J. Korean Soc. Aeronaut. Space Sci. 48(10), 831-835(2020)
DOI:https://doi.org/10.5139/JKSAS.2020.48.10.831
ISSN 1225-1348(print), 2287-6871(online)

HLS를 이용한 텔레메트리 표준 106-17 LDPC 부호기 설계

구영모[1], 이운문[2], 김복기[3]

Telemetry Standard 106-17 LDPC Encoder Design Using HLS

Young Mo Gu[1], Woonmoon Lee[2] and Bokki Kim[3]
Inha Technical College[1], DANAM SYSTEMS[2,3]

ABSTRACT

By automatically generating HDL codes from C/C++ source codes, HLS makes it possible to shorten FPGA system developing period through easy timing control and structure change. We designed LDPC encoder for telemetry standard 106-17 with Xilinx Vivado HLS and showed hardware structure can be easily adapted for different purposes through minor C code modification. Synthesis results targeting Spartan-7 xc7s100 device are presented for throughput and hardware complexity comparison.

초 록

HLS는 C/C++ 언어로 기술된 소스 코드로부터 자동으로 HDL 코드를 생성하므로 타이밍이나 제어가 간단하고 하드웨어 구조를 쉽게 변경할 수 있어 FPGA 시스템 개발 기간을 단축할 수 있는 장점이 있다. 본 논문에서는 Xilinx사의 Vivado HLS를 이용하여 텔레메트리 표준 106-17 LDPC 부호기를 설계할 때 간단한 코드 수정으로 목적에 맞는 구조 변경의 용이함을 보이고 Spartan-7 xc7s100 디바이스를 타겟으로 합성하여 throughput과 하드웨어 복잡도 등의 결과를 비교하였다.

Key Words : Telemetry(텔레메트리), LDPC(저밀도 패리티 검사 부호), Encoder(부호기), HLS(고급 합성), CCSDS(우주 데이터 시스템 자문위원회)

Ⅰ. 서 론

Low Density Parity Check (LDPC) 부호[1]는 선형 부호의 일종으로 패리티 체크 행렬의 1의 밀도가 매우 낮은 부호인데 부호의 성능이 매우 우수하고 구조상 부호 및 복호의 throughput을 높일 수 있어 다양한 통신 분야의 표준에서 오류 정정 부호로 채택되었다[2-6].

우주 통신을 위한 Consultative Committee for Space Data Systems(CCSDS)[5]에서도 LDPC 부호를 표준으로 채택하고 있는데 정보 비트수가 각각 1024, 4096, 16384의 세 가지를 규격으로 정하고 있다. 텔레메트리 표준 106-17[6]에서는 이중에서 정보 비트수가 1024, 4096인 LDPC 부호만을 규격으로 채택하고 있다.

통상적으로 Field Programmable Gate Array(FPGA)를 이용한 하드웨어 구현은 VHDL이나 Verilog와 같은 Hardware Design Language(HDL)을 이용하여 설계하는데 High-Level Synthesis(HLS)는 C나 C++ 언어로 기술된 소스 코드로부터 자동으로 HDL 코드를 생성하므로 설계 시 복잡한 타이밍이나 제어를 고려

† Received : July 13, 2020 Revised : August 25, 2020 Accepted : September 16, 2020
[1] Professor, [2] Principal Research Engineer, [3] CTO
[1] Corresponding author, E-mail : ymgu@inhatc.ac.kr, ORCID 0000-0002-3605-783X

하지 않아도 되고, C/C++ 소스 코드를 간단히 수정하여 하드웨어 구조를 변경하는 것이 용이하여 개발 기간을 단축할 수 있는 것이 장점이다.

본 논문에서는 Xilinx사의 Vivado HLS를 이용하여 텔레메트리 표준 106-17 LDPC 부호기를 설계하는 방법을 제안한다. 하드웨어 구조 변경이나 복잡도에 따른 Xilinx Spartan-7 xc7s100 디바이스를 타겟으로 구현하여 그 결과를 비교한다.

II. 본 론

2.1 텔레메트리 표준 106-17 LDPC 부호

CCSDS 규격의 LDPC 부호는 정보 비트수가 각각 1024, 4096, 16384의 세 가지가 있는데 텔레메트리 표준 106-17 규격은 이중에서 정보 비트수가 1024, 4096인 LDPC 부호만을 규격으로 채택하고 있다. 정보 비트 수 k와 부호율에 따른 부호 비트 수 n은 Table 1과 같다.

