JKIICE Journal of the Korea Institute of Information and Communication Engineering

한국정보통신학회논문지 Vol. 24, No. 9: 1180~1186, Sep. 2020

병렬화된 에러 보정 코드 모듈 기반 프로세서 속도 및 신뢰도 향상

강명진[1] · 박대진[2*]

High Speed and Robust Processor based on Parallelized Error Correcting Code Module

Myeong-jin Kang[1] · Daejin Park[2*]

[1]Ph.D. student, School of Electronics Engineering, Kyungpook National University, Daegu, 41566 Korea
[2*]Professor, School of Electronics Engineering, Kyungpook National University, Daegu, 41566 Korea

요 약

임베디드 시스템 중 하나인 TPU (Tiny Processing Unit)를 사용하는 데에는 많은 제약들이 따른다. 외부 충격에 의해 데이터 통신 중 잡음이 발생하거나, 충분한 전력이 공급되지 않아 문턱전압을 넘지 못해 올바른 값 전달이 이루어지지 않는 경우가 있다. 이러한 문제점들을 해결하기 위해 많은 임베디드 시스템에서는 ECC (Error Correcting Code)를 사용하는데, ECC를 추가하게 되면서 메모리에서 데이터를 읽어오는 시간이 더 오래 걸리게 되는 문제점이 발생한다. 따라서 우리는 ECC 처리된 코드를 읽어오는 과정을 병렬처리하여 병목현상을 완화하고 TPU의 속도 및 데이터 안정성을 높이는 모델을 제안한다. 제안된 구조는 기존 구조에 비해 메모리를 조금 더 사용하여 안정성과 더 빠른 속도를 보여준다. 실험은 행렬의 연산을 사용하여 진행되었으며, 제안된 구조는 이전의 구조보다 7% 빠른 속도를 보여준다.

ABSTRACT

One of the Embedded systems Tiny Processing Unit (TPU) usually acts in harsh environments like external shock or insufficient power. In these cases, data could be polluted, and cause critical problems. As a solution to data pollution, many embedded systems are using Error Correcting Code (ECC) to protect and restore data. However, ECC processing in TPU increases the overall processing time by increasing the time of instruction fetch which is the bottleneck. In this paper, we propose an architecture of parallelized ECC block to the reduce bottleneck of TPU. The proposed architecture results in the reduction of time 10% compared to the original model, although memory usage increased slightly. The test is evaluated with a matrix product that has various instructions. TPU with proposed parallelized ECC block shows 7% faster than the original TPU with ECC and was able to perform the proposed test accurately.

키워드 : ECC병렬 처리, 정확도 향상, 병목현상 제거, 임베디드 시스템

Keywords : Parallelized ECC block, Accuray improvement, Releasing bottleneck, Embedded system

Received 17 June 2020, **Revised** 24 June 2020, **Accepted** 17 July 2020

*** Corresponding Author** Dae-Jin Park(E-mail:boltanut@knu.ac.kr, Tel:+82-053-950-5548)
Professor, School of Electronics Engineering, Kyungpook National University, Daegu, 41566 Korea

Open Access http://doi.org/10.6109/jkiice.2020.24.9.1180 print ISSN: 2234-4772 online ISSN: 2288-4165

Ⅰ. 서론 및 관련 연구

임베디드 시스템은 특성상 일정하고 충분한 전력이 공급되지 않거나 외부의 충격에 의해 신호가 흔들리는 등 여러 쾌적하지 못한 상황에서 자주 동작된다. 이러한 상황들을 데이터에 손상을 줄 수 있고, 임베디드 시스템의 동작에서 1-bit의 작은 오류도 동작에 큰 영향을 끼칠 수 있다. 이를 방지하기 위해 여러 임베디드 시스템에서 ECC (Error Correcting Code)를 사용하여 데이터를 보호한다.

