# طراحی یک دیکدر BCH بهینه جهت افزایش اطمینان در ذخیره‌سازی اطلاعات و تصحیح خطا در حافظه‌های فلش

سعیده نبی‌پور[۱]، دانش‌آموخته کارشناسی ارشد، جواد جاویدان[۲]، استادیار، غلامرضا زارع فتین[۳]، استادیار
۱-۲-۳- گروه مهندسی برق و کامپیوتر- دانشکده فنی و مهندسی - دانشگاه محقق اردبیلی - اردبیل - ایران
saeideh.nabipour@gmail.com[1], javidan@uma.ac.ir[2], zare@uma.ac.ir[3]

چکیده: کاهش ابعاد ترانزیستورها در نسل جدید حافظه‌های فلش و رهسپار شدن آن‌ها به سمت حوزه‌های طراحی نانومتر منجر به عدم صحت در برنامه‌ریزی و پاک کردن اطلاعات در این طراحی‌ها شده؛ درنتیجه قابلیت اطمینان در ذخیره‌سازی اطلاعات به چالشی مهم در ساختار این نوع حافظه‌ها تبدیل شده است. جهت مقابله با چنین چالشی در کنترل‌کننده این نوع از حافظه‌ها از کدهای تصحیح خطای BCH استفاده می شود. دو نکته اساسی در فرآیند دیکدینگ کد BCH عبارت‌اند از: میزان تأخیر در فرآیند تصحیح خطا و حجم سخت‌افزاری هر یک از زیر بلوک‌ها. در این مقاله جهت افزایش سرعت در فرآیند تصحیح خطا و نیز افزایش راندمان مدار دیکدر، روشی مؤثر مبتنی بر معماری موازی برای زیر بلوک‌های دیکدر BCH و همچنین استفاده از تکنیک خط لوله پیشنهاد شده است. از طرف دیگر راه‌حل پیشنهادی جهت کاهش حجم سخت‌افزار بلوک دیکدر BCH، استفاده از الگوریتم اشتراک‌گذاری XORها جهت حذف گیت‌های تکراری در بلوک Chien search است. دیکدر پیشنهادشده توسط زبان توصیف سخت‌افزار VHDL شبیه‌سازی و سپس با استفاده از نرم‌افزار Xilinx ISE سنتز شده است. نتایج شبیه‌سازی‌ها نشان می‌دهند که الگوریتم پیشنهادی در مقایسه با روش‌های مشابه ضمن کاهش زمان فرآیند تصحیح خطا، توانسته است کاهش چشم‌گیری در حجم سخت‌افزاری بلوک دیکدر BCH داشته باشد.

واژه‌های کلیدی: بلوک اینکدر و دیکدر BCH، حافظه‌ی NAND flash، قابلیت اطمینان، کدهای تصحیح خطا، کد BCH.

# Design of an Optimum BCH Decoder to Increase the Reliability of Data Storage and Error Correction Mechanism in Flash Memories

S. Nabipour[1], J.d Javidan[2], assistant professor, G.R. Zare Fatin[3], assistant professor

1, 2 &3- Department of Electrical and Computer Engineering, Faculty of Engineering, University of Mohaghegh Ardabili, Ardabil, Iran- Email: saeideh.nabipour@gmail.com[1], javidan@uma.ac.ir[2], zare@uma.ac.ir[3]

**Abstract:** The shrinking of transistor dimensions and migration to nanometer region have increased the data storage errors in new generations of flash memories. Therefore, the reliability of data storage is an important challenge in the structure of these memories. In order to confront with this challenge, BCH error correction codes are utilized in the controller of these memories. There are two important points in the optimization process of a BCH decoder: speeding up the computation and reducing the hardware complexity. To speed up the decoding process, a parallel architecture is utilized for various building blocks. A Pipeline scheme is also adopted in BCH decoder to increase the throughput. To implement this parallel BCH decoder in an area-efficient manner, an iterative matching scheme is proposed to reduce the Chien search hardware complexity by reducing the number of XOR gates through removing the duplicate gates and sharing the remaining ones. The proposed decoder along with BCH encoder have been implemented in VHDL hardware definition language and synthesized in Xilinx ISE. The proposed decoder has been implemented in VHDL hardware definition language and synthesized in Xilinx ISE. The simulation results show that the proposed algorithm could reduce the decoding time and hardware complexity .

**Keywords:** BCH encoder and decoder, NAND flash memory, reliability, error correction code, BCH code.

تاریخ ارسال مقاله: ۱۳۹۴/۰۲/۱۵
تاریخ اصلاح مقاله: ۱۳۹۴/۰۴/۰۹
تاریخ پذیرش مقاله: ۱۳۹۴/۰۶/۳۰
نام نویسنده مسئول: دکتر جواد جاویدان
نشانی نویسنده مسئول: ایران- اردبیل- خیابان دانشگاه - دانشگاه محقق اردبیلی - دانشکده فنی و مهندسی

## ۱- مقدمه

نخستین رقیب جدی برای دیسک‌های سخت، حافظه‌های فلش بودند. این نوع از حافظه‌های نیمه‌رسانا همانند دیسک‌های سخت غیرفرار هستند، ولیکن دست‌یابی به محتویات آن‌ها بین ۱۰۰ تا ۱۰۰۰ برابر سریع‌تر از دیسک‌های سخت است. اندازه بسیار کوچک‌تری دارند. ازلحاظ مصرف توان بسیار مقرون به‌صرفه‌تر هستند و در مقابل ضربه بسیار مقاوم‌ترند. حافظه‌های فلش دارای این توانایی هستند که اطلاعات را روی یک تراشه از جنس سیلیکون ذخیره نمایند، بدون این‌که برای نگه‌داری آن‌ها، نیاز به منبع تغذیه باشد. حافظه‌های فلش در حقیقت نوعی از حافظه‌های EEPROM[1] (حافظه‌های فقط خواندنی با امکان برنامه‌ریزی و پاک کردن داده‌ها به‌صورت الکتریکی) هستند. به دلیل رواج گسترده حافظه‌های فلش در گوشی تلفن‌های همراه، دوربین‌های دیجیتال، پخش‌کننده‌های MP3، دستگاه‌های ذخیره‌سازی همچون درایوهای حالت‌جامد (SSD[2]) و نظایر آن، برای بهبود فناوری حافظه‌های فلش، سرمایه‌گذاری‌های کلان و بازار فروش گسترده‌ای از سوی تولیدکنندگان این نوع حافظه‌ها وجود دارد [۱].

حافظه‌های فلش با دانش و فناوری‌های مختلفی ساخته می‌شوند. نخستین حافظه فلش به دلیل شباهت هر یک از سلول‌های آن با گیت‌های استاندارد NOR به NOR Flash شهرت یافت. به فاصله چند سال حافظه‌های NAND Flash عرضه شدند که چگالی و ظرفیت بسیار بالاتری داشتند، با این تفاوت که خواندن و نوشتن داده ها در نوع NAND به‌صورت بلوکی صورت می‌گرفت و طبعاً سیم‌بندی‌های لازم برای دسترسی تصادفی به هر نقطه از فضای حافظه حذف شده بود. حافظه‌های فلش در دو نوع ساختار سلولی مختلف: تک‌سطحی (SLC[3]) و چندسطحی (MLC[4]) تولید می‌شوند. در حافظه‌هایی که بر پایه فنّاوری MLC طراحی می‌شوند، در هر سلول حافظه ۲ بیت اطلاعات ذخیره می‌شود، امّا در چیپ‌های نوع SLC تنها ۱ بیت داده در هر سلول حافظه ذخیره می‌شود.

حافظه‌های فلش مبتنی بر ساختار سلولی n سطحی با ایجاد $2^n$ حالت قادر به ذخیره‌سازی n بیت اطلاعات خواهد بود. هر یک از حالات با یک گستره ولتاژ آستانه ایجاد می‌شوند و هر سطح بیانگر یک ولتاژ آستانه خواهد بود. امروزه فناوری ساخت حافظه‌های فلش باهدف افزایش ظرفیت ذخیره‌سازی و کاهش هزینه تولید، دائماً به سمت کوچک‌تر شدن در حال پیشروی است. کوچک شدن اندازه MOSFET، به علت مزایای موجود، بسیار مورد توجه است، امّا MOSFETهای خیلی ریز در سلول‌های فلش قابل‌استفاده نیستند. کاهش ابعاد هندسی سلول‌های حافظه‌های فلش منجر به کاهش فواصل بین سطوح مجاور در سلول‌های این نوع حافظه‌ها می شود؛ درنتیجه افزایش سیکل خواندن و نوشتن اطلاعات می‌تواند منجر به نوسان در مقدار ولتاژ آستانه ترانزیستورها شود. این پدیده مخصوصاً زمانی شدت می‌گیرد که از سلول‌های چندسطحی استفاده کنیم و باید به‌طور دقیق و با اطمینان کامل بتوان روی آن اطلاعات را ذخیره نمود.

ترانزیستورهای خیلی کوچک قابلیت تحمل ولتاژ بالا در فرآیند نوشتن و پاک کردن اطلاعات فلش‌ها را ندارند. بار زیاد موجود در لایه نازک و شکننده اکسید عایق که بین گیت شناور و لایه زیرین و گیت شناور و سورس و درین موجود است، وضعیت الکترون‌های گیت شناور را به سمت اختلال می‌کشاند. این امر منجر به عدم صحت در برنامه‌ریزی و پاک کردن اطلاعات و بروز خطا در حافظه فلش می‌شود. این در حالی است که فناوری کوچک ساختن ساختار فیزیکی ترانزیستورهای MOSFET و حافظه‌های فلش با سرعت بالایی روبه افزایش است و همین مسئله، ساخت این دسته از حافظه‌ها را با دشواری زیادی همراه کرده است و فناوری ساخت آن‌ها فقط به چند کشور دنیا محدود شده ا ست [۱، ۲].

خطای بیتی در حافظه‌های فلش به دو دسته خطای نرم و خطای سخت دسته‌بندی می‌شود. خطای سخت از فروپاشی لایه اکسید ناشی از محدودیت‌های نوشتن/پاک کردن حافظه‌ی فلش ایجاد شده و منجر به اشکال دائمی بیت‌ها در آرایه حافظه می‌شود. خطای نرم از مکانیسم‌های مختلفی همچون بروز اختلال در سیکل خواندن و نوشتن اطلاعات و حفظ اطلاعات نشأت می‌گیرد که می‌تواند در سیکل بعدی نوشتن/پاک حافظه برطرف شود. در فناوری چند سطحی سلول‌های حافظه فلش که مسئله قابلیت اطمینان در ذخیره‌سازی داده به‌عنوان چالشی جدی در آن‌ها مطرح است، از روش کنترل خطا استفاده می‌شود [۲]. به‌طورکلی جهت مقابله با چنین چالشی در کنترل‌کننده این نوع از حافظه‌ها، بلوکی بنام ECC[5] تعبیه شده است که وظیفه اصلی آن غلبه بر نرخ بالای خطا است، به‌طوری‌که جهت افزایش اطمینان در ذخیره‌سازی، روش‌های پردازش سیگنال همچون کدهای تصحیح خطا با قابلیت تصحیح چندین بیت خطا موردنیاز است.

