개선된 수정 유클리드 알고리듬을 이용한 고속의 Reed-Solomon 복호기의 설계

論 文
48A - 7 - 13

Implementation of High-Speed Reed-Solomon Decoder Using the Modified Euclid's Algorithm

金 東 淳* · 崔 鍾 燦** · 鄭 德 鎭***
(Dong-Sun Kim · Jong-Chan Choi · Duck-Jin Chung)

Abstract - In this paper, we propose an efficient VLSI architecture of Reed-Solomon(RS) decoder. To improve the speed, we develope an architecture featuring parallel and pipelined processing. To implement the parallel and pipelined processing architecture, we analyze the RS decoding algorithm and the honor's algorithm for parallel processing and we also modified the Euclid's algorithm to apply the efficient parallel structure in RS decoder. To show the proposed architecture, the performance of the proposed RS decoder is compared to Shao's and we obtain the 10 % efficiency in area and three times faster in speed when it's compared to Shao's time domain decoder. In addition, we implemented the proposed RS decoder with Altera FPGA Flex10K-50.

Key Words : FEC, Reed-Solomon, Modified Euclid's Algorithm, Chien search, Forney syndrome

1. 서 론

오늘날의 통신분야에는 목적과 용도에 따라 문자, 데이터, 음성 및 영상 등의 다양하고 방대한 정보원들이 사용되고 있다. 우리가 흔히 사용하는 라디오, 텔레비전, 휴대용 전화기 및 CD-ROM등이 그 좋은 예라 할 수 있으며, 점차 증가하고 있는 정보원들을 보다 빠르고 정확하게 전송하기 위한 알고리즘 및 하드웨어의 개발이 중요하게 대두되고 있다. 또한 전송하는 데이터의 신뢰도, 보안성 및 정보의 질적인 향상을 위해 디지털방식을 이용한 통신시스템이 제안되었고, 이러한 디지털 통신방식은 기존의 아날로그 방식에 비해 많은 정보를 고속으로 전달할 수 있게 되었다. 이와 더불어 정보의 효율적인 전달을 위해 다양한 전송매체를 이용하게 되었으며, 이러한 전송매체는 전달하는 정보의 오류를 증가시키는 여러가지 전파 특성을 가지고 있어, 통신 시스템 상에서 저장 또는 전송하고자하는 정보의 신뢰도를 높이기 위해서 발생하는 오류를 정정해야하는 문제점을 가지고 있다[1]. 따라서, 본 논문에서는 정보 전달과정에서 발생할 수 있는 산발오류 및 군집오류 등에 의한 송신측 정보의 왜곡을 막고 이를 정확히 수신측에 전달하는 방법으로 사용되는 Reed-Solomon(RS) 오류정정부호의 복호기를 구현 및 검증하였다.

송신되는 전체 부호어를 N, 정보어를 I 라 하면 (N, I) RS부호라 표현하고, 이때의 검사 심볼은 $N-I$ 가 되며, 오류 정정능력은 $t=\frac{N-I}{2}$ 가 된다[2]. 오류 정정을 위한 RS 부호의 복호 과정은 많은 연산량과 하드웨어를 필요로 하며, 그 중에서 오류위치 다항식을 푸는 과정이 전체 복호기의 성능을 결정하는 중요한 부분으로 key equation이라고도 부른다. 오류위치 다항식을 푸는 방법으로는 크게 행렬을 이용한 Peterson-Gorenstein-Zeirler 알고리즘과 Berlekamp-Massey 알고리즘 및 Euclid 알고리즘으로 나눌 수 있다. 이 중에서 Peterson-Gorenstein-Zeirler 알고리즘의 경우는 패리티를 이용한 행렬연산을 이용한 방법으로 복호시 많은 나눗셈 과정이 필요하게 되어 하드웨어 구현에 많은 어려움이 있고[2], Berlekamp-Massey 알고리즘 역시 유한 필드 상에서 역수를 구해야 하는 과정이 한 심볼 클럭 안에 이루어 져야 하며, 이러한 유한체 내에서 역수를 구하는 과정은 곱셈보다 지연시간이 길어 전체 복호기의 성능을 결정하는 중요한 부분을 차지하게 된다. 따라서, 고속의 역수 계산을 위해 Look-Up Table(LUT)가 사용되어지고, 이러한 LUT의 경우 ROM을 이용해 구현되어지기 때문에, 다른 셀 보다 많은 하드웨어 크기를 차지하게 된다. 또한, 오류위치 다항식을 구하기 위해서는 계산된 오류값을 가지고, 각 레지스터의 계수들을 순간 순간마다 갱신해야 하는데 이러한 구조는 한 심벌 클럭 안에 레지스터의 갱신이 어렵다는 단점이 있다[3]. 본 논문에서는 수정 유클리드 알고리즘을 사용하여 이러한 문제점을 해결하였으며 기존의 제안되었던 수정 유클리드 연산부를 개선함으로써 전체 복호기의 성능을 향상시킬 수 있었다. 구현된 RS 복호기의 블록도는 그림 1과 같으며, 이의 기술을 위해 2장에서는 각 연산부에 대한 알고리즘 및 하드웨어 구현에 대해 설명하며 3장에서는 구현된 복호기의 구현 및 검증에 대해 기술하고 결론을 맺기로 한다.