데이터는 0과 1로 이루어져 있다고 하지만 실제로 데이터를 인식하는 장치에선 0으로 설정한 값에 가까운지 1로 설정한 값에 가까운지를 판단하여 0과 1을 인식하고 구분한다. 데이터 통신이 이루어질 때, 송신부에서는 계속하여 0과 1에 가까운 값을 내보내고 수신부에서 그 값을 받아 어디에 더 가까운지를 판단하여 데이터를 읽는 것이다. 하지만 서론에서 말했듯이 임베디드 시스템은 외부 환경에 영향을 받고 쉽게 노이즈나 외부 충격에 노출되어 있다[1].

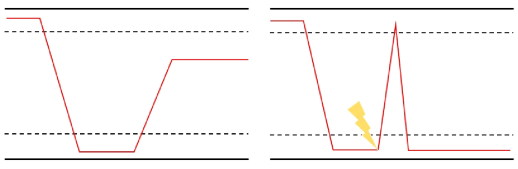


Fig. 1 (a) insufficient power environment
(b) noise caused byexternal shock

그림 1과 같은 경우에 우리는 정확한 값을 읽을 수 없게 된다. 이는 임베디드 시스템의 신뢰도가 하락하게 되고[2], 이때 우리는 잘못 수신된 데이터를 바로잡을 방법이 필요하고, 이를 위한 해결책으로 ECC를 사용한다[3].

ECC란 추가적인 데이터를 사용하여 데이터의 손상을 감지하고, 가능한 범위에서 데이터를 복원까지 시킬 수 있는 방법이다. 이는 소프트웨어나 하드웨어를 통해 여러 방법으로 ECC 알고리즘들을 구현할 수 있다.

ECC에서 오류를 감지하고 고치기 위한 여러 방법들이 있다. 오류를 감지만 하는 패리티 체크 코드, 1-bit의 오류를 고칠 수 있고, 2개의 오류까지 찾아낼 수 있는 single error correcting double error detecting (SECDED) 해밍 코드, 여러 번 전송하여, 가장 많은 빈도 수의 데이터를 사용하는 방법이 있다. 많은 임베디드 시스템에서는 메모리와 통신에 걸리는 부하를 생각하여 해밍 코드를 많이 사용한다[4].

임베디드 시스템에서 사용되는 대표적인 ECC인 해밍 코드 는 추가적인 패리티를 사용하여 오류를 찾아내고 수정하는데, 그림 2에서 tiny processing unit (TPU)의 ECC 구동 개념을 보여준다.

필요한 패리티 비트의 수는 전송하는 데이터의 크기에 비례한다. d_n 개의 데이터를 해밍 코드를 사용하기 위해 필요한 패리티 비트의 수 p_n 는 다음의 조건식 (1) 을 사용하여 나타낼 수 있다.

$$2^{p_n} \geq d_n + p_n + 1 \tag{1}$$

따라서 16-bits의 데이터 전송을 위해선 5-bits의 패리티 비트가 필요하고 7-bits의 데이터 전송을 위해선 4-bits 만큼의 패리티 비트가 필요하다. 해밍 코드에서 패리티 비트는 각각 $2^n (n \geq 0)$ 의 위치에 들어가게 된다. 자리에 들어간 패리티 비트는 자신의 위치만큼 읽어 exclusive or 연산 후, 건너뛰기를 반복한다. (7,4) 해밍 코드의 예시를 보면 다음과 같다.

$$\begin{aligned} p_0 &= d_0 \hat{} d_1 \hat{} d_3 \\ p_1 &= d_0 \hat{} d_2 \hat{} d_3 \\ p_2 &= d_1 \hat{} d_2 \hat{} d_3 \end{aligned} \tag{2}$$