کدهای تصحیح خطا به گیرنده اجازه می‌دهد تا اطلاعات منتقل‌شده را بدون نیاز به درخواست ارسال مجدد اطلاعات شناسایی کند. آن‌ها در ابتدا به پیام ارسال‌شده یک افزونگی اضافه می‌کنند که این فرآیند اینکدینگ نامیده می شود و در فرستنده صورت می‌گیرد. سپس خطای ایجادشده حین انتقال، بر اساس قابلیت استخراج افزونگی در فرآیندی بنام دیکدینگ تصحیح می‌گردد که در گیرنده انجام می‌شود. خروجی فرآیند اینکدینگ، کلمه کدی است که شامل خود پیام و افزونگی آن است که افزونگی به‌عنوان بیت توازن نامیده می‌شود. در حافظه‌های فلش کوچک‌ترین واحدی که می‌تواند به‌طور همزمان خوانده یا برنامه‌ریزی شود، صفحه نام دارد. هر صفحه به طور مستقل توسط بلوک ECC کدگذاری/کدگشایی می‌شود. در طی سیکل نوشتن، اطلاعات از گذرگاه I/O توسط بلوک ECC اینکد شده و در صفحه موردنظر نوشته می‌شود و در طول سیکل خواندن، بلوک ECC اطلاعات هر صفحه از حافظه را دیکد کرده و به گذرگاه I/O ارسال می‌کند.

کدهای تصحیح خطای BCH[6] یکی از رایج‌ترین کدهایی هستند که در حافظه‌های فلش جهت تصحیح خطا استفاده می‌شوند. کدهای BCH کدهای تصحیح خطای پیچیده‌ای هستند که جهت تصحیح

خطای چندگانه مناسب هستند. کدهای BCH برای اولین بار در سال ۱۹۵۹ توسط Hocquenghem کشف و در سال ۱۹۶۰ به‌صورت مستقل توسط Bose و Chaudhuri اثبات شد [۳]. فرآیند کدگشایی توسط بلوک دیکدر BCH جهت تصحیح خطا شامل چهار مرحله: محاسبه سیندروم، محاسبه چندجمله‌ای خطایاب، الگوریتم Chein search جهت یافتن مکان وقوع خطا و تصحیح خطا است. تاکنون کارهای متعددی در زمینه بهینه‌سازی دیکدر BCH صورت گرفته است. همچنین الگوریتم‌های متفاوتی باهدف کاهش زمان تأخیر در فرآیند کدگشایی جهت تصحیح خطا و کاهش حجم سخت‌افزاری بلوک دیکدر، در هر یک از زیر بلوک‌های آن ارائه شده است. در فرآیند پیاده سازی دیکدر BCH دو پارامتر زمان و حجم سخت‌افزاری در تقابل با یکدیگر قرار دارند. آنچه در اکثر روش‌های ارائه‌شده به چشم می‌خورد، مصالحه‌ای است که بین زمان موردنیاز در فرآیند دیکدینگ جهت تصحیح خطا و حجم سخت‌افزاری مدار دیکدر وجود دارد.

در مرجع [۴] الگوریتمی جهت کاهش پیچیدگی سخت‌افزاری ناشی از ضرب مبتنی بر پیمانه چندجمله‌ای اولیه در ضرب کننده‌های ثابت میدان متناهی در بلوک موازی Chien search با استفاده از عملیات شیفت ارائه شده است که علاوه بر کاهش تعداد ضرب کننده‌ها، قادر به کاهش تعداد جمع کننده ها نیز است. در مرجع [۵] الگوریتمی بنام شبکه های ترکیبی چندجمله‌ای مینیمال جهت جایگزینی ضرب کننده‌های ثابت میدان متناهی ارائه شده است. الگوریتم پیشنهادی معادله Chien search را با استفاده از چندجمله‌ای مینیمال مجدداً فرمول‌بندی کرده و ویژگی مهم آن ادغام دو بلوک تولید سیندروم و Chien search با کم‌ترین پیچیدگی سخت‌افزاری است. در مرجع [۶] باهدف کاهش حجم سخت‌افزاری بلوک تولید سیندروم از روش on-demand استفاده شده است. بر اساس روش پیشنهادی در بلوک سیندروم تنها سیندروم‌های فرد تولید می‌شوند و درصورتی‌که در بلوک تولید چندجمله‌ای خطایاب، سیندروم‌های زوج موردنیاز باشد با استفاده از ویژگی $S_{2i} = S_i^2$، محاسبه خواهند شد.

در مرجع [۷] جهت کاهش تأخیر فرآیند کدگشایی از روش کدگذاری مجدد استفاده شده است که در آن محاسبه سیندروم در دو فاز تشخیص خطا و محاسبه مؤلفه‌های سیندروم انجام می شود. روش پیشنهادی فاز اول را به بلوک اینکدر که در زمان خواندن اطلاعات بیکار است، محول می‌کند؛ درنتیجه بدون استفاده از هرگونه سخت‌افزار اضافی و تنها با استفاده از LFSR قادر به تشخیص خطا است. از طرفی جهت تولید چندجمله‌ای خطایاب الگوریتمی بنام oBM پیشنهاد شده است که قادر به اجرای هم‌روند دستورات است. در مراجع [۸، ۹] نیز الگوریتم‌های بهینه‌ای همچون RiBM، SiBM و TiBM جهت تولید چندجمله‌ای خطایاب پیشنهاد شده است که در پالس ساعت کم‌تر با پیچیدگی سخت‌افزاری کم‌تری قادر به یافتن ضرایب چندجمله‌ای خطایاب هستند.

دو نکته اساسی که در فرآیند دیکدینگ کدهای BCH علاوه بر کارکرد تصحیح خطا موردتوجه است، زمان تأخیر در فرآیند دیکدینگ و حجم سخت‌افزاری هر یک از زیر بلوک‌ها است. هدف ما در این مقاله آن است که با دو چالش مذکور مقابله کرده و به معرفی روش‌ها و الگوریتم‌های بهینه‌ای بپردازیم که تا حد زیادی زمان تأخیر در فرآیند کدگشایی را کاسته و درعین‌حال منجر به پیچیدگی سخت‌افزاری مدار نگردد و مصالحه‌ای بین این دو فاکتور ایجاد شود. با توجه به گسترش استفاده از کدهای تصحیح خطای BCH در حافظه‌های فلش باهدف افزایش قابلیت اطمینان در ذخیره‌سازی اطلاعات، لزوم ارائه روش‌های بهینه‌سازی که هر دو پارامتر زمان و حجم سخت‌افزاری را در فرآیند دیکدینگ همزمان مورد توجه قرار می‌دهند، امری مهم تلقی می‌شود.

در این مقاله ضمن استفاده از روش توازی در زیر بلوک‌های دیکدر BCH باهدف کاهش زمان تأخیر در فرآیند دیکدینگ، الگوریتم اشتراک‌گذاری XORهای تکراری جهت کاهش حجم سخت‌افزاری بلوک دیکدر BCH معرفی شده است. برای نشان دادن برتری روش پیشنهادی، مجموعه‌ای از داده‌ها با تعداد خطاهای متفاوت به بلوک دیکدر BCH اعمال شده است و نتایج با روش‌های مشابه بررسی خواهد شد. نتایج حاصل، بهبودهای پیش‌بینی‌شده توسط معماری‌ها و تکنیک‌های به کار گرفته‌شده را تأیید می‌کند.

ادامه مقاله بدین‌صورت تدوین شده است: بخش (۲) به توصیف کدهای تصحیح خطای BCH می‌پردازد و عملکرد دو بلوک اینکدر و دیکدر BCH را در کنترل‌کننده حافظه‌های فلش بررسی می‌کند. در بخش (۳) معماری موازی بلوک دیکدر BCH جهت کاهش زمان تأخیر در فرآیند دیکدینگ معرفی شده است. سپس در بخش (۴) الگوریتم اشتراک گذاری XOR ها در بلوک دیکدر BCH جهت کاهش حجم سخت‌افزاری به‌طور کامل تشریح می شود. بخش (۵) به معرفی تکنیک خط لوله جهت افزایش بازدهی مدار دیکدر BCH اختصاص دارد. بخش (۶) به توضیح و تحلیل نتایج به‌دست‌آمده از شبیه‌سازی می‌پردازد. درنهایت در بخش (۷) نتیجه‌گیری و خلاصه مقاله ارائه می‌شود.

## ۲- توصیف کدهای تصحیح خطای BCH

تاکنون کدهای تصحیح خطای متنوعی شامل کدهای همینگ، کدهای Reed-Solomon، کدهای BCH، کدهای Goppa، کدهای Golay و غیره جهت تصحیح خطا در حافظه‌های فلش پیشنهاد شده است. اگرچه کارآمدی کدهایی همچون کدهای Goppa جهت تصحیح خطا اثبات شده است [۲]، امّا با توجه به گستره وسیع حافظه‌های فلش و لزوم استفاده از مکانیسم‌هایی با دقت بسیار بالا جهت تصحیح خطا با بالاترین کارایی و کم‌ترین پیچیدگی سخت‌افزاری، تولیدکنندگان این نوع حافظه‌ها به سمت مجموعه کدهایی باقابلیت بالا در تصحیح خطا و پیاده‌سازی سخت‌افزاری بهینه روی آورده‌اند.

طرح های قدیمی حافظه های فلش مبتنی بر ساختار SLC از کدهای همینگ جهت تصحیح خطا استفاده می‌کردند. باوجود ساختار ساده کدهای همینگ، این نوع کدها قادر به اصلاح فقط یک خطای

تصادفی‌اند، به همین دلیل عملاً در حافظه‌های فلش مفید نیستند. با توجه به افزایش نرخ خطا در نسل جدید حافظه‌های SLC و MLC فلش، طراحان این نوع حافظه‌ها از کدهای قدرتمندتر و پیچیده‌تری همچون Reed-Solomon و BCH استفاده می‌کنند. این کلاس از کدها از خانواده کدهای بلوکی خطی و زیرمجموعه‌ی کدهای چرخشی هستند [۳]. کدهای Reed-Solomon (RS) عملیات تصحیح خطا را روی سمبل‌های چندبیتی انجام داده و عملکرد مناسبی در مقابل نویزهای انفجاری دارند، درحالی‌که کدهای BCH عملیات تصحیح خطا را روی سمبل‌های تک‌بیتی انجام داده و نسبت به کدهای RS عملکرد بهتری در مقابل نویزهای تصادفی دارند.