* 正 會 員 : 電子部品硏究院 硏究員
** 正 會 員 : 電子部品硏究院 先任硏究員
*** 正 會 員 : 仁荷大 電子材料工學科 敎授 · 工博
接受日字 : 1999年 4月 6日
最終完了 : 1999年 6月 10日

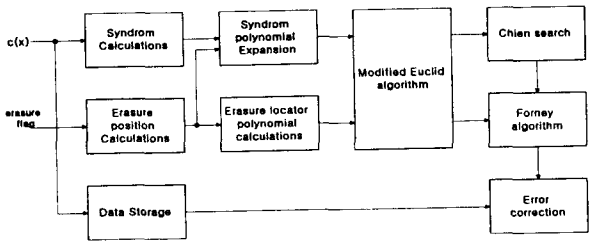


그림 1 RS 복호기의 블럭도
Fig. 1 Block diagram of RS Decoder

2. 본 론

2.1 Reed-Solomon 복호기의 구조

본 논문에서 제안한 RS 복호기의 구조는 크게 8개의 연산 블록으로 구성되어 있으며, 각 연산 블록은 적용되는 알고리즘과 구조에 따라 연산하는데 소요되는 지연시간이 달라지게 된다. 이러한 연산블럭의 특징은 전체 복호기의 동시 실행성에 중요한 영향을 미치는 요소가 되며, 전체 RS 복호기의 연산의 효율성과도 밀접한 관계가 있다. 본 논문에서는 고속의 데이터 처리가 가능한 RS 복호기의 구현을 위해 병렬구조를 사용하였으며, 각 연산블럭간의 병목현상을 파이프라인구조를 사용하여 해결하였다. 일반적인 병렬구조의 경우 전체 연산시간을 병렬처리의 작업량만큼 빠르게 처리 할 수 있는 반면에 병렬처리 하는 작업 수만큼 하드웨어의 크기가 증가하고, 병렬처리 된 결과를 다시 최종 결과 값으로 변환하는 부가의 하드웨어 역시 필요하게 되며 이는 전체 회로의 집접화를 어렵게 하는 원인이 된다. 따라서, 본 논문에서는 각각의 연산블럭의 지연시간을 고려한 신호의존도(Signal Dependency Diagram:SDD)를 사용하여 병렬구조를 효과적으로 적용할 수 있었다. 본 논문에서 구현한 RS 복호기는 표 1과 같이 크게 8부분의 연산블럭으로 분리할 수 있으며, 이는 다시 동시성을 가지는 3개의 병렬그룹과 2개의 연산블럭으로 나눌 수 있다.

표 1 RS복호기의 블록별 상태도
Table 1 State diagram of RS Decoder

연산블럭		각 연산블럭별 함수
S1	병렬그룹1	Si ← SYNDROME(DATA)
S2		ei ← α-Generation(Flag)
S3	병렬그룹2	Ti ← SYNDROME_EXPANSION(Si,ei)
S4		Λ_i ← ERASURE_EXPANSION(ei)
S5		ω_i , σ_i← Modified_Euclide(Ti, Λ_i)
S6	병렬그룹3	Ci ← CHIEN_SEARCH(ω_i , σ_i)
S7		Fi ← FORNEY_SEARCH(ω_i , σ_i)
S8		E'i ← DIVIDE(Ci ,Fi)

표 1을 기준으로 전체 신호의존도를 나타내는 SDD를 작성하면 그림 2와 같이 나타낼 수 있다. 여기서 (S_1,S_2), (S_3,S_4), (S_6,S_7)의 블록이 각각 독립적인 병렬 그룹을 형성하고 있음을 알 수 있다. 이러한 병렬 그룹을 형성하는 블록들은 그룹내의 다른 블록과는 독립적으로 연산이 가능하게 된다. 하지만, S_3은 (S_1,S_2)와 S_5는 (S_3,S_4), S_8은 (S_6,S_7)과 각각 신호의존도가 존재하게 되므로 같은 병렬 그룹내의 블록들은 이러한 신호의존도를 고려해서 동일한 지연시간을 갖도록 설계하여야 한다. 따라서, 본 논문에서는 각 블록의 동시 실행성과 효율성의 증대를 위해 동일한 성능을 유지하는 한도 내에서 병렬처리를 실행하도록 설계하여 서로 다른 지연시간으로 인한 별도의 지연 소자의 사용을 없애도록 하였다.

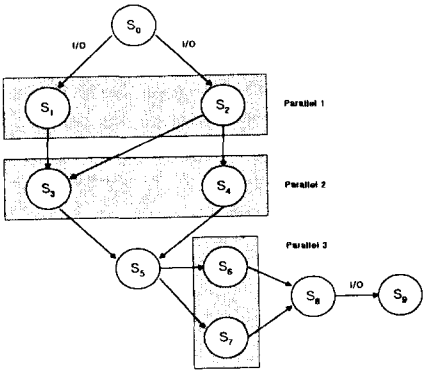


그림 2 RS 복호기의 SDD
Fig. 2 SDD of RS Decoder

또한, 각 병렬그룹내의 연산시간이 모두 다르게 되므로, 연속적인 코드워드를 입력받기 위해 본 논문에서는 파이프라인구조를 적용하여 설계하였으며, 이러한 파이프라인구조는 코드워드의 길이 및 하드웨어의 지연시간에 따라 다른 스테이지를 가지게 되므로 효율적인 파이프라인구조의 적용을 위해 표 2의 지연시간을 고려한 단위 시간별 연산블럭의 신호 흐름도를 이용하여 파이프라인구조를 분석하였다.