식 (2)는 d_0 부터 d_3 까지 총 4개의 이진 데이터를 해밍 코드로 담아 보내기 위한 연산이다. 이렇게 만들어진 코

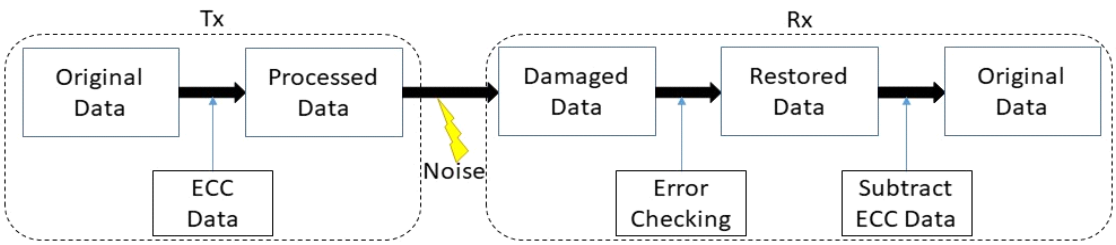


Fig. 2 Conceptualization of ECC processing

드는 해밍 디코드를 하여 원본 데이터를 뽑아내고 사용한다. 해밍 디코드를 위해선 수신부에서 신드롬을 제작한다. 이때의 신드롬을 제작하기 위해서 송신부에서는 수신부와 같이 데이터의 위치를 사용하여 또 하나의 패리티 비트를 만든다. 만들어진 패리티 비트와 전송받은 패리티 비트를 exclusive or 연산하여 신드롬을 제작한다.

$$\begin{aligned} p'_0 &= d_0 \wedge d_1 \wedge d_3 \\ p'_1 &= d_0 \wedge d_2 \wedge d_3 \\ p'_2 &= d_1 \wedge d_2 \wedge d_3 \\ S_0 &= p_0 \wedge p_0' \\ S_1 &= p_1 \wedge p_1' \\ S_2 &= p_2 \wedge p_2' \end{aligned} \tag{3}$$

식 3의 방법으로 제작된 신드롬은 오류가 발생한 비트의 순번을 가리키게 된다. 신드롬의 크기가 0인 경우 오류가 발생하지 않은 것이고, $2^n (n \geq 0)$ 번째를 가리키게 되면 패리티 비트의 오류, 이외의 수는 데이터의 오류를 알린다.

Hamming ECC를 TPU의 동작에 추가하게 되면 ECC를 추가하여 저장하는 부분은 메모리에 데이터를 쓰는 부분에 추가될 것이고, ECC를 디코드하여 원본의 데이터를 뽑아내는 과정을 메모리에서 데이터를 읽어오는 부분에 추가될 것이다.

Table. 1 Syndrome

Syndrome			Error Pattern							Error
0	0	0	0	0	0	0	0	0	0	NO
0	0	1	0	0	0	0	0	0	1	7
0	1	0	0	0	0	0	0	1	0	6
1	0	0	0	0	0	0	1	0	0	5
0	1	1	0	0	0	1	0	0	0	4
1	0	1	1	0	0	0	0	0	0	1
1	1	0	0	0	1	0	0	0	0	3
1	1	1	0	1	0	0	0	0	0	2

식 2에 의해 코드로 만들어진 데이터는 메모리에 저장되게 되고, TPU에서 실행되기 위해선 ECC 메모리에서 데이터를 읽고 해석과정을 통하여 원본 데이터를 뽑아내 처리 후, 다시 결과를 ECC를 통하여 보호하는 과정을 반복한다. 이때, 문제가 되는 점은 ECC 메모리에서 원본 데이터를 추출하는 과정이 레지스터나 ALU에 데이터가 들어가기 전에 디코드가 완료되어야 정상적인 동작이 가능하다는 점이다. 일반적 TPU 구조에서 가장 문제가 되는 점 중 하나는 메모리에서 명령어를 가져오는 시간이 길어 병목현상이 나타난다는 것이다[5].

TPU에서 빠른 속도로 동작을 수행하기 위해선 병목현상을 줄이는 것이 중요한데[6], ECC 처리를 추가하게 된다면 명령어를 읽어 오는 과정에 더 많은 시간이 소모되며 이는 TPU의 동작시간에 직접적인 영향을 주게 된다. 따라서 ECC의 디코드를 어떻게 하느냐가 TPU의 성능에 영향을 끼치게 된다[7].