بررسی‌های صورت گرفته روی حافظه‌های فلش نشان می‌دهد که خطاهای ایجادشده در هر صفحه از حافظه از نوع خطای تصادفی هستند، به همین دلیل کدهای BCH از رایج‌ترین کدهایی هستند که در حافظه‌های فلش استفاده می‌شوند [۱۰]. کدهای BCH مبتنی بر بسط میدان گالوا به‌صورت $GF(2^m)$ هستند و با استفاده از حساب میدان متناهی ساخته و کدگشایی می‌شوند. به‌طور خلاصه $GF(2^m)$ شامل تمام ترکیبات $m$ بیتی است که به‌صورت توان‌های $\alpha$ یا عنصر اولیه میدان گالوا نشان داده می شوند. میدان گالوای $GF(2^m)$ با استفاده از چندجمله‌ای بنام چندجمله‌ای اولیه از درجه $m$ که $\alpha$ ریشه آن است، ساخته می‌شود. برای هر عدد صحیح مثبت $m(m \geq 3)$، یک کد دودویی BCH با پارامترهای نشان داده‌شده در جدول ۱ وجود دارد [۳].

جدول ۱: پارامترهای کد تصحیح خطای **BCH** [۳]

| | |
|---|---|
| طول قالب کد | $n = 2^m - 1$ |
| تعداد بیت افزونگی | $n - k \leq mt$ |
| حداقل فاصله | $d_{min} \geq 2t + 1$ |
| قابلیت تصحیح خطا | $t = \left\lfloor \frac{n-k}{2} \right\rfloor$ |

جدول ۱ پارامترهای کد BCH را نشان می‌دهد که شامل: $n$: تعداد کل بیت‌ها به انضمام بیت افزونگی، $k$: طول پیام، $t$: قابلیت تصحیح خطا، $m$: گسترش میدان متناهی $GF(2^m)$ و $d_{min}$ حداقل فاصله بین دو کلمه کد است. عمدتاً طول پیام برای حافظه‌های فلش توانی از ۲ است. با توجه به این‌که اندازه یک صفحه یا سکتور در یک حافظه فلش لزوماً با پارامتر $k$ یک کد BCH یکسان نیست، از کدهای کوتاه‌شده BCH جهت تصحیح خطا استفاده می شود. کدهای کوتاه شده BCH$(n\text{-}l, k\text{-}l, t)$ با حذف $l$ بیت صفر به دست می‌آید و قابلیت تصحیح خطای آن همانند کد BCH اولیه است [۱۱].

بانک حافظه NAND Flash متشکل از چندین بلوک است که هر بلوک شامل ۶۴ صفحه برای حافظه نوع SLC و ۱۲۸ صفحه برای حافظه نوع MLC است. هر بلوک حافظه شامل ۲۰۴۸ بایت داده به انضمام ۶۴ بایت افزونگی است، از طرفی در هر بلوک از حافظه NAND Flash، هر صفحه به چهار سکتور ۵۱۲ بایتی تقسیم می‌شود و هر سکتور شامل ۱۶ بایت افزونگی است؛ بنابراین با توجه به $۸۱۹۲ = ۲^{۱۳} \leq ۸ \times (۱۶ + ۵۱۲) \leq ۲^{۱۲}$، این نوع از حافظه‌ها تحت میدان گالوای $GF(2^{13})$ عمل می‌کنند. با توجه به این‌که در حافظه‌های NAND Flash، ۵۱۲ بایت معادل ۴۰۹۶ بیت داده از یک آرایه از حافظه به‌طور همزمان خوانده می‌شود، کد کوتاه‌شده BCH(۴۲۰ ,۴۰۹۶) تحت میدان گالوای $GF(2^{13})$ جهت تصحیح خطای احتمالی در هر صفحه از حافظه به کار گرفته می‌شود [۶]؛ امّا در مورد حافظه‌های NOR Flash که یک صفحه به حجم ۲۵۶ بیت داده (۱۶ کلمه ۱۶ بیتی) به‌طور همزمان خوانده می‌شود، کد کوتاه‌شده BCH(۲۷۴ ,۲۵۶) تحت میدان گالوای $GF(2^9)$ جهت تصحیح خطا به هر صفحه از حافظه اعمال می‌شود [۱۲، ۱۳].

کدهای BCH قادر به تصحیح $t$ خطا یا کم‌تر در بلوکی به طول $n = 2^m - 1$ بیت هستند. چندجمله‌ای مولد این نوع کدها با ریشه‌هایشان که عناصری از میدان گالوا $GF(2^m)$ هستند، مشخص می‌شود. عنصر $\alpha$ به‌عنوان عنصر اولیه $GF(2^m)$ شناخته می‌شود. چندجمله‌ای مولد $g(x)$ یک کد تصحیح خطای BCH به طول $n = 2^m - 1$، چندجمله‌ای با کم‌ترین درجه تحت میدان $GF(2)$ است، به‌طوری‌که $\alpha, \alpha^2, ..., \alpha^{2t}$ ریشه‌های آن هستند ($g(\alpha^i)_{1 \leq i \leq 2t} = 0$). اگر $\phi_i(X)$ چندجمله‌ای با کوچک‌ترین درجه تحت میدان $GF(2)$ باشد، به‌طوری‌که $\phi(\alpha^i) = 0$، در این صورت $\phi_i(X)$ چندجمله‌ای کمینه عنصر $\alpha^i$ است و $g(x)$ از کوچک ترین مضرب مشترک $\phi_1(X), \phi_2(X), ..., \phi_{2t}(X)$ مطابق معادله‌ی (۱) به دست می‌آید [۳].

$$g(X) = LCM\{\phi_1(X), \phi_2(X), ..., \phi_{2t}(X)\} \quad (۱)$$

بلوک کد تصحیح خطا (ECC) در کنترل‌کننده حافظه‌های فلش از دو زیر ماژول اصلی تشکیل شده است: ماژول اینکدر و ماژول دیکدر. زمانی که $k$ بیت داده کاربر در حافظه فلش نوشته می‌شود، مدار اینکدر بیت‌های افزونگی را تولید می‌کند و آن‌ها را به $k$ بیت اطلاعات داده اضافه کرده و یک کلمه کد $n$ بیتی تولید می‌کند؛ سپس تمام کلمه کد در یک صفحه از بانک حافظه نوشته و ذخیره می‌شود. در طی عملیات خواندن، مدار دیکدر خطاها را در یک کلمه کد جستجو می‌کند و با توجه به قابلیت تصحیح خطا، کلمه کد حاوی خطا را تصحیح کرده و درنتیجه کلمه کد صحیح بازیابی می‌شود.

### ۲-۱- بلوک اینکدر **BCH**

عمل کد کردن توسط چندجمله‌ای مولد انجام می‌شود. ماژول اینکدر در بلوک ECC کنترل‌کننده حافظه فلش، یک داده‌ی $k$ بیتی را به یک کلمه کد $n$ بیتی به‌صورت $(c_1, c_2, ..., c_n), (c_i \in GF(2))$ طی مراحل زیر کدگذاری می‌کند:

۱- $v(x)$ را به تعداد $n-k$ بیت به سمت چپ شیفت داده تا $n-$

$k$ بیت با مقدار صفر در سمت راست آن ایجاد شود. این عمل را می‌توان با ضرب کردن $v(x)$ در $X^{n-k}$ نمایش داد: $X^{n-k}v(X)$.

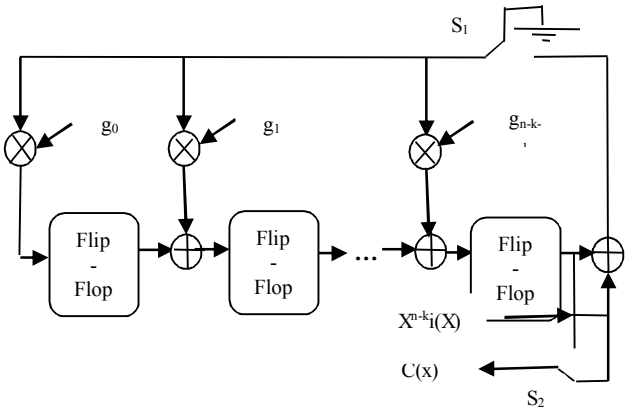
۲- باقی‌مانده چندجمله‌ای حاصل از مرحله قبل نسبت به چندجمله‌ای مولد محاسبه می‌شود (مقدار بیت افزونگی):

$$r(X) = X^{n-k}v(X) \bmod g(X) \qquad (۲)$$

۳- $r(X)$ در $(n-k)$ مکان خالی در سمت راست $X^{n-k}v(X)$ قرار می‌گیرد تا یک کلمه کد $n$ بیتی بنام $C(X)$ تولید شود:

$$C(X) = V(X)x^{n-k} + \mathrm{Re}\,m(V(X)x^{n-k})_{g(x)} \qquad (۳)$$

شیفت رجیستر خطی بازخوردی (LFSR[7]) بخش اصلی یک بلوک اینکدر را تشکیل می‌دهد که بلوک دیاگرام آن در شکل ۱ نشان داده شده است [۱۴]. مقدار بیت افزونگی با استفاده از یک شیفت رجیستر به طول $(n-k)$ که مساوی درجه چندجمله‌ای مولد است، با اتصال بازخوردی مرتبط با ضرایب چندجمله‌ای مولد به دست می‌آید.




شکل ۱: بلوک دیاگرام اینکدر **BCH**

### ۲-۲- بلوک دیکدر **BCH**

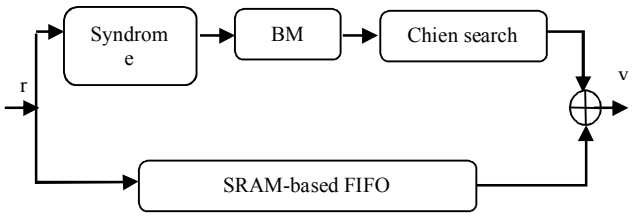
شکل ۲ نشان می‌دهد که کدگشایی کد BCH شامل ۴ مرحله: محاسبه سیندروم، محاسبه چندجمله‌ای خطایاب، الگوریتم Chien search جهت یافتن مکان وقوع خطا و تصحیح آن است. با توجه به محدودیت‌های ذاتی حافظه‌های فلش در طی سیکل ذخیره‌سازی داده، تکرار چرخه نوشتن/پاک‌سازی[8] ممکن است به اطلاعات ذخیره‌شده در حافظه صدمه بزند، درنتیجه منجر به وقوع خطا در عملیات خواندن می‌شود. اگر کلمه کد ذخیره‌شده در حافظه به‌صورت $c(X) = c_0 + c_1X + c_2X^2 + \ldots + c_{n-1}X^{n-1}$ باشد و خطای انتقال منجر به دریافت کلمه کد $r(X)$ مطابق معادله (۴) شود.

$$r(X) = r_0 + r_1X + r_2X^2 + \ldots + r_{n-1}X^{n-1} \qquad (۴)$$

پس از افزوده شدن خطای حافظه $e$ به بردار کدشده $c$، بردار دریافتی $r$، به‌صورت معادله (۵) همراه با چندجمله‌ای خطای $e(x) = e_0 + e_1x + \ldots + e_{n-1}x^{n-1}$ به‌صورت زیر خواهد بود:

$$r(X) = c(X) + e(X) \qquad (۵)$$

سپس فرآیند کدگشایی و تصحیح خطا توسط بلوک دیکدر BCH اجرا می‌شود.