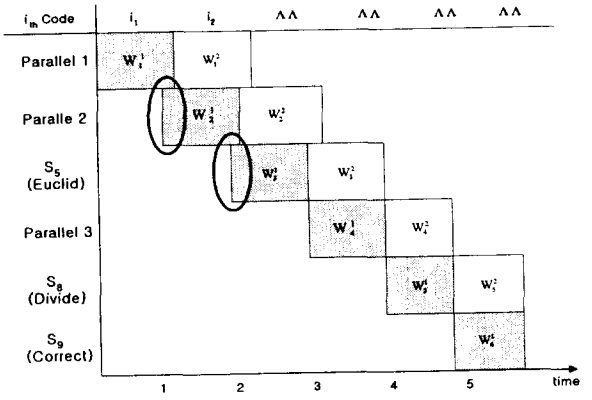


그림 3 파이프라인구조를 위한 단위시간별 신호 흐름도
Fig. 3 Space-time diagram for efficient pipeline structure

위에서 제시한 3개의 병렬 그룹과 2개의 연산 블록의 지연시간을 고려하여 각 연산 블럭들간의 겹침 동작이 생기는 부분을 파이프라인 처리함으로써 하드웨어의 추가 및 최초

연산 시간을 줄이고, 연속적인 복호가 가능하도록 설계하였다. 즉, 그림3과 표2에서의 지연시간을 이용하여 프로세서의 충돌이 일어나는 병렬그룹 1과 2, $2(N-1)>N$ 인 코드의 경우에는 추가로 병렬그룹2와 수정 유클리드 연산부분사이에 pipeline 구조를 적용하여 연산의 동시성을 고려한 고속의 RS 복호기를 구현할 수 있다.

표 2 파이프라인구조 적용을 위한 지연시간 분석
Table 2 Delay time analysis for pipelined structure

연산 블록	연산블록별 지연시간
병렬그룹 1	$\lfloor \frac{N}{2} \rfloor$
병렬그룹 2	N
S5	$2(N-1)$
병렬그룹 3	$(N-1)$

* $\lfloor N \rfloor$ 는 N을 넘지 않는 최대 정수

2.2 신디롬 계산 부 및 α^k 생성 부의 설계

RS 부호의 복호 원리는 송신단에서 코드워드가 생성다항식 g(x)를 코드워드로 나눈 나머지를 잉여항으로 더해 생성되는 것을 이용하여 이루어진다[2]. 이러한 복호를 신디롬 기반의 복호라 하고, 이를 위해 생성다항식 g(x)의 근을 실제 수신된 코드워드에 대입함으로써 수신된 코드워드의 이상유무를 판단하는 신디롬을 계산하게 된다. 그 원리는 실제 수신된 코드워드는 $C(x)=x^{n-k}m(x)+R(x)$ 와 같이 나타낼 수 있고[2], 여기에 생성다항식의 근을 대입하게 되면, $C(\alpha^i)=0$ 임으로, $R(\alpha^i)=e(\alpha^i)$ 가 된다. 즉, 수신된 다항식에 생성다항식의 근을 대입한 후의 값이 신디롬의 값이 되며, 이를 하드웨어로 구현하기 위해 식 (1)과 같은 Honor's rule을 적용하였다[3][4].

$$s_i=[(r_{n-1}x+r_{n-2})x+\cdots+r_1]x+r_0 \tag{1}$$

위의 수식을 보면 실제 수신되는 다항식은 코드워드의 길이 N개만큼의 지연시간을 가지므로, 실제 클럭만 최대 N clock을 소요하게 된다. 본 논문에서는 Honor's rule을 수정하여 병렬처리구조를 적용하여 실제 지연시간을 반으로 줄일 수 있었다. 이를 위해 수정된 Honor's rule를 식(2)와 같이 두 개의 항으로 나누었으며, 수정된 구조는 그림 4와 같다.

$$\begin{aligned} s_1&=[((r_{n-1}x^2+r_{n-3})x^2+\cdots+r_3)x^2+r_1]x \\ s_2&=[((r_{n-2}x^2+r_{n-4})x^2+\cdots+r_2)x^2+r_0] \end{aligned} \tag{2}$$

본 논문에서 설계한 RS 복호기는 연판정(Soft Decision)을 위해 삭제신호를 받아들여 삭제다항식을 계산한다. 이러한 삭제다항식을 계산하기 위해서는 삭제되는 신호가 입력되었을 때 그 값을 기억해야 하므로, 신드롬의 계산을 위해 입력되는 코드워드와 동기 되어 값을 계산하는 α^k 생성부를 두어 삭제 신호가 입력되면 이 때에 계산된 값을 다음 블록의 삭제다항식 계산에 사용하도록 설계하였다.