본 논문에서는 해밍 코드 기반 TPU 디코더의 병목현상을 줄이기 위해 ECC 디코더를 여러 부분으로 나누어 원본 데이터를 부분적으로 병렬처리하는 기법을 제안하고[8] 추가되는 패리티 비트로 인한 안정성을 분석한다.

본 논문의 구성은 다음과 같다. Ⅱ장에서는 문제를 정의 과정에 대해 서술하고 이를 해결하기 위한 가정 내용을 언급하며, Ⅲ장에서는 기존의 ECC 디코드 블록과 제안된 병렬처리형 ECC 디코드 블록의 차이를 살펴본다. Ⅳ장에서는 제안된 TPU에 적용하였을 때 두 블록으로 인해 생기는 차이를 시뮬레이션 결과를 통해 알아보고, 타당성을 입증하며, 마지막으로 Ⅴ장에서 결론을 맺는다.

Ⅱ. 문제정의 및 가정

2.1. 문제 정의

TPU는 충분한 전력이 공급되지 않거나 외부의 충격이 있는 외부 환경에서 작동될 때, 데이터 신호가 흔들리는 현상으로 인해 데이터가 손상되는 현상이 자주 발생하게 된다. 이때, 데이터들 사이에 손실되거나 손상된 데이터를 오류를 감지하거나 가능하다면 복구까지 가능한 ECC 데이터를 추가하여 TPU 동작의 신뢰성을 향상시킬 수 있다.

하지만 TPU는 기존의 폰 노이만 구조를 가지는 프로세서로 설계 구조상 메모리에서 데이터를 읽어 오는데 많은 시간을 소모하게 된다. 또한 메모리를 가져오는 구간에서 ECC 데이터를 디코드 하여 ALU로 넘겨주어야 하기 때문에 신뢰성 향상을 위해 추가한 ECC가 TPU의 상당한 성능 저하를 가져오게 된다.

메모리에서 지연되는 시간은 TPU가 파이프라인으로 동작될 때 더 많은 속도의 저하를 가져오게 되는데, 추가된 ECC 디코드 과정은 이 현상을 더욱 악화시킨다.

본 논문은 ECC 기반의 TPU에서 ECC 디코드를 병렬적으로 처리하는 방법을 제안함으로써 TPU의 병목현상을 감소시키는 방법을 제안한다.

2.2. 가정

TPU는 폰 노이만 구조를 가지며 폰 노이만 구조에서 메모리 병목현상은 자주 중요한 요인으로 제시되었다. ALU와 레지스터의 동작 속도에 비해 메모리의 속도가 느리므로 명령어를 읽어오고 디코드 하여 제어신호를 보내는데 많은 병목현상을 갖고 있고, 이를 해결하기 위해 외부 고속 메모리 코어를 사용하거나 메모리 셀에 프로세서를 통합하는 등 여러 연구가 이루어지고 있다 [9-10].

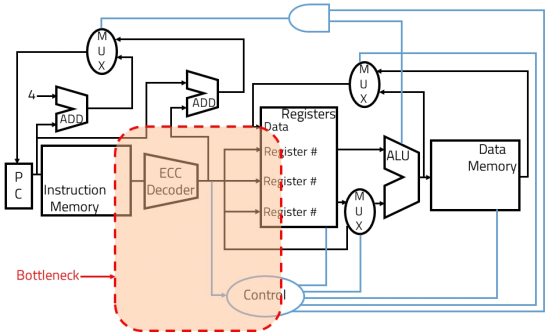


Fig. 3 Von Nuemann architecture's Bottleneck

본 논문에서는 가정된 TPU는 폰 노이만 구조를 기반으로 메모리에서 읽어온 데이터를 디코드 과정을 통해 ALU와 레지스터에 제어 신호를 보낸다. 이 과정을 파이프 라이닝을 통해 진행하게 되고, 고속화를 위해서 많은 부하가 걸리는 명령어를 읽어와 제어신호를 보내는 과정의 고속화가 필요하다. ECC 처리된 명령어를 읽어오는 것은 이 부분에서 더 많은 부하를 야기하므로 이 과정의 병렬화를 통해 ECC 처리된 명령어를 보다 빠르게 처리하여 제어신호를 보내는데 초점을 두고 있다.