LDPC 부호는 1의 밀도가 매우 낮은 패리티 체크 행렬 H에 의해 정의되는데 부호 벡터 $c=[c_0, c_1 \dots, c_{n-1}]$와 식 (1)의 관계가 성립한다. Fig. 1은 n=8192, k=4096, 부호율 1/2인 패리티 체크 행렬이다.

$$Hc^T = 0 \tag{1}$$

부호율을 L/(L+2) (L=2,4,8)라고 할 때, 패리티 체크 행렬 H는 다시 크기가 MxM인 부행렬(sub-matrix)로 구성되며 그 크기는 3Mx(L+3)M이다. 정보 비트수 k와 부호율에 따른 부행렬 크기 M은 Table 2와 같다. Fig. 1의 패리티 체크 행렬은 크기가 2048x2048인 부행렬이 세로축으로 3개, 가로축으로 5개로 구성되어 있다.

Table 1. Telemetry Standard 106-17 LDPC Code length n per information length k

k	n		
	1/2	2/3	4/5
1024	2048	1536	1280
4096	8192	6144	5120

Table 2. Sub-matrix size M per information length k and code rate

k	M		
	1/2	2/3	4/5
1024	512	256	128
4096	2048	1024	512

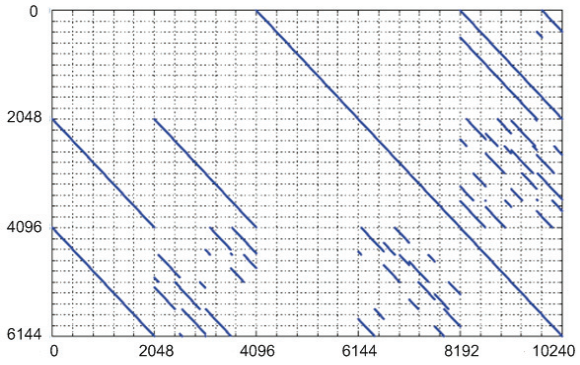

Fig. 1. Parity check matrix H for n=8192, k=4096, rate 1/2 [6]

CCSDS 규격 LDPC 부호의 특이한 점은 Tables 1, 2에 의하면 정보 비트 수 k가 LM, 부호 비트 수 n이 (L+2)M이므로 식 (1)이 성립하려면 행렬 H의 크기는 3Mx(L+2)M이 되어야 하는데 실제 크기는 3Mx(L+3)M인 것이다. 그 이유는 원래 부호기 출력의 크기는 (L+3)M 인데 송신기는 그 중에서 마지막 M 비트는 puncturing하여 송신하지 않고 (L+2)M 비트만 송신하기 때문이다. 따라서 부호기 및 복호기 설계 시 이 점을 반드시 고려해야 한다. Fig. 1에서 M이 2048인 패리티 체크 행렬의 크기가 6144x8192가 아니고 6144x10240인 이유는 송신기에서 부호기 출력 10240 비트 중에서 마지막 2048 비트는 puncturing하여 송신하지 않기 때문이다.

2.2 LDPC 부호기 구성

부호 벡터 c를 크기가 k인 정보 벡터 i와 패리티 벡터 p로 분리하면 $[i|p]$로 다시 쓸 수 있고 패리티 체크 행렬 H도 다시 크기가 3MxLM인 행렬 Q와 크기가 3Mx3M인 정방 행렬 P로 분리하여 $[Q|P]$와 같이 다시 쓸 수 있는데 (Fig. 1에서 L=2, M=2048이므로 Q의 크기는 6144x4096, P의 크기는 6144x6144), 식 (1)로부터 식 (2), (3), (4)를 거쳐 패리티 벡터 p를 구하는 식 (5)를 유도할 수 있다[6]. 식 (4)에서 덧셈은 modulo-2 연산이므로 (-)는 생략하였다.

$$Hc^T = [Q|P]\begin{bmatrix} i^T \\ p^T \end{bmatrix} = 0 \tag{2}$$

$$Qi^T + Pp^T = 0 \tag{3}$$

$$p^T = P^{-1}(-Qi^T) = (P^{-1}Q)i^T \tag{4}$$

$$p = i(P^{-1}Q)^T = iW,\ W = (P^{-1}Q)^T \tag{5}$$

식 (5)의 행렬 W는 원래 크기가 LMx3M 인데 puncturing하여 전송하지 않는 M 비트를 제외하면 그 크기는 LMx2M이고, 식 (6)과 같이 작은 cyclic