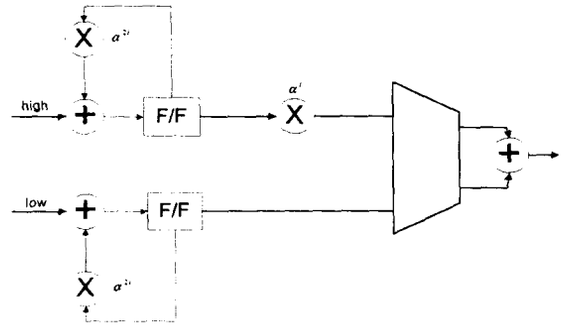


그림 4 신디롬 계산회로
Fig. 4 Syndrome Calculation circuit

본 논문에서는 신디롬 다항식의 계산을 병렬 처리함으로써 연산시간이 2배 빨라졌으므로, α^k 생성부도 동일한 성능을 유지하도록 설계되어야 한다. 따라서, 본 논문에서 제시한 구조는 동시에 두 개의 삭제신호를 입력받을 수 있도록 하였으며, 이의 블록선도는 그림 5와 같다.

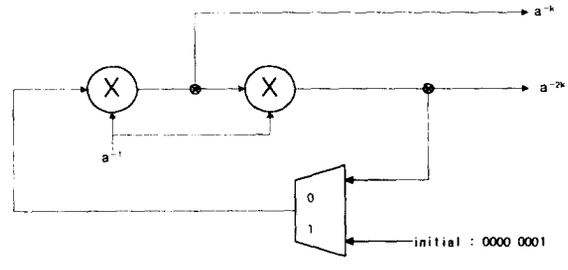


그림 5 수정된 α^k 생성부
Fig 5 Modified α^k Generation Block

2.3 Forney 신디롬 및 삭제다항식 확장부의 설계

입력된 삭제다항식의 근과 신디롬의 값을 이용해 새로운 다항식을 생성하는 부분으로 식3과 같은 과정을 통해 계산된다[5][6].

$$\begin{aligned} T(x) &= S(x)\cdot\lambda(x) \bmod x^{2t} \\ &= S(x)\prod_{j=1}^{v}(x-\alpha^{-i_j}) \bmod x^{2t} \end{aligned} \tag{3}$$

이는 반복적 구조를 이용하여 구현할 수 있으며, 수식적으로 표현하면 $T_{i_j}(x)=S(x)(x-\alpha^{-i_j})=S(x)x-$

$S(x)\alpha^{-i_j}$ 과 같이 나타낼 수 있다. 즉, 단순히 신디롬 다항식을 한 차수 이동시킨 후 신디롬 다항식과 삭제다항식의 위치 값을 곱한 다음 더하는 과정을 삭제신호가 발생한 수만큼 반복함으로써 구할 수 있고, x^{2t} mod 연산은 2t이상의 항을 제거한 후 각 다항식을 한 차수 이동시키는 과정을 통해 구현할 수 있다. 본 논문에서는 병렬구조를 적용하여 다항식 전개부분을 설계하였으며, 초기에 신드롬 값이 계산되면 삭제되는 수신 부호의 값을 순차적으로 입력받아 다항식 전개를 하게 된다. 이의 기본적인 블록 도는 그림 6과 같다.

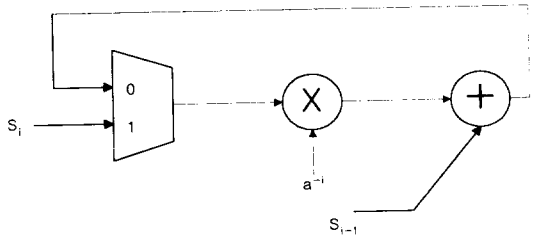


그림 6 수정된 신디롬 다항식 확장
Fig. 6 Modified syndrome expansion circuit

삭제다항식은 입력된 삭제 부호를 이용해 다항식을 형성하는 부분으로, 삭제 위치 다항식은 $\lambda(x)=\prod_{j=1}^{t}(x-\alpha^{-i_j})$ 로 표현되며, 이러한 다항식의 확장은 $(x+\alpha^{-i})\cdot\Lambda_{i-1}(x)=x\cdot\Lambda_{i-1}(x)+\alpha^{-i}\cdot\Lambda_{i-1}(x)$ 로 표현 할 수 있다. 삭제 위치 다항식은 수정 신디롬 다항식의 계산과 같은 방법으로 반복적인 하드웨어 구조를 이용하여 설계가 가능하다. 예를 들어 4개의 임의의 원소를 통해 다항식을 계산하는 경우 삭제 다항식은 다음과 같은 식4와 같이 표현 할 수 있으며, 이는 각각의 원소의 순차적인 입력으로 계산된다. 삭제부호의 수가 4개일 경우 이의 clock(N)에 따른 register(R)의 값은 표3 과 같으며, 기본 셀은 그림 7과 같다.

$$\begin{aligned}\lambda(x) &= (x+\alpha)(x+\beta)(x+\gamma)(x+\delta)\\ &= x^4+(\alpha+\beta+\gamma+\delta)x^3\\ &+(\alpha\beta+\alpha\gamma+\alpha\delta+\beta\gamma+\beta\delta+\gamma\delta)x^2\\ &+(\alpha\beta\gamma+\alpha\beta\delta+\alpha\gamma\delta+\beta\gamma\delta)x+\alpha\beta\gamma\delta\end{aligned} \tag{4}$$