Ⅲ. 제안 구조

그림 4의 (a) 와 (b)는 기본적인 ECC 디코더를 추가한 TPU의 구조와 ECC 병렬처리 디코더를 추가한 TPU 구조의 차이를 나타낸다. 기존 ECC의 디코더를 추가한 그림 4의 (a)는 레지스터와 메모리 사이에 위치한 하나의 커다란 ECC 디코더이다. 디코더 하나를 사용하여 ECC 데이터를 포함한 20-bit의 데이터를 분리하고 신드롬을 만들어 오류를 검출하고 수정하는 역할을 지속적으로 담당하고 있다.

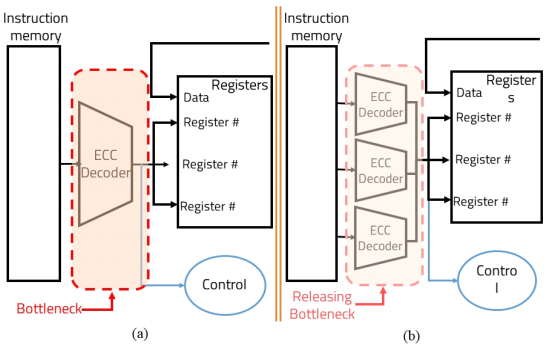


Fig. 4 Original model and proposed model

3.1. 제안된 ECC 디코드 과정

제안된 구조의 ECC 디코더는 그림 4 (b)와 같은 구조를 하고 있다. 명령어에는 opcode의 종류, 사용할 레지스터 등이 포함된 operand들이 포함되어 있다. 이는 ECC로 처리된 데이터를 디코더가 레지스터와 제어 모듈로 명령어가 들어가기 전에 원본 데이터를 추출하여야 한다는 것을 의미한다. 기존 모델의 경우 1개의 ECC 디코더가 동작하므로 원본의 데이터를 디코드하기 위해 오랜 시간이 걸리게 되고, 이는 TPU의 성능 저하로 이어진다. 반면 제안된 모델의 경우 명령어를 3개의 부분으로 나누어 각각을 병렬적으로 동시에 처리하여 병목현상을 줄이려 한다. 각 패리티가 들어가는 장소는 그림 5와 같다.

Fig. 5 Parity bit location

3.2. 제안된 구조의 장점

Opcode와 operand들이 각각 독립적으로 동작하게 되므로 제안된 구조에서의 ECC 디코더는 opcode만을 빠르게 디코딩하여 제어 모듈에 빠르게 신호를 보낼 수 있

다. opcode는 그림 4 (a)와 같은 커다란 모듈을 거쳐 제어 모듈로 가는 것이 아닌 병렬화된 하나의 작은 디코딩 모듈을 거쳐 제어 모듈로 가게 된다. 이는 이전 병목현상의 문제였던 명령어를 읽어와 제어 모듈까지 신호를 보내는 시간이 짧아짐을 의미한다. 이는 TPU의 병목현상을 줄을 수 있는 중요한 열쇠가 된다.

또한 그림 5에서와 같이 명령어를 나누게 되면서 더 많은 패리티 비트를 사용하여야 한다. 이는 명령어가 길어져 완벽한 데이터를 보낼 가능성이 급격히 낮아지는 현상을 방지한다. 짧은 데이터를 더욱 많은 패리티 비트를 사용하여 보호하기 때문에, TPU의 신뢰성이 더욱 향상된다.