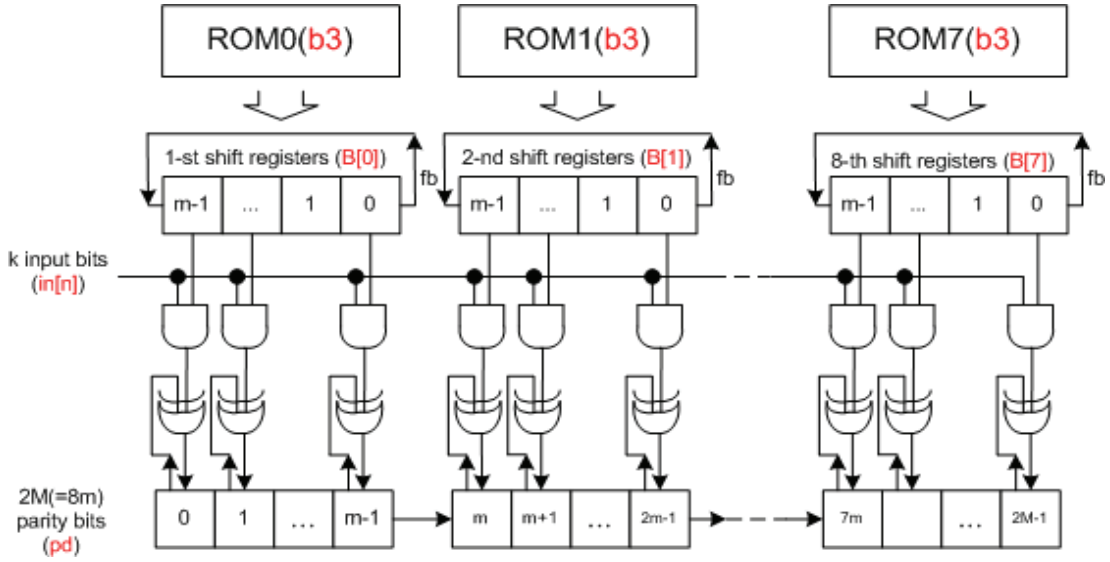

Fig. 2. LDPC encoder structure with feedback shift registers

행렬들로 구성된 quasi-cyclic 행렬이다. 식 (6)에서 각각의 행렬 B_{ij}는 크기가 mxm인 정방행렬이고, cyclic 행렬이므로 첫 번째 행벡터 b_{ij}를 cyclic shift하여 전체 행렬 B_{ij}를 표현할 수 있다. m = M/4의 관계가 성립하여 행렬 W의 크기는 4Lmx8m로 다시 쓸 수 있다.

$$W=\begin{bmatrix} B_{00} & \cdots & B_{07} \\ \vdots & \ddots & \vdots \\ B_{(4L-1)0} & \cdots & B_{(4L-1)7} \end{bmatrix} \quad (6)$$

식 (5)와 식 (6)을 이용한 패리티 비트 생성기의 구성도는 Fig. 2와 같다. 행렬 원소 간의 곱셈은 AND 게이트로, modulo-2 덧셈은 exclusive-OR 게이트로 구현하고 패리티 비트를 저장하는 2M 비트 레지스터는 동작 시작 전에 0으로 초기화한다. 패리티 생성기의 8개의 피드백 시프트 레지스터는 동작을 시작하기 전에 b_{0j} (j=0,1,⋯,7)로 초기화되고 매 입력 비트가 들어올 때마다 오른쪽으로 피드백 시프트 동작을 하면서 m개의 입력 비트가 들어올 때까지 동작한다. 그 다음 8개의 피드백 시프트 레지스터는 ROM에서 읽은 b_{1j} (j=0,1,⋯,7)로 초기화되고 다시 m개의 입력 비트가 들어올 때까지 반복 동작을 하는 식으로 전체 k개의 입력 비트에 반복 동작을 수행한다. 마지막 m개의 입력 비트에 대한 동작을 마치면 Fig. 2의 레지스터 pd에 저장된 2M 비트가 최종 계산된 패리티 비트이다.

2.3 HLS를 이용한 부호기 설계 1

통상적으로 FPGA를 이용한 하드웨어 구현은 VHDL이나 Verilog와 같은 HDL을 이용하여 설계하는데 Xilinx사의 Vivado HLS는 C/C++소스 코드로부터 자동으로 HDL 코드를 생성하므로 복잡한 타이밍이나

```
void encoder(ap_uint<1> in[4096], ap_uint<1> out[4096]) {
    int n, i, j, q;
    ap_uint<512> B[8];
    ap_uint<1> fb;
    ap_uint<4096> pd;

    l1: for(n=0;n<4096;n++) pd[n] = 0; // UNROLL directive
    l2: for(n=0;n<4096;n++) { // PIPELINE directive
        if(n%512==0) { // register initialize
            q = n/512;
            l21: for(i=0;i<8;i++) {
                B[i].range(511,0) = b3[8*q+i];
            }
        }
        else { // register circular shift-right
            l22: for(i=0;i<8;i++) { // UNROLL directive
                fb = B[i][0];
                B[i] >>= 1;
                B[i][511] = fb;
            }
        }
        l23: for(i=0;i<8;i++) { // UNROLL directive
            l231: for(j=0;j<512;j++)
                pd[512*i+j] = pd[512*i+j] xor (in[n] and B[i][511-j]);
        }
    }
    l3: for(n=0;n<4096;n++) out[n] = pd[n];
}
```