표 3 삭제 다항식의 계산
Table 3 Erasure Calculation table

R \ N	x^3	x^2
1	0	0
2	α	0
3	$\alpha+\beta$	$\alpha\beta$
4	$\alpha+\beta+\gamma$	$\alpha\beta+(\alpha+\beta)\gamma$
5	$\alpha+\beta+\gamma+\delta$	$\alpha\beta+(\alpha+\beta)\gamma+(\alpha+\beta+\gamma)\delta$

R \ N	x^1	x^0
1	0	0
2	0	0
3	0	0
4	$\alpha\beta\gamma$	0
5	$\alpha\beta\gamma+[\alpha\beta+(\alpha+\beta)\gamma)]\delta$	$\alpha\beta\gamma\delta$

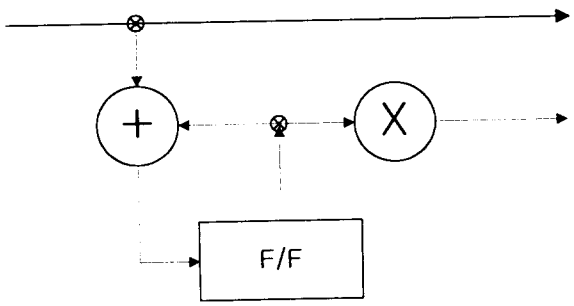


그림 7 삭제 다항식 계산을 위한 기본 셀
Fig. 7 Basic cell of erasure polynomial calculation circuit

2.4 수정 유클리드 연산부

유클리드 알고리즘은 오류 위치다항식을 구하기 위한 방법중의 하나로, Berlekamp 알고리즘에 비해 이해가 쉽고 하드웨어 구현이 용이하다는 장점을 가지고 있다. 이러한 유클리드 알고리즘은 두 다항식의 최대공약수를 구하는 방법으로 본 논문에서는 반복적인 구조를 사용하여 전체 하드웨어의 크기를 줄일 수 있도록 설계하였다.

$$\begin{aligned}R_0(x) &= A(x), \quad Q_0(x)=S(x)\\ \lambda_0(x) &= 0, \quad \mu_0(x)=1\\ \lambda_0(x) &= 1, \quad \eta_0(x)=0\end{aligned} \tag{5}$$

위와 같은 초기 조건하에서 유클리드 알고리즘을 적용하여 전개하면 다음과 같다[4].

$$\begin{aligned}R_i(x) &= [\sigma_{i-1}\cdot b_{i-1}\cdot R_{i-1}(x)+\sigma_{i-1}\cdot a_{i-1}\cdot Q_{i-1}(x)\\ &\quad -x^{ls}[\sigma_{i-1}\cdot a_{i-1}\cdot Q_{i-1}(x)+\sigma_{i-1}\cdot b_{i-1}\cdot R_{i-1}(x)]\\ \lambda_i(x) &= [\sigma_{i-1}\cdot b_{i-1}\cdot \lambda_{i-1}(x)+\sigma_{i-1}\cdot a_{i-1}\cdot \mu_{i-1}(x)\\ &\quad -x^{ls}[\sigma_{i-1}\cdot a_{i-1}\cdot \mu_{i-1}(x)+\sigma_{i-1}\cdot b_{i-1}\cdot \lambda_{i-1}(x)]\\ \gamma_i(x) &= [\sigma_{i-1}\cdot b_{i-1}\cdot \gamma_{i-1}(x)+\sigma_{i-1}\cdot a_{i-1}\cdot \eta_{i-1}(x)\\ &\quad -x^{ls}[\sigma_{i-1}\cdot a_{i-1}\cdot \mu_{i-1}(x)+\sigma_{i-1}\cdot b_{i-1}\cdot \gamma_{i-1}(x)]\end{aligned} \tag{6}$$

$$\begin{aligned}Q_i(x) &= \sigma_{i-1}\cdot Q_{i-1}(x)+\sigma_{i-1}\cdot R_{i-1}(x)\\ \mu_i(x) &= \sigma_{i-1}\cdot \mu_{i-1}(x)+\sigma_{i-1}\cdot \lambda_{i-1}(x)\\ \eta_i(x) &= \sigma_{i-1}\cdot \eta_{i-1}(x)+\sigma_{i-1}\cdot \gamma_{i-1}(x)\end{aligned} \tag{7}$$

여기서 a_{i-1}, b_{i-1} 은 각각 $R_{i-1}(x)$ 과 $Q_{i-1}(x)$ 의 최고차 항의 계수를 나타내며, 각각의 변수들이 의미하는 바는 다음과 같다[4][7][8].

$$\begin{aligned} l_{i-1} &= \deg(R_{i-1}(x)) - \deg(Q_{i-1}(x)) \\ ls &= |l_{i-1}| \\ \sigma_{i-1} &= 1 \quad \text{if} \quad l_{i-1} \geq 0 \\ \sigma_{i-1} &= 0 \quad \text{if} \quad l_{i-1} < 0 \\ \sigma_{i-1} &= |\sigma_{i-1} - 1| \end{aligned} \tag{8}$$