شکل ۲: بلوک دیاگرام دیکدر **BCH**

مرحله‌ی اول: محاسبه‌ی سیندروم

اولین مرحله فرآیند کدگشایی یک کد BCH، محاسبه سیندروم با استفاده از بردار دریافتی $r(X)$ جهت تشخیص خطاست. همان‌طور که در بخش کدگذاری کدهای BCH ملاحظه شد، هر کلمه کد از حاصل‌جمع $X^{n-k}v(X)$ با باقی‌مانده آن نسبت به $g(x)$ به وجود می‌آید؛ پس خروجی بلوک اینکدر یک چندجمله‌ای است که باقی‌مانده‌ای نسبت به $g(x)$ ندارد؛ زیرا در هنگام کد کردن، باقی‌مانده $X^{n-k}v(X)$ به آن افزوده می‌شود. پس چندجمله‌ای حاصل بر $g(x)$ بخش‌پذیر است. از این ویژگی برای تشخیص وجود خطا استفاده می‌شود. درصورتی‌که داده‌های دریافتی بر چندجمله‌ای مولد بخش‌پذیر باشد، یعنی خطایی در آن رخ نداده است، امّا در غیراین‌صورت می‌توان نتیجه گرفت که کلمه کد دریافتی نامعتبر و حاوی خطاست و $2t$ مؤلفه سیندروم مطابق معادله (۶) محاسبه می‌شود.

$$S(X) = (S_1, S_2, \ldots, S_{2t})$$

$$S_i = r(\alpha^i) = e(\alpha^i) = \sum_{j=0}^{n-1} r_j \alpha^{ij} \qquad (۶)$$

$$1 \le i \le 2t$$

مرحله دوم: یافتن چندجمله‌ای خطایاب

دومین مرحله از فرآیند کدگشایی در بلوک دیکدر BCH یافتن چندجمله‌ای خطایاب $\sigma(X)$ مطابق معادله (۷) است. این مرحله از فرآیند کدگشایی با استفاده از الگوریتم‌های مختلفی انجام می‌شود که می‌توان آن‌ها را به سه گروه اصلی دسته‌بندی نمود: الگوریتم Paterson، الگوریتم Berlekamp-Massey (BM) و الگوریتم Eucliden.

$$\sigma(X) = \sigma_0 + \sigma_1X + \sigma_2X^2 + \ldots + \sigma_vX^v \qquad (۷)$$

باوجوداینکه الگوریتم Paterson ازلحاظ مفهومی ساختار ساده‌ای دارد، امّا با افزایش تعداد خطای قابل تصحیح ($t$) در یک کلمه کد، پیچیدگی محاسباتی آن افزایش می یابد. محاسبه دنباله‌ای از دترمینان ها با افزایش مقدار $t$ منجر به افزایش پیچیدگی محاسبات خواهد شد؛ به همین دلیل عمدتاً الگوریتم های کارآمدتری همچون الگوریتم BM و Euclid جهت یافتن چندجمله‌ای‌های خطایاب استفاده می‌شوند [۱۵]. در این مقاله از الگوریتم BM جهت یافتن چندجمله‌ای خطایاب استفاده شده است.

مرحله سوم: یافتن مکان وقوع خطا توسط الگوریتم **Chien search**

پس از یافتن چندجمله‌ای خطایاب $\sigma(X)$، مرحله بعدی پیدا کردن مکان وقوع خطا ست. ریشه‌های $\sigma(X)$ مقادیری هستند که اگر آن‌ها را معکوس کنیم و برحسب توان‌های $\alpha$ بنویسیم، توانشان مکان خطا را نشان می‌دهد. یک روش ساده جهت به دست آوردن ریشه‌های $\sigma(X)$ استفاده از الگوریتمی بنام Chien search است. ریشه‌های چندجمله‌ای خطایاب $\sigma(X)$، از عناصر میدان گالوای تعریف‌شده برای کد BCH موردنظر است. جهت به دست آوردن این ریشه‌ها کافی است که مقدار $\sigma(X)$ را به ازای عناصر میدان ارزیابی کنیم. هرکدام از مقادیری که منجر به نتیجه صفر شود ($\sigma(\alpha^i)=0$)، به‌عنوان یکی از ریشه‌های $\sigma(X)$ در نظر گرفته می‌شود.

مرحله چهارم: تصحیح خطا

در آخرین مرحله جهت به دست آوردن کلمه کد صحیح باید $c(X)$ یعنی کلمه کد حاوی خطا را با $e(X)$ جمع کنیم؛ به‌عبارت‌دیگر جهت تصحیح خطا باید خروجی حاصل از بلوک Chien search را با بیت‌های کلمه کد ذخیره‌شده در بافر FIFO[۹] در مکان‌های متناظر با آن‌ها جمع کنیم. بدین ترتیب بلوک تصحیح‌کننده خطا تنها از یک جمع‌کننده میدان متناهی گالوا که ساختار آن بر اساس گیت XOR است، تشکیل می‌شود.

$$V(X)=C(X)+e(X) \qquad (۸)$$

## ۳- معماری موازی بلوک دیکدر BCH

بلوک دیکدر BCH از محدودیت‌هایی که در معماری سریال وجود دارد، همچون افزایش زمان محاسبه رنج می‌برد، به‌ویژه در فناوری MLC Flash که داده‌ها در بلوک‌هایی که معمولاً ۵۱۲ بایت هستند، نوشته می‌شوند، این افزایش تأخیر چشمگیر خواهد بود. درنتیجه جهت غلبه بر چنین مشکلی روشی مؤثر مبتنی بر معماری موازی برای هر یک از زیر بلوک‌های تولید سیندروم و Chien search اتخاذ شده که منجر به کاهش میزان تأخیر زمانی و تسریع در انجام محاسبات می شود [۱۶]. در ادامه به معرفی معماری موازی هر یک از زیر بلوک های تولید سیندروم و Chien search می‌پردازیم.

معماری موازی بلوک تولید سیندروم: معماری $p$-parallel بلوک دیاگرام تولید سیندروم‌ها با استفاده از ضرب‌کننده میدان متناهی[۱۰] به تعداد $p$ و به دنبال آن یک جمع‌کننده میدان متناهی[۱۱] و یک رجیستر قابل پیاده‌سازی است. این ساختار، $p$ بیت ورودی را در یک پالس ساعت دریافت کرده و به‌طور همزمان محاسبات سیندروم را روی تمام بیت‌ها انجام می‌دهد. میزان تأخیر با اجرای موازی معماری $p$-parallel از $n$ پالس به $\lceil n/p \rceil$ پالس کاهش می‌یابد.

از طرفی با توجه به رابطه‌ای که بین سیندروم های زوج و فرد $S_{2i}=S_i^2$ در کدهای باینری BCH وجود دارد [۳]، محاسبه مستقیم سیندروم‌های زوج حذف‌شده و مؤلفه‌های زوج سیندروم به‌سادگی با استفاده از مؤلفه‌های فرد سیندروم و با استفاده از یک مدار مربع‌کننده که ساختار آن بر اساس تعدادی گیت XOR است، محاسبه می‌شوند و به میزان قابل‌توجهی از حجم سخت‌افزاری معماری موازی بلوک تولید سیندروم کاسته می‌شود [۱۷].

معماری موازی بلوک **Chien search**: جهت یافتن ریشه چندجمله‌ای خطایاب در معماری سریال بلوک Chien search، $n$ عنصر از میدان گالوا $GF(2^m)$ ارزیابی می‌شود، درنتیجه به $n$ پالس ساعت زمان نیاز دارد. در معماری موازی بلوک Chien search، $p$ عنصر $[\sigma(\alpha^i),\sigma(\alpha^{2i}),...,\sigma(\alpha^{pi})]$ از میدان گالوا $GF(2^m)$ به‌طور همزمان در یک پالس ساعت ارزیابی می‌شوند؛ درنتیجه تأخیر در محاسبات از $n$ پالس ساعت به $\lceil n/p \rceil$ پالس ساعت کاهش پیدا می‌کند. در معماری موازی، بلوک Chien search دقیقاً از $(p\times t)$ ضرب‌کننده میدان متناهی، $p$ جمع‌کننده $m$ بیتی میدان متناهی که دارای $t$ ورودی است، $p$ رجیستر $m$ بیتی و $p$ مالتی‌پلکسر $m$ بیتی، استفاده می‌کند، یادآور می‌شویم که $m$ و $t$ به ترتیب گسترش میدان گالوا $GF(2^m)$ و قابلیت تصحیح خطای کد BCH است [۱۸-۲۰].

## ۴- الگوریتم اشتراک‌گذاری **XOR** ها جهت کاهش حجم و پیچیدگی سخت‌افزار بلوک دیکدر **BCH**

توجه به این نکته ضروری است که تکنیک موازی نیازمند سخت‌افزار پیچیده و حجیم‌تری است؛ بنابراین در این قسمت از طراحی بلوک دیکدر BCH به دنبال کاهش حجم سخت‌افزار موردنیاز بلوک دیکدر هستیم. در میان سه بلوک دیکدر BCH، بلوک Chien search بیش‌ترین حجم سخت‌افزاری و پیچیدگی محاسباتی را دارا است، به‌طوری‌که حدوداً ۶۵ درصد از حجم کل بلوک دیکدر BCH را در سیستم‌های تصحیح خطا اشغال کرده است [۴، ۵]. الگوریتم Chien search مکان وقوع خطا را با ارزیابی $\sigma(\alpha^i)_{0\le i\le n-1}=0$ مشخص می‌کند. به‌طورکلی چندجمله‌ای خطایاب می‌تواند به‌صورت معادله (۹) بازنویسی شود.

$$\sigma(X)=\sigma_0+\sigma_1X+\sigma_2X^2+...+\sigma_tX^t$$
$$\sigma(\alpha^i)=\sigma_0+\sigma_1\alpha^i+\sigma_2\alpha^{2i}+...+\sigma_t\alpha^{ti} \qquad (۹)$$

از معادله (۹) می‌توان مشاهده کرد که جهت محاسبه مقدار آن به چندین ضرب‌کننده ثابت میدان متناهی (CFFM[12]) نیازمندیم. بنابراین بیش‌ترین عنصر به کار گرفته‌شده در ساختار داخلی بلوک Chien search ضرب‌کننده ثابت میدان متناهی (CFFM) است که منجر به افزایش پیچیدگی سخت‌افزاری مدار در پیاده‌سازی موازی خواهد بود. جهت کاهش این پیچیدگی، حاصل‌ضرب مضروب $m$ بیتی $\sigma_j$ که $(1 \le j \le t)$ در مضروب‌فیه را که به‌صورت یک ثابت $\alpha^{ij}_{1\le i\le p}$ است، به‌صورت معادله (۱۰) در نظر بگیرید.