3.3. 메모리 사용량과 회로 크기

제안된 모델에서는 그림 5에서 보이는 것처럼 opcode에 3개의 패리티 비트를, op1, op2에 각각 3개와 4개의 패리티 비트를 필요로 한다. 제안된 구조는 명령어를 여러 부분으로 나누어 동작하기 때문에, 명령어당 필요한 ECC의 데이터는 5-bit에서 10-bit로 늘어나게 된다. 또한 ECC를 검증하기 위한 3개의 디코드 모듈이 필요하게 된다. 이는 10개의 신드롬을 생성하여 ECC 디코드를 진행하기 때문에 회로의 크기 또한 커질 수 있다.

기존의 모델과 비교하여 보았을 때, 5개의 패리티가 필요한 것과 달리 제안된 모델에서는 하나의 명령어에 총 10개의 패리티 비트가 필요하게 되고 이로 인해 16-bit 데이터를 보내는데 21-bit, 26-bit가 필요하게 되면서 전송률이 80%에서 61%로 감소하게 된다. 하지만 많은 ECC 데이터의 사용은 ECC의 정확도와 신뢰도 증가로 이어진다.

Ⅳ. 시뮬레이션 및 결과

본 논문에서는 제안한 TPU 구조를 구현하여 실험 및 기본 구조와 비교 실험을 진행하였다. 실험을 위한 TPU 구조는 C++을 사용하여 초기 구현 후 Verilog를 이용하여 최종 구현하였고, 시뮬레이션의 경우 “SIMVISION”을 이용하였다[11]. 또한 작성한 디코더의 합성을 위해 “Design Compiler”를 사용하여 진행되었다.

시뮬레이션 결과는 TPU에서 여러 명령어들을 실행시켜 실험하기 위하여 3*3 행렬의 곱을 Binary 코드로 제작 후, 수행하였다. 각 ECC 디코딩 모듈이 장착된 TPU를 동일한 테스트 벤치를 수행하여 각 속도와 정확도를 비교하는 실험을 수행하였다.

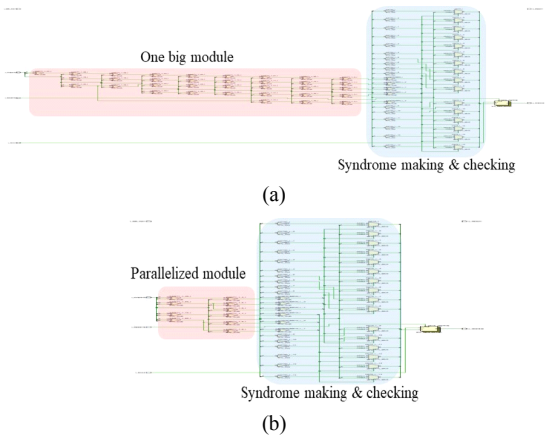


Fig. 6 Synthesiezed decoder

그림 6 (a)는 기존 모델의 합성 결과, (b)는 제안된 모델의 합성 결과를 보여준다. 합성 결과로 디코더 모듈이 병렬화되어 총면적이 늘어나게 되지만 그림 6의 (b)에서 볼 수 있듯이 병렬화된 모듈이 짧아져 더 짧은 기간에 ECC 디코드 처리가 가능하게 된다.

4.1. 속도 향상

실험 데이터는 행렬 곱셈을 위한 반복 문과 덧셈 곱셈이 섞인 테스트 벤치를 사용하였다. ECC를 적용하게 될 때, 기존의 모델은 ECC를 적용하지 않은 시스템에 비해 약 30%의 속도 감소를 보여준다. 이는 처리된 데이터를 ECC 처리하거나 ECC 처리된 명령어를 디코드 할 필요가 없기 때문이다.