이러한 유클리드 알고리즘은 $\deg(\lambda(x)) > \deg(R(x))$가 되면 반복연산을 종류하고 그때의 $\lambda(x)$값과 $R(x)$값을 다음 연산 블록으로 전달하게 된다. 기존에 제안되었던 유클리드 연산부의 경우 $(N-I)^2$의 연산시간이 필요하며, 이는 초기 신디롬 계산을 위한 지연시간 N과 비교했을 때 두 연산 블럭간의 겹침 동작이 발생하는 원인이 되었다. 따라서, 본 논문에서는 다항식의 차수를 표준화하여 제어신호를 생성하는 방법을 제안하고자 한다. 이 경우 계산된 두 다항식의 차수만큼 이동시키는 별도의 회로가 설계되어야 하나 전체 계산시간을 기존의 $(N-I)^2$에서 $2(N-I)$ 로 줄임으로써 전체 RS 복호기의 속도를 향상시킬 수 있으며, Shao가 제시한 겹침 동작의 해소를 위해 사용되어지는 파이프라인 구조를 위한 셀의 수 역시 줄일 수 있다. 즉, 그림 7에서 보는 바와 같이 수정 유클리드 알고리즘을 수행하기 위해 입력되는 두 다항식 $T(x)$ 와 $\Lambda(x)$ 의 계수 값들이 동시에 입력되면 입력단의 MUX에서 초기값과 함께 제어신호 발생부로 입력되고, 이 제어 신호부에서는 수정유클리드 알고리즘의 연산을 위해 필요한 제어 신호를 계산한 후 다항식 연산부로 입력되며 이 다항식 연산부에서는 주어진 다항식의 계수를 이용해 수정 유클리드 알고리즘을 수행하게 된다.

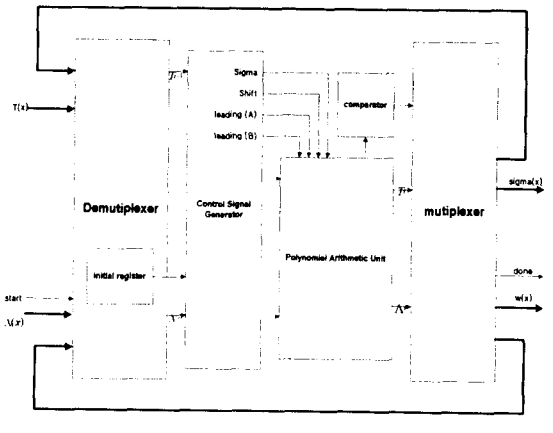


그림 8 제안된 수정 유클리드 연산부

Fig. 8 Proposed modified Euclid Calculation circuit

본 논문에서는 두 다항식의 비교와 최고차항의 계수를 제어신호부에서 처리함으로 효율적인 제어신호부의 설계가 매우 중요하다. 이를 위해 그림 9 에서와 같이 입력되는 다항식의 최고 차수에 1 값을 저장하는 부가의 표준화 회로를 추가시키는 방법을 사용하여, 두 다항식의 비교와 최고차항의 계수를 효율적으로 구할 수 있다. 제어 신호부는 이렇게 생성된 최고차항의 위치 값을 가지고 다항식 연산부의 연산에 필요한 곱셈기의 상수 값과 두 다항식의 차수의 차를 계산한 뒤 래치에 이 제어신호를 저장한 후 입력된 다항식을 가지고 그림 10의 다항식 연산부에서 연산을 수행하게 된다. 따라서, 제어신호와 각 연산 값을 동기 시키기 위해 (N-I)의 지연시간과는 별도로 1 clock을 더 소모하게 되며, 표 4에서 보는 바와 같이 기존의 유클리드 연산부보다 30% 정도 하드웨어의 크기가 증가하게 된다. 하지만 코드워드의 길이 만큼 이동시키는 과정이 필요 없으므로 전체 수정 유클리드 알고리즘을 수행하는데 기존의 구조보다 $(N-I)^2 - 2(N-I) = 4t \cdot (t-1)$ 만큼의 속도 향상을 기대할 수 있다.