$$\begin{aligned} P &= \alpha^{ij}\sigma_j \\ &= \alpha^{ij}(\sigma_{j,0} + \sigma_{j,1}\alpha + \ldots + \sigma_{j,m-1}\alpha^{m-1}) \end{aligned} \quad (۱۰)$$

جهت انجام عمل ضرب به‌جای استفاده از ضرب‌کننده میدان متناهی که شامل دو مرحله: ۱) ضرب چندجمله‌ای مضروب در مضروب‌فیه ۲) محاسبه‌یباقی‌مانده حاصل از تقسیم حاصل‌ضرب بر چندجمله‌ای اولیه میدان گالوا است، می‌توان از ضرب مسترویتو[13] استفاده نمود. مسترویتو با به‌کارگیری ضرب ماتریسی نوعی ضرب موازی جهت ادغام دو مرحله مذکور در ضرب‌کننده‌های میدان گالوا مطابق معادله (۱۱) ارائه داده است [۲۱].

$$\begin{aligned} P &= \alpha^{i}\sigma_j \\ &= \alpha^{i}(\sigma_{j,0} + \sigma_{j,1} + \ldots + \sigma_{j,m-1}) \\ &= \begin{bmatrix} \alpha_0^{i} & \alpha_0^{i+1} & \cdots & \alpha_0^{i+m-1} \\ \alpha_1^{i} & \alpha_1^{i+1} & \cdots & \alpha_1^{i+m-1} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_{m-1}^{i} & \alpha_{m-1}^{i+1} & \cdots & \alpha_{m-1}^{i+m-1} \end{bmatrix} \begin{bmatrix} \sigma_{j,0} \\ \sigma_{j,1} \\ \vdots \\ \sigma_{j,m-1} \end{bmatrix} \\ &= \left[a_{coeff}\right]_{m\times m} * \left[\sigma_j\right]_{m\times 1} \\ &= \left[P_{j,0} \quad P_{j,1} \quad P_{j,2} \quad \cdots \quad P_{j,m}\right]^T \end{aligned} \quad (۱۱)$$

ازآنجایی‌که عناصر $\alpha_l^{i+j}{}_{(0\le l,j\le(m-1))}$ به‌صورت اعداد باینری هستند و نیز عملگر جمع تحت میدان گالوا به‌صورت جمع پیمانه دو تعریف می‌شود، پیچیدگی محاسباتی ضرب‌کننده‌های ثابت میدان متناهی گالوا توسط تعدادی گیت XOR قابل‌اجرا است در ادامه و در جهت ساده‌سازی بیش‌تر و کاهش حجم سخت‌افزاری این عمل ضرب، از الگوریتم اشتراک‌گذاری XORها استفاده می‌شود.

این الگوریتم با یافتن زیر دنباله‌های تکراری مشترک، در جهت حذف محاسبات اضافی و کاهش تعداد XORها اجرا می‌شود. الگوریتم اشتراک‌گذاری XORها قادر به یافتن XORهای اضافی است که به دفعات تکرار شده‌اند. چنانچه بتوان جفت عناصری که به دفعات در ماتریس حاصل ضرب تکرار می شوند را از پیش محاسبه کرد، محاسبات اضافی حذف و حجم سخت‌افزار کاهش پیدا می‌کند. الگوریتم اشتراک‌گذاری XORها با اجرای ۴ مرحله بدین شرح قادر به حذف گیت‌های XOR اضافی است [۲۲، ۲۳].

- مرحله ۱: درایه‌های ۱ هر ستون $i$ و $j$ از ماتریس $a_{coffe}$ را بررسی کرده و ستون‌هایی که تعداد انطباق بین درایه‌های ۱ آن‌ها بیش‌تر از یک‌بار است انتخاب می‌شوند.
- مرحله ۲: بین ستون‌های انتخابی از مرحله ۱، آن دسته از ستون‌هایی که بیش‌ترین تعداد انطباق بین درایه‌های ۱ آن‌ها وجود دارد، به‌عنوان best-match انتخاب می‌شوند.
- مرحله ۳: با اضافه نمودن یک ستون در سمت راست ماتریس $a_{coffe}$، محاسبات تکراری ناشی از گیت XOR حذف خواهد شد.
- مرحله ۴: مراحل ۱ تا ۳ برای تمامی ستون‌های ماتریس $a_{coffe}$ تکرار شده تا جایی که تعداد انطباق بین درایه‌های ۱ هر دو ستون بیش‌تر از یک‌بار نباشد.

پس از پایان ۴ مرحله بهینه سازی ذکر شده در بالا، گیت XORهایی که به دفعات تکرار شده‌اند، انتخاب‌شده، از پیش محاسبه‌شده و بین اعمال ضرب جملات مختلف به اشتراک گذاشته می شوند. بدین ترتیب تعداد گیت‌های XOR و پیچیدگی سخت‌افزاری ضرب‌کننده ثابت میدان متناهی کاهش خواهد یافت. الگوریتم اشتراک‌گذاری XORها نه‌تنها می‌تواند به هر یک از ضرب‌کننده های ثابت میدان متناهی در بلوک Chien search اعمال شود، بلکه می‌تواند به گروهی از ضرب‌کننده‌ها که مضروب یکسانی ($\sigma_j$) دارند، نیز اعمال شده و به میزان قابل‌توجهی تعداد گیت‌های XOR را کاهش دهد.

در ادامه بـا یـک مثال اجرای این الگوریتم را برای گروهی از ضرب‌کننده‌ها بررسی می‌کنیم. چهار ضرب‌کننده ثابت میدان متناهی تحت میدان گالوا $GF(2^9)$ را به‌صورت $\sigma_5\alpha^{5}, \sigma_5\alpha^{10}, \sigma_5\alpha^{15}, \sigma_5\alpha^{20}$ در بلوک Chien search در نظر بگیرید. ازآنجایی‌که هر چهار ضرب‌کننده مضروب یکسانی $(\sigma_5 = B)$ دارند، الگوریتم اشتراک‌گذاری XORها می‌تواند به هر یک از ضرب‌کننده‌ها اعمال شود.

اگر حاصل‌ضرب چهار ضرب‌کننده ثابت میدان متناهی را مطابق معادله (۱۱) به ترتیب با $P_1$، $P_2$، $P_3$ و $P_4$ بدین ترتیب نشان دهیم:

$$P_1 = \alpha^5 B = \begin{bmatrix} b_5 \\ b_6 \\ b_7 \\ b_8 \\ b_0 + b_5 \\ b_1 + b_6 \\ b_2 + b_7 \\ b_3 + b_8 \\ b_4 \end{bmatrix} \quad (۱۲)$$

$$P_2 = \alpha^{10} B = \begin{bmatrix} b_0 + b_5 \\ b_1 + b_6 \\ b_2 + b_7 \\ b_2 + b_3 + b_8 \\ b_0 + b_4 + b_5 \\ b_1 + b_5 + b_6 \\ b_2 + b_6 \\ b_3 + b_7 + b_8 \\ b_4 + b_7 + b_8 \end{bmatrix} \quad (۱۳)$$

$$P_3 = \alpha^{15} B = \begin{bmatrix} b_0 + b_4 + b_5 \\ b_1 + b_5 + b_6 \\ b_2 + b_6 + b_7 \\ b_2 + b_3 + b_7 + b_8 \\ b_0 + b_5 + b_8 \\ b_0 + b_1 + b_6 \\ b_1 + b_7 \\ b_2 + b_3 + b_8 \\ b_2 + b_3 + b_4 \end{bmatrix} \quad (۱۴)$$

$$P_4 = \alpha^{20} B = \begin{bmatrix} b_0 + b_5 + b_8 \\ b_0 + b_1 + b_6 \\ b_1 + b_2 + b_7 \\ b_2 + b_3 + b_8 \\ b_0 + b_3 + b_4 + b_5 + b_8 \\ b_1 + b_4 + b_5 + b_6 \\ b_2 + b_5 + b_6 + b_7 \\ b_3 + b_6 + b_7 + b_8 \\ b_4 + b_7 + b_8 \end{bmatrix} \quad (۱۵)$$

با توجه به ماتریس حاصل ضرب معادلات بالا، تعداد گیت‌های XOR موردنیاز برای هر یک از چهار ضرب‌کننده به ترتیب برابر ۴، ۱۴، ۱۸ و ۲۳ عدد است. پس از اعمال الگوریتم اشتراک‌گذاری XORها به هر یک از ضرب‌کننده‌ها و جستجوی عبارات تکراری موجود، سه گروه معادله XOR به شرح زیر به دست می‌آید:

$$\begin{aligned} P_2 &: \; c_1 = b_7 + b_8; c_2 = b_0 + b_5; c_3 = b_1 + b_6 \\ P_3 &: \; c_1 = b_2 + b_3; c_2 = b_4 + b_5; c_3 = b_1 + b_6 \\ P_4 &: \; c_1 = b_7 + b_8; c_2 = b_0 + b_5; c_3 = b_1 + b_6 \\ & \quad c_4 = b_2 + b_7 \end{aligned} \quad (۱۶)$$

درنتیجه با اشتراک‌گذاری XORهای تکراری موجود در هر یک از ضرب‌کننده‌ها، تعداد گیت XOR موردنیاز برای $P_2$ از ۱۴ به ۷، برای $P_3$ از ۱۸ به ۱۱ و برای $P_4$ از ۲۳ به ۱۴ عدد کاهش پیدا می‌کند. امّا با جستجوی XORهای مشترک بین ماتریس ضرایب $P_1$ تا $P_4$ و اشتراک‌گذاری آن‌ها، ۱۵ معادله بدین شرح به دست می‌آید:

$$\begin{aligned} & c_1 = b_0 + b_5; \quad c_2 = b_1 + b_6; \quad c_3 = b_2 + b_7 \\ & c_4 = b_3 + b_8; \quad c_5 = b_2 + b_6; \quad c_6 = b_7 + b_8 \\ & c_7 = b_2 + b_3; \quad c_8 = c_1 + b_4; \quad c_9 = c_2 + b_5 \\ & c_{10} = b_4 + c_6; \quad c_{11} = c_6 + b_3; \quad c_{12} = c_1 + b_8 \\ & c_{13} = b_0 + c_2; \quad c_{14} = c_2 + b_4; \quad c_{15} = b_7 + c_5 \end{aligned} \quad (۱۷)$$

درنتیجه با استفاده از ۱۵ معادله به‌دست‌آمده در بالا، ماتریس ضرایب $P_1$ تا $P_4$ می‌تواند بدین ترتیب بازنویسی شود:

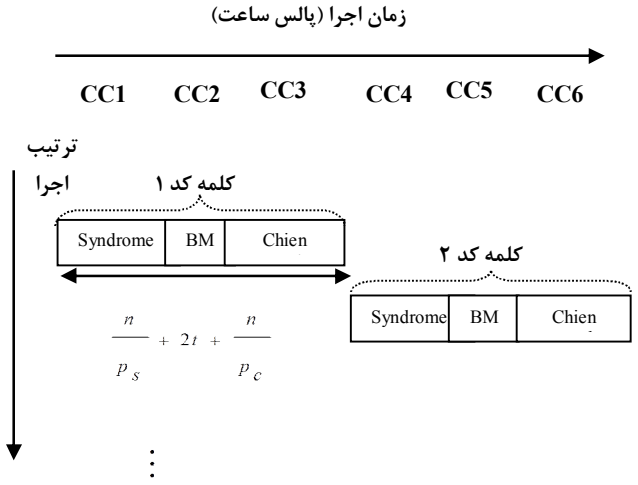
$$P_1 = \alpha^5 B = \begin{bmatrix} b_5 \\ b_6 \\ b_7 \\ b_8 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \\ b_4 \end{bmatrix} \quad ; \quad P_2 = \alpha^{10} B = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_{11} \\ c_8 \\ c_9 \\ c_5 \\ c_{11} \\ c_{10} \end{bmatrix}$$

$$P_3 = \alpha^{15} B = \begin{bmatrix} c_8 \\ c_9 \\ c_{15} \\ c_6 + c_7 \\ c_{11} \\ c_{13} \\ b_1 + b_7 \\ c_{14} \\ c_4 + c_7 \end{bmatrix} ; P_4 = \alpha^{20} B = \begin{bmatrix} c_8 \\ c_9 \\ c_{15} \\ c_6 + c_7 \\ c_{11} \\ c_{13} \\ b_1 + b_7 \\ c_{14} \\ c_4 + c_7 \end{bmatrix}$$

به‌عنوان یک نتیجه کلی می‌توان گفت که هر دو نوع الگوریتم، با اشتراک‌گذاری XORهای تکراری چه در هر یک از ضرب‌کننده‌ها و چه در گروهی از ضرب‌کننده‌ها می‌توانند به میزان قابل‌توجهی حجم سخت‌افزاری بلوک Chien search را کاهش دهد و درنتیجه منجر به کاهش حجم سخت‌افزاری بلوک دیکدر BCH شود.

## ۵- تکنیک خط لوله‌ای در مدار دیکدر **BCH** جهت افزایش بازدهی

به‌طورمعمول دیکدر BCH مبتنی بر معماری موازی بسته به طول یک کلمه کد، در $n/p_s$ پالس ساعت ($p_s$ فاکتور موازی‌سازی یا تعداد عناصری است که به‌طور همزمان در مدار سیندروم پردازش می‌شوند) به‌طور کامل آن را از ورودی دریافت کرده و در همین زمان مقادیر سیندروم را محاسبه می‌کند. سپس در $2t$ پالس ساعت الگوریتم BM را روی آن اجرا کرده و مقادیر ضرایب چندجمله‌ای خطایاب را به دست می‌آورد و درنهایت طی $n/p_c$ پالس ساعت ($p_c$ فاکتور موازی‌سازی یا تعداد عناصری است که به‌طور همزمان در بلوک Chien search پردازش می‌شوند) مکان وقوع خطا را با استفاده از الگوریتم Chien search محاسبه کرده و کلمه کد تصحیح‌شده را به همان ترتیب که از ورودی دریافت کرده بود، در خروجی خود قرار می‌دهد. بنابراین مقدار تأخیر فرآیند تصحیح خطا در بلوک دیکدر BCH معادل $n/p_s + 2t + n/p_c$ پالس ساعت است. آنچه بدیهی است این مدار بعد از دریافت هر کلمه کد باید به مدت $2t + n/p_c$ پالس ساعت روی آن محاسبات انجام دهد ($2t$ پالس برای BM و $n/p_c$ پالس برای الگوریتم Chien search) و در این مدت ورودی جدیدی دریافت نکند تا کلمه کد قبلی به‌طور کامل از مدار دیکدر خارج شود. به‌عبارت‌دیگر عملکرد مدار ازلحاظ زمانی مانند شکل ۳ خواهد بود.



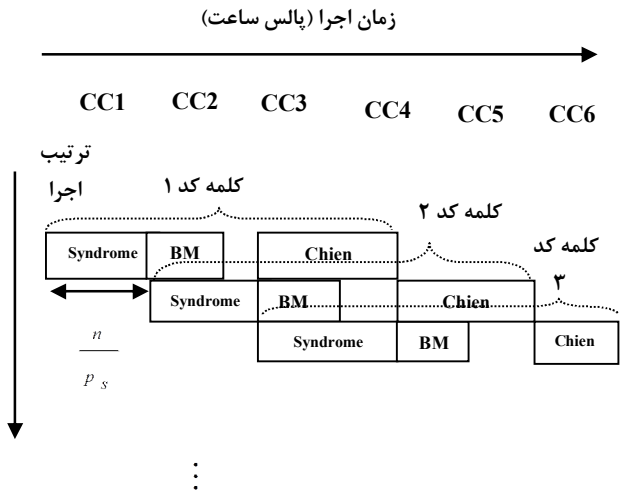

شکل۳: معماری غیر خطّ لوله‌ای مدار دیکدر **BCH**

جهت بهبود عملکرد مدار می‌توانیم از تکنیک خط لوله که در آن چندین دستورالعمل به‌طور همپوشان[۱۴] به اجرا درمی‌آیند [۲۴]، استفاده کنیم. درواقع معماری خط لوله از تعدادی مرحله (stage) تشکیل شده است که هر مرحله وظیفه خاصی به عهده دارد و می‌توانند همزمان عملیات خاص خود را انجام دهند. به‌عبارت‌دیگر می‌خواهیم هنگامی‌که پردازش روی یک کلمه کد در جریان است، ورودی مدار بتواند کلمه کد جدیدی را دریافت کند. نکته قابل‌توجه این است که برای دریافت ورودی به $n/p_s$ پالس ساعت و برای پردازش آن به $2t + n/p_c$ پالس ساعت نیاز داریم. پس نمی‌توان پردازش یک کلمه کد ورودی را با دریافت کلمه کد جدید به‌طور همزمان انجام داد. به عبارتی سخت‌افزار فی‌نفسه قادر به دریافت دو کلمه کد در بلوک سیندروم و اجرای محاسبات روی آن‌ها در یک سیکل واحد نیست؛ بنابراین باید ورودی را به‌اندازه $t$ ۲ پالس ساعت متوقف کنیم.

معماری خط لوله‌ای سه مرحله‌ای مدار دیکدر BCH در شکل ۴ نشان داده شده است که قادر به اجرای فرآیند دیکدینگ و تصحیح خطا روی سه کلمه کد به‌طور همزمان است. مطابق شکل ۴ عملکرد مدار سیندروم برای کلمه کد دوم با مدار BM برای کلمه کد اول همزمان شده است؛ همچنین عملکرد مدار سیندروم برای کلمه کد سوم با مدار BM برای کلمه کد دوم و مدار Chien search برای کلمه کد اول همزمان شده است. هر دو معماری در شکل‌های ۳ و ۴ مؤلفه‌های سخت‌افزاری یکسانی دارند. متوسط زمان اجرای فرآیند دیکدینگ بین دو کلمه کد از $n/p_s + 2t + n/p_c$ پالس ساعت در حالت غیر خط لوله به $n/p_s$ پالس ساعت در حالت خط لوله کاهش یافته است. نکته قابل‌توجه این است که تکنیک خط لوله در بلوک دیکدر BCH، زمانی سریع‌تر و مفیدتر است که

تعداد کلمه کدها زیاد باشند، چراکه مراحل لازم برای تکمیل فرآیند تصحیح خطا روی چند کلمه کد به‌طور موازی انجام شده و متوسط زمان اجرای فرآیند تصحیح خطا توسط بلوک دیکدر BCH بین چند کلمه کد کاهش می‌یابد.

در طراحی خط لوله طول هر سیکل ساعت باید بر اساس بدترین حالت ممکن برای هر مرحله تعیین شود، ولو آنکه برخی از مراحل بتوانند سریع‌تر کارشان را به پایان برسانند. با این اوصاف، در مدار دیکدر BCH با استفاده از تکنیک خط لوله هیچ‌گاه ورودی متوقف نمی‌شود و محاسبه سیندروم برای کلمه کد جدید همزمان با تصحیح خطا و خروج کلمه کد قبلی انجام می‌شود، اما در مدار دیکدر BCH فاقد خط لوله در هرلحظه تنها یک کار انجام می‌شود. به‌عنوان یک نتیجه کلی می‌توان گفت که وقتی با تعداد زیادی کلمه کد مواجه هستیم، توسل به شیوه خط لوله زمان انجام کل کار را به‌طور چشمگیری کاهش و ظرفیت گذردهی و بازدهی مدار دیکدر BCH را افزایش خواهد داد. در حافظه‌های فلش با استفاده تکنیک از خط لوله، مدار دیکدر BCH می‌تواند به‌طور همزمان به چند سکتور اعمال گردد و درنتیجه منجر به تسریع در فرآیند تصحیح خطا در حافظه‌های فلش شود.




شکل۴: معماری خط لوله‌ای مدار **BCH** دیکدر

## ۶- نتایج شبیه‌سازی

هر یک از روش‌های پیشنهادی جهت کاهش زمان و حجم سخت‌افزاری بلوک دیکدر BCH با استفاده از نرم‌افزار Xilinx ISE ۱۲٫۱ شبیه‌سازی و سنتز شده است، به‌طوری‌که طرح مذکور در ابتدا با ویراستار متن HDL، با زبان توصیف سخت‌افزار VHDL توصیف گردید و پس از کامپایل، برنامه VHDL موردنظر توسط ابزار برنامه‌ریزی FPGA، Xilinx ISE سنتز شده است که متناسب با آن، قطعات مدار مانند گیت‌ها، فلیپ‌فلاپ‌ها، جمع‌کننده‌ها و netlist مدار مشخص می‌شود. در این مقاله کد تصحیح خطای BCH(۲۷۴,۲۵۶) برای حافظه‌های NOR Flash که در آن یک صفحه به حجم ۲۵۶ بیت داده (۱۶ کلمه ۱۶ بیتی) به‌طور همزمان خوانده می شود، در نظر گرفته شده است. جدول ۲ پارامترهای کد BCH موردنظر را نشان می‌دهد.