하지만 제안된 ECC 병렬처리 디코더 모델은 기존 ECC 디코더와 비교하여 성능이 10% 증가하게 된다. 제안된 디코더의 추가는 TPU의 속도 저하를 최소화한다. 그림 7에서처럼 제안된 모델은 ECC를 병렬 처리 하지 않은 그림 6의 (a) 모델과 비교하여 7%의 속도가 증가를 보여준다. TPU의 경우 파이프라인으로 동작하기 때문에 모듈의 속도 향상 비율과 전체 시스템 속도 향상 비율이 다르게 측정된다.

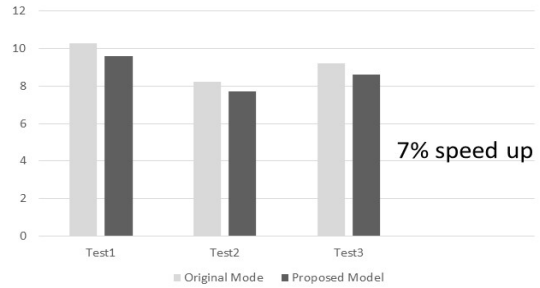


Fig. 7 Run time comparison

4.2. 정확도 향상

ECC는 TPU에서 데이터 전송이 높지 않을 때의 신뢰도를 높일 수 있다. 표 2는 각 전송 성공률에 따른 TPU의 신뢰도 향상을 보여준다. 제안된 모델은 기존 모델에서 추가적인 패리티 비트를 사용하게 되어 더욱 향상된 신뢰도를 보여준다.

Table. 2 Reliability according to transmission success rate

Transmission Success Rate / Model	95%	98%	99%
Original Model	0.810	0.960	0.989
Proposed Model	0.955	0.985	0.996
Rate of rise	17%	2%	0.7%

4.3. 메모리 사용량 비교

제안된 모델에서는 명령어를 3부분으로 나누어 ECC 코드를 만들고 저장하게 되므로 이전의 모델과 비교하여 더 많은 량의 패리티 비트가 필요하게 된다. 실제 16-bit의 TPU에선 기존 5-bits의 패리티 비트가 필요했던 것과 비교하여 제안된 모델에서는 10-bits가 필요하게 되고, 그로 인해 16-bit 데이터 전송에 필요한 데이터가 21-bit에서 26-bit로 늘어나게 되어 전송률이 80%에서 61%로 떨어지게 된다. 임베디드에서 따로 사용하는 ECC 메모리의 용량을 늘려 해결하여야 한다.

Ⅴ. 결 론

본 논문의 연구에서는 폰 노이만 구조의 TPU에서 해밍코드 ECC를 적용할 때 발생하는 병목현상의 최소화와 TPU의 신뢰도 향상을 위한 디코드 블록 병렬화 구조를 제안하였고, 해당 시스템을 구현하여 속도와 정확도가 향상되는 결과를 도출하였다.

제안된 모델의 실험 결과 회로의 크기와 메모리의 사용량이 23% 증가하지만, 7%의 속도 증가와 신뢰도 향상을 검증하였다.

하지만 속도 향상과 안정성의 향상에 따른 메모리 사용량의 증가와 회로의 복잡도 증가가 따라오게 되므로 이를 효과적으로 해결하기 위한 추가 연구가 필요하다.

제안된 모델은 임베디드 시스템에서 속도와 신뢰도를 동시에 높여 시스템이 동작하기 위한 전원이 일정하게 공급되지 않는 상황의 임베디드 시스템에 적용하여 향후 고속화 및 안정화가 필요한 임베디드 시스템에서 이용될 수 있다.

ACKNOWLEDGEMENT

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science and ICT (NRF-2019R1A2C2005099), and Ministry of Education (NRF-2018R1A6A1A03025109), and BK21 Plus project funded by the Ministry of Education, Korea (21A20131600011), and the EDA tool was supported by the IC Design Education Center(IDEC), Korea.

REFERENCES

[1] S. Saha, S. Ehsan, A. Stoica, R. Stolkin and K. McDonald-Maier, “Real-Time Application Processing for FPGA-Based Resilient Embedded Systems in Harsh Environments,” *2018 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, Edinburgh, pp. 299-304, 2018.