Fig. 3. HLS C source code for LDPC encoder

제어부를 고려하지 않아도 되는 편리함이 있고, 하드웨어 구조를 쉽게 변경할 수 있다. Fig. 3은 Fig. 2의 패리티 비트 생성기에 바탕을 둔 정보 비트수 k = 4096, 부호율 1/2(m = 512)일 때의 부호기 구현 C 코드이다. Vivado HLS는 합성을 최적화하는 다양한 directive[7]를 제공하는데 이를 주석으로 표시하였다. UNROLL directive는 for loop를 순차적으로 구현하지 않고 동시에 구현하도록 하여 메모리가 아니라 레지스터로 합성하게 하며, PIPELINE directive는 for loop를 pipeline으로 구현하여 지연(delay)을 최소화한다.

Figure 3의 C 소스 코드를 Xilinx spartan7 xc7s100 디바이스를 타겟으로 합성한 결과는 Table 3과 같다. 클럭 주기 타겟을 각각 10ns, 5ns, 2.5ns로 설정하여

Table 3. HLS Synthesis Results of C source code of Fig. 3

Target	10ns	5ns	2.5ns
Synthesized clock period	5.032ns	2.771ns	1.708ns
latency	8195	8196	8197
Throughput (Mbps)	99.3	180.4	292.6
BRAM_18K (240)	116 (48%)	116 (48%)	116 (48%)
FF (128000)	8226 (6.4%)	12326 (9.6%)	16486 (12.9%)
LUT (64000)	20674 (32.3%)	20677 (32.3%)	20709 (32.4%)

합성을 한 결과, 합성 클럭 주기 값이 각각 5.032ns, 2.771ns, 1.708ns로 합성되었다. 지연(latency) 값은 입력 4096 비트를 읽는 데 4096 클럭, 계산된 4096 패리티 비트를 읽는 데 4096 클럭이 소요되므로 합하여 기본 8196 클럭이고 클럭 주기 타겟을 줄일수록 critical path에 플립플롭(FF) 삽입하여 지연을 줄이도록 합성하므로 1씩 증가한다. 따라서 필요한 전체 FF 개수는 약 4096개씩 증가하는 것을 확인할 수 있다. 입력 비트를 처리하는 데 걸린 시간은 클럭 주기와 latency를 곱한 값이므로 throughput을 계산하는 식은 아래 식 (7)과 같다.

$$Throughput = \frac{\text{처리 비트수}}{\text{클럭 주기} \times latency} \tag{7}$$

입력 4096 비트에 대한 throughput을 계산하면 각각 99.3Mbps, 180.4Mbps, 292.6Mbps이며 합성 시 클럭 주기 타겟 설정을 변경하여 쉽게 throughput을 높일 수 있다. 논리 게이트 수는 클럭 주기에 거의 영향을 받지 않으므로 필요한 Look-Up Table (LUT) 수는 거의 비슷하다. BRAM_18K는 블록 램으로 식 (6)의 행렬 W의 부행렬 B_{ij}의 첫 번째 행벡터 b_{ij} (Fig. 3의 C 소스 코드의 배열 b3로 별도의 include 파일에 선언됨)를 저장하는 Read Only Memory (ROM)을 구현하기 위해 필요하다. ROM에 저장된 값을 읽어 8개의 피드백 시프트 레지스터를 동시에 초기화해야 하기 때문에 한 개가 아닌 동일한 8개의 ROM으로 합성된다.