جدول ۲: پارامترهای کد تصحیح خطای **BCH**

| پارامتر | مقادیر |
|---|---|
| $m$ | $۹[GF(2^9)]$ |
| $n$ | ۲۷۴ |
| $k$ | ۲۵۶ |
| $t$ | ۳ |
| تعداد بیت افزونگی | ۱۸ |
| $R = n/k$ (نرخ کد) | ۱/۱۰۷ |

جهت سنتز مدار در نرم‌افزار ISE، خانواده IC انتخابی، FPGA Spartan3 سری XC3S1000 است. نتایج حاصل از سنتز مدار اینکدر و دیکدر BCH در جدول ۳ نشان داده شده است.

جدول ۳: پارامترهای کد تصحیح خطای **BCH**

| Device Utilization Summary<br>Selected Device: FPGA Sparatan3 XC3S1000 | | | | |
|---|---|---|---|---|
| Logic Utilization | Used **اینکدر** | Utilization **اینکدر** | Used **دیکدر** | Utilization **دیکدر** |
| Number of Slices | ۱۱ out of ۱۹۲۰ | ٪۱ | ۴۱ out of ۱۹۲۰ | ٪۲ |
| Number of Slice Flip Flops | ۱۴ out of ۳۸۴۰ | ٪۱ | ۳۳ out of ۳۸۴۰ | ٪۱ |
| Number of 4 input LUTs | ۲۱ out of ۳۸۴۰ | ٪۱ | ۷۶ out of ۳۸۴۰ | ٪۱ |
| Number of bonded IOBs | ۱۴ out of ۱۷۳ | ٪۸ | ۵ out of ۱۷۳ | ٪۲ |
| Number of GCLKs | ۱ out of ۸ | ٪۱۲ | ۱ out of ۸ | ٪۱۲ |

جدول ۴ زمان موردنیاز جهت فرآیند اینکدینگ و دیکدینگ در هریک از بلوک‌های اینکدر و دیکدر BCH، فرکانس کاری، مقدار تأخیر مسیر بحرانی هر یک از زیر بلوک‌های دیکدر BCH و مقدار توان مصرفی کل مدار اینکدر و دیکدر BCH را که معادل مجموع توان فعال و نشتی مدار است و با استفاده از Power Compiler پس از سنتز محاسبه‌شده را نشان می‌دهد. مطابق جدول ۴ در بلوک دیکدر BCH حداقل زمان فرآیند دیکدینگ ۲/۵ نانوثانیه است و این مقدار معادل زمانی است که داده دریافتی فاقد هرگونه خطایی بوده و فرآیند کدگشایی در فاز اول بلوک تولید سیندروم (تشخیص خطا) خاتمه می‌یابد. متوسط زمان فرآیند دیکدینگ نشان‌دهنده زمانی است که تنها یک بیت خطا در داده دریافتی به وقوع پیوسته و معادل ۴/۳ نانوثانیه است و حداکثر زمان فرآیند دیکدینگ مساوی ۷/۹ نانوثانیه است که نشانگر زمانی است که بیش‌ترین خطا در داده دریافتی رخ داده که در این مقاله ۳ بیت در نظر

گرفته شـده اسـت. با توجه به جدول ۴ می‌توان ملاحظه نمود که زمان موردنیاز در فرآیند دیکدینگ مقدار ثابتی نیست و به تعداد خطای موجود در داده ذخیره شده و مکان وقوع آن بستگی دارد. تأخیر مسیر بحرانی در بلوک دیکدر BCH عمـدتاً نـاشـی از ضـرب‌کننده‌های ثابت میدان متناهی اسـت که مطابق جدول ۴ این مقدار تأخیر در زیر بلوک‌های Chien search و تولید چندجمله‌ای خطایاب بیش‌تر اسـت. از طرفی به کارگیری تکنیک خط لوله در مدار دیکدر BCH اسـتفاده از منابع سـخت‌افزاری و توان عملیاتی را افزایش می‌دهد و منجر به تسـریع دو برابری فرآیند دیکدینگ و تصـحیح خطا بین هر یک از کلمه کدها می شود؛ به‌طوری‌که در این مقاله با توجه به پارامترهای $n = ۲۷۴$ ،$p = ۴$ و $t = ۳$ متوسـط زمان فرآیند دیکدینگ بین دو کلمه کد از ۱۴۳ پالس ساعت به ۶۸ پالس کاهش می‌یابد.

جدول ۴: مشخصات اجرایی مدار اینکدر و دیکدر **BCH**

| پارامتر | مقادیر | |
|---|---|---|
| فاکتور موازی‌سازی ($p$) | ۴ | |
| زمان اینکدینگ | ۴/۳ $ns$ | |
| زمان دیکدینگ | ۲/۵ $ns$ (min) | بدون خطا |
| | ۴/ ۳ $ns$ (typ) | ۱ بیت خطا |
| | ۷/۹ $ns$ (max) | ۳ بیت خطا |
| فرکانس کاری | ۲۰۰ MHz | |
| مجموع توان مصرفی مدار اینکدر و دیکدر BCH (Active Power + Leakage Power) | ۷/۲۳ µW | ۱ بیت خطا |
| | ۸/۸۷ µW | ۲ بیت خطا |
| | ۱۱/۷ µW | ۳ بیت خطا |
| تأخیر مسیر بحرانی هر یک از زیر بلوک‌های مدار دیکدر BCH | بلوک تولید سیندروم $T_{mult}+T_{add}+T_{ff} =$ ۱/۵ $ns$ | |
| | بلوک تولید چندجمله‌ای خطایاب $T_{mux}+T_{mult}+T_{add}+T_{ff} =$ ۳/۷ $ns$ | |
| | بلوک Chien search $T_{mux}+T_{mult}+T_{add} =$ ۴/۲ $ns$ | |

نتایج حاصل از به‌کارگیری الگوریتم اشتراک‌گذاری گیت‌های XOR جهت کاهش تعداد این گیت‌ها در معماری موازی بلوک Chien search در مدار دیکدر BCH(۲۷۴, ۲۵۶) در جدول ۵ نشـان داده شـده اسـت. نتایج حاصـل، کاهش قابل‌توجه تعداد گیت‌های XOR را در معماری موازی بلوک Chien search پس از اعمال الگوریتم اشتراک‌گذاری نشان می‌دهد، به‌طوری‌که میزان کاهش کل پس از اشتراک‌گذاری XORها در هر یک از ضرب‌کننده‌های ثابت میدان متناهی به ۲۳٪ می‌رسد. این در حالی اسـت که این میزان کاهش پس از اشـتراک‌گذاری XORها در گروهی از ضرب‌کننده‌های ثابت میدان متناهی به ۴۴٪ می‌رسد، بنابراین الگوریتم اشـتراک‌گذاری می‌توانـد منجر بـه کـاهش قـابل‌توجهی در پیچیدگی سخت‌افزاری بلوک دیکدر BCH شود.

جدول ۵: نتایج حاصل از الگوریتم اشتراک‌گذاری **XOR** در بلوک **Chien search**

| میزان بهبود | تعداد گیت XOR | نوع معماری بلوک Chien Search |
|---|---|---|
| - | ۱۰۲ | معماری سریال |
| - | ۵۵۴ | معماری موازی |
| ٪ ۲۳ | ۴۲۴ | معماری موازی مبتنی بر اشتراک‌گذاری گیت‌های XOR در هر یک از ضرب‌کننده‌ها |
| ٪ ۴۴ | ۳۰۹ | معماری موازی مبتنی بر اشتراک‌گذاری گیت‌های XOR بین ۴ گروه ضرب‌کننده |

بلوک اینکدر جهت تولید بیت افزونگی توسط مدار LFSR به زمانی معادل ۴/۳ نانوثانیه نیاز دارد، این در حالی است که مطابق مشخصه‌ی زمانی یک حافظه‌ی NOR Flash که مقادیر آن در جدول ۶ آمده است، حداکثر زمان ممکن جهت نوشتن ۳۲ بایت (۲۵۶ بیت) داده در یک صفحه از حافظه معادل ۱۲۸ میکروثانیه است، بنابراین بلوک اینکدر BCH پیاده‌سازی شده در این پروژه بر اساس ساختار LFSR برای حافظه‌های NOR Flash مناسب خواهد بود. از طرفی جهت دسترسی به داده‌ی ذخیره‌شده در حافظه، در بدترین حالت که بیش‌ترین خطا در داده‌ی دریافتی رخ داده است، به زمانی معادل ۷/۹ نانوثانیه نیازمندیم که معادل مجموع زمان‌های موردنیاز برای هر یک از زیر بلوک‌های دیکدر BCH است، این مقدار زمانی هنوز کم‌تر از حداکثر تأخیر زمانی ممکن جهت دستیابی ترتیبی یا تصادفی در یک حافظه‌ی NOR Flash مطابق جدول ۶ است؛ بنابراین طبق نتایج به‌دست‌آمده بلوک دیکدر BCH که در این پروژه بر اساس ساختار موازی دو زیر بلوک تولید سیندروم و Chien search و نیز به‌کارگیری الگوریتم BM جهت تولید چندجمله‌ای خطایاب پیاده‌سازی شده است، در ساختار داخلی کنترل‌کننده حافظه‌های NOR Flash مناسب خواهد بود و فرآیند کدگذاری و کدگشایی جهت تصحیح خطا در سیکل نوشتن و خواندن اطلاعات در کارکرد حافظه خللی ایجاد نخواهد کرد.

جدول ۶: مشخصه زمانی حافظه **NOR Flash** [۲۵]

| مشخصه‌ی زمانی حافظه NOR Flash | مقادیر |
|---|---|
| زمان نوشتن داده‌ها | ۱۲۸ $\mu s$/ (۲۵۶ بیت، ۳۲ بایت) |
| زمان پاک کردن داده‌ها | ۱ s/ (۱۲۸ کیلوبایت) |
| زمان دستیابی تصادفی به داده‌ها | ۰/۷۵ $\mu s$ |
| پهنای باند نوشتن | ۰/۲۵ MB/s |

جهت رو شن شدن عملکرد دیکدر پیشنهادی، دیکدر طراحی شده با قابلیت تصـحیح ۳ بیت خطا جهت ذخیره‌سـازی مطمئن ۲۵۶ بیت اطلاعات برای حافظه NOR Flash در این مقاله با سـایر روش‌های ارائه شـده مطابق جدول ۷ مقایسه شده است. همان‌طور که ملاحظه می‌شـود بهبود پیچیدگی سـخت‌افزاری دیکدر BCH از حـیث تعداد گیت‌هـای XOR موردنیاز در روش پیشـنهادی نسـبت به روش

[۱۲، ۱۳، ۲۶] بسیار چشم‌گیرتر بوده است، به‌طوری‌که درمجموع ۲۴۰۰ گیت XOR در کل مدار دیکدر BCH به کار گرفته شده است که نصف تعداد XORهای ذکرشده در هر یک از دیکدرهای [۱۲، ۱۳، ۲۶] مطابق جدول ۷ است و مؤید برتری دیکدر پیشنهادی از حیث حجم سخت‌افزاری است. ازلحاظ تأخیر زمانی در فرآیند دیکدینگ، دیکدر ارائه‌شده در این مقاله سریع‌تر از دیکدر [۱۳، ۲۶] عمل می‌کند اما در مقایسه با دیکدر [۱۲] مشاهده می شود که کندتر عمل می‌کند، یکی از دلایل کند بودن دیکدر ارائه‌شده در این مقاله در مقایسه با دیکدر [۱۲] استفاده از الگوریتم BM در زیر بلوک تولید چندجمله‌ای خطایاب است. الگوریتم BM ازلحاظ سخت‌افزاری دارای پیچیدگی کم‌تری است، اما به علت وجود عمل تقسیم در آنکه از عملیات وقت‌گیر در میدان متناهی گالوا محسوب می‌شود، کندتر است. به‌عنوان یک نتیجه کلی می‌توان اذعان داشت که دیکدر BCH پیشنهادی در این مقاله علاوه بر کاهش زمان فرآیند کدگشایی جهت تصحیح خطا ازلحاظ کاهش حجم سخت‌افزاری موفق‌تر از سایر روش‌ها عمل کرده ست.