[2] J. Chen and M. Shafique, “Embedded software reliability for unreliable hardware,” *2014 International Conference on Embedded Software (EMSOFT)*, Jaypee Greens, pp. 1-1, 2014.

[3] D. Lee, M. Kang, P. Plesznik, J. Cho and D. Park, “Scrambling Technique of Instruction Power Consumption for Side-Channel Attack Protection,” *2020 International*

Conference on Electronics, Information, and Communication (ICEIC), Barcelona, Spain, pp. 1-2, 2020.

[4] G. Subhasri, and N. Radha, "VLSI design of Parity check Code with Hamming Code for Error Detection and Correction," *2019 International Conference on Intelligent Computing and Control Systems (ICCS)*, Madurai, India, pp. 15-20, 2019. doi: 10.1109/ICCS45141.2019.9065790.

[5] J. Edwards and S. O'Keefe, "Eager recirculating memory to alleviate the von Neumann Bottleneck," *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, Athens, pp. 1-5, 2016.

[6] M. Ramakrishnan and J. Harirajkumar, "Design of 8T ROM embedded SRAM using double wordline for low power high speed application," *2016 International Conference on Communication and Signal Processing (ICCSP), Melmaruvathur*, pp. 0921-0925, 2016.

[7] M. Sever and E. Çavus, "Parallelizing LDPC Decoding Using OpenMP on Multicore Digital Signal Processors," *2016 45th International Conference on Parallel Processing Workshops (ICPPW)*, Philadelphia, PA, pp. 46-51, 2016.

[8] D. Park and T. G. Kim, "Safe microcontrollers with error protection encoder-decoder using bit-inversion techniques for on-chip flash integrity verification," *2013 IEEE 2nd Global Conference on Consumer Electronics (GCCE)*, Tokyo, pp. 299-300, 2013.

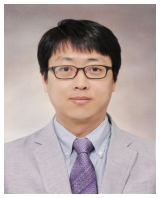
[9] L. Azriel, A. Mendelson and U. Weiser, "Peripheral memory: a technique for fighting memory bandwidth bottleneck," in *IEEE Computer Architecture Letters*, vol. 14, no. 1, pp. 54-57, Jan.-Jun. 2015.

[10] J. Kim, J. Cho and D. Park, "Low-Power Command Protection Using SHA-CRC Inversion-Based Scrambling Technique for CAN-Integrated Automotive Controllers," *2018 IEEE Conference on Dependable and Secure Computing (DSC)*, Kaohsiung, Taiwan, pp. 1-2, 2018.

[11] A. Kafi, A. Rahman, B. Mahjabeen and M. Rahman, "An efficient design of FSM based 32-bit unsigned high-speed pipelined multiplier using Verilog HDL," *8th International Conference on Electrical and Computer Engineering*, Dhaka, pp. 164-167, 2014.

강명진(Myeongjin Kang)

2020년 2월 : 경북대학교 전자공학부 졸업
2020년 3월 ~ 현재 : 경북대학교 전자공학부 석 · 박사 통합과정
※관심분야 : 임베디드 시스템, 저전력 구동, 사물 인터넷 네트워크, Robust 임베디드 시스템 등



박대진(Daejin Park)

2001 경북대학교 전자전기공학부 학사
2003 한국과학기술원(KAIST) 전기 및 전자공학과 석사
2014 한국과학기술원(KAIST) 전기 및 전자공학과 박사
2003 ~ 2014 SK Hynix/Samsung (차세대 LSI 설계) 수석연구원
2014 ~ 2016 경북대학교 전자공학부 초빙교수 (2014년 대통령 Postdoctoral Fellow 선정)
2016 ~ 현재 경북대학교 전자공학부 조교수
※관심분야 : 저전력 SoC 설계, 하드웨어-소프트웨어 Co-design, Dependable 스마트 IoT 시스템, Robust 임베디드 시스템