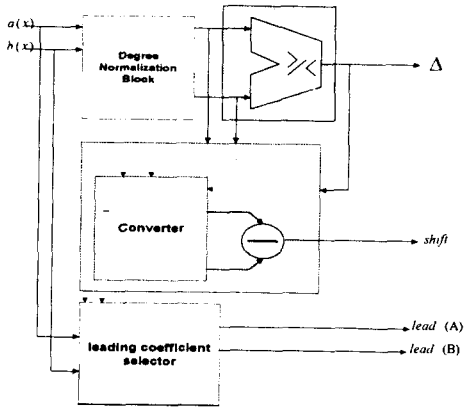


그림 9 수정 유클리드 연산부의 제어

Fig. 9 Arithmetic unit of modified euclid algorithm

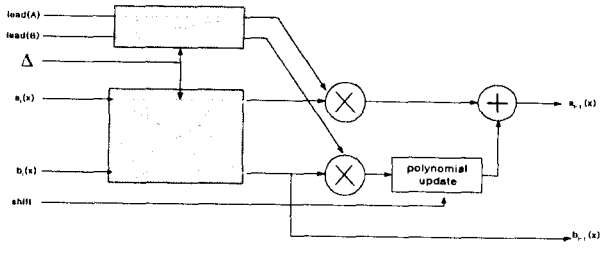


그림 10 수정 유클리드 연산부의 다항식 계산

Fig. 10 Control unit of modified euclid algorithm

2.5 오류 위치 및 오류 값의 계산

본 논문에서는 수정 유클리드 연산부에서 계산된 오류 위치 다항식의 근을 구하기 위해 Chien search 알고리즘을 사용하였다. 이러한 Chien search는 오류 위치 다항식과 오류 크기 다항식을 입력받아 에러가 발생한 위치를 계산하며, Chien search의 근을 이용한 Forney 알고리즘을 사용하여 그 크기를 계산한다. Chien search의 경우 수정 유클리드

연산부에서 계산된 오류위치다항식이 입력되면 α^i의 값을 대입시켜 그 결과가 0 인 값을 찾으며 이때의 값이 오류위치 다항식의 근이 된다[2]. 최종적으로 오류의 크기를 알려면 식 (9)에서와 같이 오류평가다항식의 미분 값을 알아야 하고 이를 위해 chien search 알고리즘의 수행과 함께 이의 미분 값이 동시에 계산되며 chien search 값이 구해지면 그 때의 오류평가다항식의 미분 값을 가지고 오류의 크기를 계산하게 된다. 계산된 값의 역수 계산을 위해서 ROM과 곱셈기를 이용하여 구현하였으며, 전체 오류 평가다항식을 위한 블록선도는 그림11과 같다[4].

$$e_{ik} \equiv \frac{-x_k \Omega(x_k^{-1})}{\Lambda(x_k^{-1})} \tag{9}$$

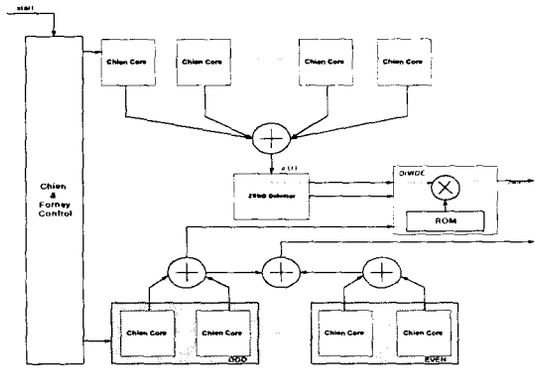
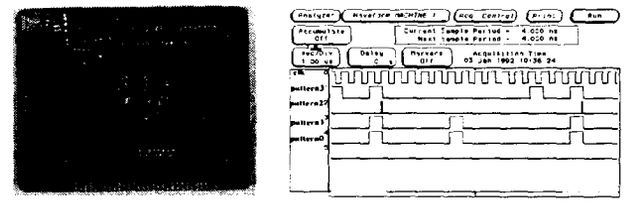


그림 11 오류크기 및 오류 위치 계산부
Fig. 11 Error magnitude and location calculation unit

3. 결 론

본 논문에서 제안한 RS 복호기의 동작 검증은 ALTERA의 MAXPlus II를 사용하여 게이트 레벨에서의 시뮬레이션을 수행하였으며, 이의 검증을 위해 FLEX 10K-50을 사용하여 동작을 검증하였다. 제안된 복호기는 약 3,5000 게이트 크기를 가졌으며, Flex 10K-50 디바이스의 특성상 20MHz 정도의 속도로 동작 가능하였으나, 설계시 43MHz까지 동작 가능하도록 설계하였다. 동작 검증을 위해 PCB를 이용한 범용 FPGA 테스트 보드를 제작하였고, 내부 테스트 벡터를 생성해 주는 부분은 수정의 용이를 위해 VHDL을 사용해 설계하였다. 사용한 테스트 벡터는 all-zero인 코드워드가 송신된 후 임의의 오류가 발생하였을 경우 및 반복적으로 발생한 오류에 대해 검증하였으며, 그림12에서와 같이 x^{12}, x^8, x^3 에서 각각 α^4 (0011), α^3 (1000), α^7 (1011)의 오류가 발생했을 경우, HP 1660A 로직 분석기를 이용한 검증 결과와 같이 오류가 발생한 위치와 그때의 크기인 α^4, α^3, α^7 을 정확히 복호함을 알 수 있다.