2.4 HLS를 이용한 부호기 설계 2

Figure 2의 부호기 구조는 throughput 향상을 목적으로 하기 때문에 피드백 시프트 레지스터, NAND, EX-OR 게이트, FF 레지스터로 구성된 동일한 구조

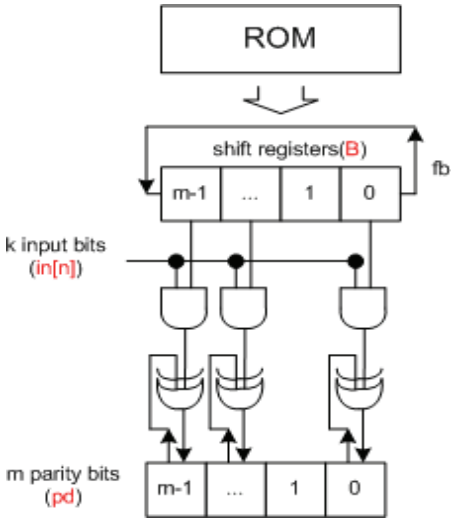

Fig. 4. simplified LDPC encoder structure

의 블록 8개가 동시에 구현된다. 한편 이를 Fig. 4와 같은 구조로 변경한 후 이를 순차적으로 8번 반복적으로 동작하게 하면 Table 4와 같이 throughput은 약 1/4로 감소하지만 대신 하드웨어 복잡도(또는 디바이스 이용률)도 약 1/8로 크게 감소시킬 수 있다. Throughput이 1/8이 아니라 1/4로 감소하는 이유는 Fig. 4의 구조는 Fig. 2의 구조와 비교하여 패리티 비트 계산에 필요한 latency는 8배로 증가하지만 계산된 패리티 비트를 출력하는데 필요한 latency는 동일하기 때문이다. Fig. 5는 이를 구현한 C 코드인데 Fig. 3의 C 코드와 비교하면 for 루프 l1이 추가되었고 대신 Fig. 3의 for 루프 l21, l22가 삭제되었다. HLS를 이용한 설계가 HDL 코드보다 복잡한 제어부를 고려할 필요가 없어 구조 변경이 매우 용이한 것을 알 수 있다.

Table 4. HLS Synthesis Results of C source code of Fig. 5

Target	10ns	5ns	2.5ns
Synthesized clock period	5.032ns	2.771ns	1.708ns
latency	36905	36921	36929
Throughput (Mbps)	22.1	40.0	64.9
BRAM_18K (240)	15 (6.25%)	15 (6.25%)	15 (6.25%)
FF (128000)	1072 (0.8%)	1738 (1.4%)	2251 (1.8%)
LUT (64000)	2804 (4.4%)	2890 (4.5%)	2890 (4.5%)

```
void encoder(ap_uint<1> in[4096], ap_uint<1> out[4096]) {
    int n, i, j, q, r;
    ap_uint<512> B, pd;
    ap_uint<1> fb;

    l1: for(r=0;r<8;r++) {
        l11: for(n=0;n<512;n++) pd[n] = 0; // UNROLL directive
        l12: for(n=0;n<4096;n++) { // PIPELINE directive
            if(n%512==0) { // register initialize
                q = n/512;
                B.range(511,0) = b3[8*q+r];
            }
            else { // register circular shift-right
                fb = B[0];
                B >>= 1;
                B[511] = fb;
            }
            l121: for(j=511;j>-1;j--) // UNROLL directive
                pd[j] = pd[j] xor (in[n] and B[j]);
        }
        // PIPELINE directive
        l13: for(n=0;n<512;n++) out[512*r+n] = pd[511-n];
    }
}
```

Fig. 5. Modified HLS C source code for simplified structure of Fig. 4

III. 결 론

본 논문에서는 Xilinx사의 Vivado HLS를 이용하여 C 언어로 텔레메트리 표준 106-17 LDPC 부호기를 throughput 향상과 하드웨어 복잡도 감소의 두 가지 목적으로 설계하였고 각각을 Spartan7 xc7s100 디바이스를 타겟으로 합성한 결과를 비교하였다. 전자는 후자에 비해 약 4배 정도의 throughput이 높고, 후자는 전자에 비해 약 1/8의 하드웨어 복잡도가 감소하였다. HLS는 C 언어로 기술된 소스 코드로부터 자동으로 HDL 코드를 생성하므로 C 언어 소스 코드를 일부 수정하여 하드웨어 구조를 쉽게 변경할 수 있음을 확인하였다.

References

1) Gallager, R. G., “Low-density parity-check codes,” *IRE Trans. Inf. Theory*, Vol. 8, pp. 21-28, January 1962.

2) *ETSI EN 302 755 V1.3.1*, April, 2012.

3) *ETSI EN 302 307 V1.1.2*, June, 2006.

4) *3GPP TS 36.201 V10.0.0*, December, 2010.

5) CCSDS *131.1-O-2 Experimental Specification*, September 2007.

6) Telemetry Standards, *RCC Standard 106-17*, July 2017.

7) Vivado HLS optimization methodology guide, *UG1270(v20174)*, December 2017.