جدول ۷: نتایج مقایسه بین دیکدر **BCH** پیشنهادی و سایر روش‌ها

| پارامتر | دیکدرBCH پیشنهادی | دیکدرBCH [۱۲] | دیکدرBCH [۱۳] | دیکدرBCH [۲۶] |
|---|---|---|---|---|
| نوع پیاده‌سازی | Synthesis FPGA Spartan 3 XC3S1000 | Post-Lyout Technology 90 nm CMOS | Synthesis Technology 180 nm CMOS | Post-Lyout Technolog 90 nm CMOS |
| BCH $(n, k, t)$ | BCH (۲۷۹، ۲۵۶، ۳) | BCH (۲۷۹، ۲۵۶، ۳) | BCH (۲۷۹، ۲۵۶، ۲) | BCH (۲۵۵، ۲۳۹، ۲) |
| تعداد گیت XOR | ۲/۴ K | ۴/۱ K | N/A | ۴/۲ K |
| تأخیر | ۴/۱ *ns* (۱۴۳ clock) | ۲/۵ *ns* | ۴/۵۱ *ns* | ۶/۸ *ns* (۲۷۲ clock) |

## ۷- نتیجه‌گیری

در این مقاله به بررسی کدهای BCH و کدگذاری و کدگشایی این نوع کدها در حافظه‌های فلش پرداختیم. با توجه به این‌که قابلیت اطمینان در ذخیره‌سازی به‌عنوان چالشی مهم در نسل جدید حافظه‌های فلش با سلول چندسطحی، مطرح است، افزایش ظرفیت ذخیره‌سازی که با افزایش تعداد سطوح ذخیره‌سازی در هر سلول به وجود آمده تا حد زیادی منجر به کاهش ذخیره‌سازی مطمئن داده شده است. همان‌طور که پیش‌تر نیز اشاره کردیم، حافظه‌های فلش از کدهای تصحیح خطا جهت حصول اطمینان از یکپارچگی ذخیره‌سازی داده و تشخیص خطا و تصحیح آن‌ها استفاده می‌کنند. کدهای تصحیح خطای BCH به‌طور گسترده‌ای در واحد کنترل‌کننده حافظه‌های فلش استفاده می‌شوند و الگوریتم‌های متنوعی جهت کدگذاری و کدگشایی داده در این نوع حافظه‌ها پیشنهاد شده است.

دو نکته اساسی که در فرآیند کدگشایی داده در این نوع حافظه‌ها وجود دارد؛ افزایش سرعت محاسبات و کاهش میزان پیچیدگی سخت‌افزاری مدار دیکدر است. فرآیند کدگشایی توسط بلوک دیکدر BCH جهت تصحیح خطا شامل چهار مرحله: محاسبه سیندروم، محاسبه چندجمله‌ای خطایاب، الگوریتم Chein search جهت یافتن مکان وقوع خطا و تصحیح خطا است. در این مقاله، به‌کارگیری معماری موازی جهت کاهش تأخیر نا شی از محا سبات در هر یک از زیر بلوک‌های تولید سیندروم و Chien search و همچنین استفاده از الگوریتم اشتراک‌گذاری گیت‌های XOR جهت حذف گیت‌های تکراری در بلوک Chien search ارائه شده است که در بهبود کارایی و پیچیدگی سخت‌افزاری مدار دیکدر BCH مؤثر است. از طرف دیگر نشان داده شد که به‌کارگیری تکنیک خط لوله در مدار دیکدر BCH استفاده از منابع سخت‌افزاری و توان عملیاتی را افزایش داده و منجر به تسریع دو برابری فرآیند دیکدینگ و تصحیح خطا بین هر یک از کلمه کدها می شود. نتایج حا صل از شبیه سازی با ا ستفاده از زبان تو صیف سخت‌افزار VHDL و همچنین سنتز با استفاده از نرم‌افزار XILINX ISE و مقایسه آن‌ها با سایر روش‌های موجود بهبودهای پیش‌بینی‌شده توسط معماری‌ها و تکنیک‌های به کار گرفته‌شده را تأیید می‌کند.

## مراجع


[1] K. Kim, "Future memory technology: challenges and opportunities," *Proceedings of International Symposium on VLSI Technology, Systems and Applications*, pp. 5-9, 2008.

[2] K. Pangai, "90 nm Multi-level-cell flash memory technology," *IEEE International Symposium on Semiconductor Manufacturing*, pp. 197-199, 2010.

[3] S. Lin and D. J. Costello, *Error Control Coding: Fundamentals and Applications*, Prentice- Hall Inc, 2004.

[4] J. Cho and W. Sung, "Strength-reduced parallel chien search architecture for strong BCH codes," *IEEE Transaction on Circuits and Systems*, vol. 55, no. 5, pp. 427-431, 2008.

[5] Y. M. Lin, C. H. Yang, C. H. Hsu, H. C. Chang and C. Y. Lee, "A MPCN-based parallel architecture in BCH decoders for NAND flash memory devices," *IEEE Transaction on Circuits and Systems*, vol. 58, no. 10, pp. 682-686, 2011.

[6] H. Yoo, Y. Lee and I. C, Park, "7.3 Gb/s universal BCH encoder and decoder for SSD controllers," *Design Automation Conference (ASP-DAC)*, pp. 37-38, 2014.

[7] Y. Lee, H. Yoo, I. Yoo and I. C. Park, "High-throughput and low-complexity BCH decoding architecture for solid-state drives," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 5, pp. 1183–1187, 2014.

[8] M. Yin, M. Wie and B. Yi, "Optimized algorithm for binary BCH codes," *Circuits and Systems (ISCAS) IEEE Internationa Symposium*, pp. 1552-1555, 2013.

[9] D. V. Sarwate and R. S. Shanbhag, "High-speed architecture for BCH decoders," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 9, no. 5, pp. 641–655, 2001.

[10] H. Choi, W. Liu and W. Sung, "VLSI implementation of BCH error correction for multilevel cell NAND flash memory," *IEEE Transactions on Very Large Scale*

*Integration (VLSI) Systems*, vol. 18, no. 5, pp. 843–847, 2010.

[11] T. Chen, Y. Hsiao and Y. Hsing, "An adaptive-rate error correction scheme for NAND flash memory," 27th IEEE VLSI Test Symposium, pp. 53-58, 2009.

[12] C. Chu, Y. Lin, C. Yang and H. Chang, "A fully parallel BCH codec with double error correcting capability for NOR flash applications," in *IEEE International Conference of Acoustics, Speech, and Signal Processing*, pp. 1605–1608, 2012.

[13] X. Wang, P. Liyang, W. Dong and H. Chaohong, "A high-speed two cell BCH decoder for error correcting in mlc NOR flash memories," *IEEE Transactions on Circuits and Systems*, vol. 56, no. 11, pp. 865 – 869, 2009.

[14] V. Mahadevaswamy, S. Sunitha and BN. Shobha, "Implementation of fault tolerant method using BCH code on FPGA," *International Journal of Soft Computing and Engineering (IJSCE)*, vol. 2, no. 4, pp. 2231-2307, 2012.

[15] Y. Jiang, *A Practical Guide to Error-Control Coding Using MATLAB*, Artech House, 2010.

[16] F. Sun, S. Devarajan, K. Rose and T. Zhang, "Design of on-chip error correction systems for multilevel NOR and NAND flash memories," *IET Circuits, Devices and Systems*, vol. 1, no. 3, pp. 241-249, 2007.

[17] Y. Chen and K.K. Parhi, "Area efficient syndrome calculation for strong BCH decoding," *Electronics Letters*, vol. 47, no. 2, pp. 107-108, 2011.

[18] W. Liu, J. Rho and W. Sung, "Low-power high-throughput BCH error correction VLSI design for multi-level cell NAND flash memories," *IEEE Workshop on Signal Processing Systems Design and Implementation*, pp. 303-308, 2006.

[19] L. Song, M. Yu and M. S. Shaffer, "10- and 40-Gb/s Forward error correction devices for optical communications," *IEEE Journal of Solid-State Circuits*, vol. 37, no. 11, pp. 1565-1573, 2002.

[20] H. Lee, "High-speed VLSI architecture for parallel reed-solomon decoder," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 11, no. 2, pp. 288–294, 2003.

[21] M. Edoardo, *VLSI Architecture for Computations in Galois Fields,* Ph.D. Thesis, University of Linkoping, Sweden, WA, 2006.

[22] C. Paar, "Optimized arithmetic for reed-solomon encoders," in Proceedings of IEEE International Symposium Information Theory, 250-250, 1997.

[23] M. Potkonjak, M. Srivastava and A. Chandrakasan, "Multiple constant multiplications: efficient and versatile framework and algorithms for exploring common subexpression elimination," *IEEE Transactions on Computer-Aided Design*, vol. 15, pp. 151-165, 1996.

[24] J. Hennessy and D. Patterson, *Computer Architecture a Quantitative Approach*, Morgan Kaufmann Publishers, 2007.

[25] Micron, *Technical Note: Parallel NOR Flash Embedded Memory*.

[26] Y. M. Lin, H. C. Chang and C. Y. Lee, "Improve high code-rate soft BCH decoder architecure with one extra error compensation," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, no. 11, pp. 2160–2164, 2013.

زیرنویس‌ها

[1] Electrically Erasable Programmable Read-Only Memory
[2] Solid State Drive (SSD) (EEPROM)
[3] Multi Level Cell (MLC)
[4] Single Level Cell (SLC)
[5] Error Correction Code (ECC)
[6] Bose-Chaudhuri-Hocqunghem (BCH)
[7] Linear Feedback Shift Register (LFSR)
[8] program/erase
[9] First In First Out (FIFO)
[10] Finite Field Adder (FFA)
[11] Finite Field Multipliers (FFM)
[12] Constant Finite Field Multiplier (CFFM)
[13] Mastrovito
[14] Overlapped