프트웨어와의 연동 실험을 실시하여 정상적으로 동작함을 확인하였다.

향후 과제로는 고속의 임베디드 시스템에 적합한 JBIG2 전체 시스템 개발을 위해 처리 속도를 향상시킨 심벌사전에 있는 기존의 심벌과 새로 입력되는 심벌을 고속으로 비교하는 심벌 비교 및 매칭부의 하드웨어 개발이 필요하다.

감사의 글

본 연구는 2010학년도 교내 학술연구비에 의하여 이루어졌음.

참 고 문 헌

[1] CCITT Draft Rec. T.4, *Standardization of Group 3 Facsimile Apparatus for Document Transmission*, 1979.

[2] CCITT Rec. T.6, *Facsimile Coding Schemes and Coding Control Function for Group 4 Facsimile Apparatus,* 1988

[3] ITU-T Rec. T.82, *Information Technology - Coded Representation of Picture and Audio Information - Progressive Bi-Level Image Compression,* March, 1993.
[4] ISO/IEC JTC1/SC29/WG1 (ITU-T SG8) N1359, *JBIG2 Final Committee Draft,* July, 1999.
[5] P.G. Howard, "AT&T JBIG2 Coder Proposal," *ISO/ISE JTC1/SC29/WG1,* Feb, 1996.
[6] P.G. Howard, F. Kossentini, B. Martins, S. Forchhammer, and W.J. Rucklidge, "The Emerging JBIG2 Standard," *IEEE Trans. Circuits, Syst. Video Technol.*, vol. 8, no. 7, pp. 838-848, Nov. 1998.
[7] http://www.adobe.com
[8] http://www.leadtools.com
[9] http://www.verypdf.com
[10] O. Johnsen, J. Segen, and G. Cash, "Coding of Two-Level Pictures by Pattern Matching and Substitution," *Bell System Technical Journal*, vol. 62, no. 8, Oct., 1983.
[11] P. G. Howard, "Lossless and Lossy Compression of Text Images by Soft Pattern Matching," *ISO JTC1/SC29/WG1*, N205, 1995.
[12] 박경준, 고형화, "JBIG2 허프만 부호화기의 하드웨어 설계," *한국멀티미디어학회논문지*, pp. 101 - 109, Feb., 2009
[13] 서석용, 고형화, "임베디드용 JBIG2 부호화기의 하드웨어 설계," *한국통신학회논문지*, Vol. 35, No. 2, pp. 182 - 192, Feb., 2010
[14] M. J. Atallah, *Algorithms and Theory of Computation Handbook. Chapter 12*., CRC Press, 1998
[15] http://www.faqs.org/rfcs/rfc1951.html
[16] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Transactions on Information Theory,* vol. 23, Issue 3, pp. 337 - 343
[17] J. Ziv and A. Lempel, "Compression of Individual Sequences via Variable-Rate Coding," *IEEE Transactions on Information Theory.*, pp. 530 - 536, Sept., 1978.
[18] http://www.impulsec.com
[19] 김 혁, Real Xilinx Processor World. *엔트미디어*, 2005
[20] http://www.aldec.com

서 석 용 (徐錫用)

1996년 2월 : 관동대학교 전자통신공학과 학사 졸업
2000년 8월 : 광운대학교 전자통신공학과 석사 졸업
2001년 3월~현재 : 광운대학교 전자통신공학과 박사과정

관심분야 ; 영상신호처리, 임베디드 시스템, JBIG2, MPEG-4

고 형 화 (高亨和)

1979년 2월 : 서울대학교 전자공학과 학사 졸업
1982년 2월 : 서울대학교 전자공학과 석사 졸업
1989년 2월 : 서울대학교 전자공학과 박사 졸업
1985년 3월~현재 : 광운대학교 전자통신공학과 교수

관심분야 ; 영상신호처리, 임베디드시스템, JBIG2, H.264, Network Video 처리