본 논문에서는 수정 유클리드 알고리즘 연산부를 단일 셀을 반복적으로 사용하여 구성함으로써 전체 회로 크기를 줄일 수 있었으며, 최고차 항을 소거시키기 위해 사용되었던 쉬프트 레지스터 대신에 수정 유클리드 연산의 최고차 항을 소거하기 위한 제어 신호부를 추가함으로써 복호기의 속도를 향상 시킬 수 있었다. 따라서, 오류 위치다항식을 계산하는 연산 시간을 줄임으로써 RS 복호기를 구성하는 각 연산 블록들간의 겹침 동작을 해소할 수 있었으며, 이를 통하여 전반적인 RS 복호기의 구조를 병렬처리와 파이프라인 구조를 효과적으로 적용할 수 있었다. 본 논문에서 설계된 RS 복호기의 비교를 위해 Shao가 제시한 방법에 따라 복호기의 면적을 피승수가 고정된 곱셈기와 D F/F의 면적을 a_u로 표준화해 비교하였으며, 이 경우 상수×상수의 곱셈기는 $2a_u$가 된다[4]. 비교를 위해 RS(255,223) 코드를 사용한 복호기를 기준으로 평가하였고 이의 결과는 표 4와 같다. Shao가 제시한 구조의 경우 수정 유클리드 연산부에서는 파이프라인구조의 적용을 위해 5개의 반복 셀이 필요한 반면에 본 논문에서 제안한 구조의 경우는 단일 셀의 사용으로도 동일한 성능을 가지므로, 단일 셀의 면적은 3배정도 증가하였으나, 전체 연산부의 면적은 30% 감소하는 효과를 얻을 수 있었다. 이러한 결과는 RS 복호기의 구조를 병렬 처리함으로써 나타나는 면적의 증가를 감소시킬 수 있다. 결과적으로 본 논문에서 제안한 RS 복호기의 경우 Shao가 제안한 구조보다 RS(255,223) 코드의 경우 전체 크기 면에서 시간 영역에서는 10%의 작은 면적으로도 약 3배의 속도 향상을 얻을 수 있었으며, G.S. Choi의 RS 복호기의 경우보다도 40%정도의 속도 향상을 기대할 수 있었다[8].



(a) FPGA 테스트 보드 (b) logic analyzer로 분석한 화면

그림 12 FPGA 테스트 보드와 logic analyzer로 분석한 화면
Fig. 12 FPGA Board and the verification using logic analyzer

표 4 칩의 성능 비교
Table 4 Performance comparisons of proposed decoder

	Shao		G.S. Choi		제안된 구조	
	크기	지연	크기	지연	크기	지연
신디롬 계산부	32 a_u	255	32 a_u	255	80 a_u	128
α^k 생성부	8 a_u		8 a_u		3 a_u	
신디롬 확장부	64 a_u	255	64 a_u	255	64 a_u	256
삭제다항식 확장부	64 a_u		64 a_u		64 a_u	
유클리드 연산부	200 a_u	1024	119 a_u	272	135 a_u	64
오류크기 계산부	32 a_u	32	32 a_u	32	32 a_u	32
오류위치 계산부	32 a_u		32 a_u		32 a_u	
Total	432 a_u	1,566	351 a_u	814	410 a_u	480

참 고 문 헌

[1] F.J MacWilliams and N.J.A. Sloane, "The Theory of Error Correcting Codes," North-Holland, 1981.

[2] 李晩榮, "BCH 부호와 Reed-Solomon 부호," 대우학술총서 · 자연과학 65, 民音社, 3月, 1990.

[3] R. T. Chien, "Cyclic decoding procedures for the Bose-Chaudhuri-Hocquenghem codes," IEEE Trans. Inf. Theory, Vol. IT-10, pp 357-363, Oct. 1964.

[4] H. M. Shao and et. al., " A VLSI design of a pipeline Reed-Solomon Decoder," IEEE Trans. Comput., vol. C-34. pp393-403, May 1985

[5] Sunghoon Kwon, "An Area-Efficient VLSI Architecture of A Reed-Solomon Decoder/Encoder For Digital VCRs," IEEE Trans. on Computer, vol. 43, No4, pp1019-1027, 1997.

[6] Kuang Yung Liu, "Architecture for VLSI Design of Reed-Solomon Decoders," IEEE Trans. on computers, Vol . C-33, No 2, pp 178-189, Feb.1984

[7] R. P. Brent and H. T. Kung, "Systolic VLSI arrays for polynomial GCD computations," DEP. Comput. Sci,. Carneigie-Mellon Univ., Pittsburgh. PA Rep., 1982

[8] GoangSeog Choi, HoonSoon Choi, YoungHwan Kim, " RS Decoder using Modified Euclidean Algorithm for DVD/CD," Proceedings of the ISCPAT(The international Conference on Signal Processing Applications & Technology- Volume 1 , 9/14/97

저 자 소 개

김 동 순 (金 東 淳)

1972년 8월 11일 생. 1997년 인하대 전자재료공학과(공학사). 1999년 동 대학원 전자재료공학과(공학 석사). 1999년~현재 전자부품연구원 연구원.

Tel : 0333-6104-264,
Fax : 0333-6104-048
E-mail : dskim@nuri.keti.re.kr

최 종 찬 (崔 鍾 讚)

1963년 1월 2일 생. 1985년 경희대학교 전자공학과 졸업. 1990년 삼성전관 주임연구원. 1991년~현재 전자부품연구원 선임연구원.

Tel : 0333-6104-347,
Fax : 0333-6104-048
E-mail:choijc@nuri.keti.re.kr

정 덕 진 (鄭 德 鎭)

1948년 2월 8일생. 1970년 서울대 공대 전기공학과 졸업. 1984년 미국 Utah State University 졸업(석사). 1988년 미국 University of Utah 졸업(공박). 1989년~인하대 전자재료공학과 교수

Tel : 032-874-1663,
Fax : 032-875-5882
E-mail:djchung@dragon.inha.ac.kr