TOBB EKONOMİ VE TEKNOLOJİ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

GERÇEK DRAM AYGITLARINDA DÖRTLÜ ETKİNLEŞTİRME İLE YÜKSEK HIZDA GERÇEK RASTGELE SAYI ÜRETİLMESİ

YÜKSEK LİSANS TEZİ

Ataberk OLGUN

Bilgisayar Mühendisliği Anabilim Dalı

Tez Danışmanı: Prof. Dr. Oğuz ERGİN

EKİM 2021

TEZ BİLDİRİMİ

Tez içindeki bütün bilgilerin etik davranış ve akademik kurallar çerçevesinde elde edilerek sunulduğunu, alıntı yapılan kaynaklara eksiksiz atıf yapıldığını, referansların tam olarak belirtildiğini ve ayrıca bu tezin TOBB ETÜ Fen Bilimleri Enstitüsü tez yazım kurallarına uygun olarak hazırlandığını bildiririm.

Ataberk OLGUN

ÖZET

Yüksek Lisans Tezi

GERÇEK DRAM AYGITLARINDA DÖRTLÜ ETKİNLEŞTİRME İLE YÜKSEK HIZDA GERÇEK RASTGELE SAYI ÜRETİLMESİ

Ataberk OLGUN

TOBB Ekonomi ve Teknoloji Üniversitesi
Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı

Tez Danışmanı: Prof. Dr. Oğuz ERGİN

Tarih: EKİM 2021

Gerçek rastgele sayı üreteçleri (TRNG), güvenlik açısından kritik kriptografik uygulamalar, bilimsel benzetim ve yapay öğrenme uygulamaları gibi çeşitli alanlarda, çok miktarda rastgele sayı oluşturmak üzere kullanılmaktadır. Ancak her bilgisayar sistemi bu tür uygulamalar için güvenlik garantilerini sağlayan özel TRNG donanımı ile donatılmamıştır. Bu tür sistemlerin uygulama alanını genişletmek ve özel TRNG donanımına sahip olmayan bilgisayar sistemlerinin büyük çoğunluğu için güvenlik garantilerini sağlamak için QUAC-TRNG'yi geliştirmekteyiz.

QUAC-TRNG, dikkatle tasarlanmış bir DRAM komut dizisinin art arda dört ardışık DRAM satırını etkinleştirmesi gözleminden yararlanmaktadır. Dörtlü Etkinleştirme (QUAC) birbirine zıt verilerin saklandığı dört satır etkinleştirildiğinde DRAM bit hattı gerilimindeki net sapmayı güvenli algılama sınırlarının üzerine çıkaramadığından, bit hattı algılama yükselteçlerinin bit hattındaki değerleri rastgele değerlere yakınsamasına neden olmaktadır.

QUAC'ın güvenilir bir şekilde gerçek rastgele değerler ürettiğini, bir DRAM üreticisinin 136 DDR4 DRAM yongasını kullanarak deneysel olarak göstermekteyiz. QUAC'a dayalı etkili bir TRNG'nin (QUAC-TRNG) nasıl geliştirileceğini açıklamaktayız. QUAC-TRNG'nin niteliğini NIST STS kullanarak değerlendirmekte ve QUAC-TRNG'nin her testi başarıyla geçtiğini göstermekteyiz. Deneysel değerlendirmelerimiz, QUAC-TRNG'nin 3,44 Gb/s (DRAM kanalı başına) hızla gerçek rastgele sayılar ürettiğini, en hızlı DRAM-tabanlı TRNG'lerin temel ve geliştirilmiş sürümlerinden sırasıyla 15,08 ve 1,41 kat daha iyi başarıma sahip olduğunu göstermektedir. QUAC-TRNG'nin, DRAM veri yolu frekansı 12 GT/s'ye eriştiğinde, en hızlı DRAM-tabanlı

TRNG'nin geliştirilmiş sürümünden 2,03 kat daha hızlı olduğunu, DRAM bant genişliğini son teknolojiden daha iyi kullandığını göstermekteyiz.

Anahtar Kelimeler: Bellek, DRAM, Gerçek rastgele sayı.

ABSTRACT

Master of Science

HIGH THROUGHPUT TRUE RANDOM NUMBER GENERATION USING QUADRUPLE ROW ACTIVATION IN REAL DRAM CHIPS

Ataberk OLGUN

TOBB University of Economics and Technology
Institute of Natural and Applied Sciences
Department of Computer Engineering

Supervisor: Prof. Dr. Oğuz ERGİN

Date: OCTOBER 2021

True random number generators (TRNG) sample random physical processes to create large amounts of random numbers for various use cases, including security-critical cryptographic primitives, scientific simulations, machine learning applications, and even recreational entertainment. Unfortunately, not every computing system is equipped with dedicated TRNG hardware, limiting the application space and security guarantees for such systems. To open the application space and enable security guarantees for the overwhelming majority of computing systems that do not necessarily have dedicated TRNG hardware, we develop QUAC-TRNG.

QUAC-TRNG exploits the new observation that a carefully-engineered sequence of DRAM commands activates four consecutive DRAM rows in rapid succession. This QUadruple ACtivation (QUAC) causes the bitline sense amplifiers to non-deterministically converge to random values when we activate four rows that store conflicting data because the net deviation in bitline voltage fails to meet reliable sensing margins.

We experimentally demonstrate that QUAC reliably generates random values across 136 commodity DDR4 DRAM chips from one major DRAM manufacturer. We describe how to develop an effective TRNG (QUAC-TRNG) based on QUAC. We evaluate the quality of our TRNG using NIST STS and find that QUAC-TRNG successfully passes each test. Our experimental evaluations show that QUAC-TRNG generates true random numbers with a throughput of 3.44 Gb/s (per DRAM channel), outperforming the state-of-the-art DRAM-based TRNG by 15.08x and 1.41x for basic

and throughput-optimized versions, respectively. We show that QUAC-TRNG utilizes DRAM bandwidth better than the state-of-the-art, achieving up to 2.03x the throughput of a throughput-optimized baseline when scaling bus frequencies to 12 GT/s.

Keywords: Memory, DRAM, True random number.

TEŞEKKÜR

Çalışmalarım boyunca değerli yardım ve katkılarıyla beni yönlendiren hocam Prof. Dr. Oğuz Ergin'e, kıymetli tecrübelerinden faydalandığım TOBB Ekonomi ve Teknoloji Üniversitesi Bilgisayar Mühendisliği Bölümü öğretim üyelerine, sundukları üretken ve keyifli çalışma ortamı için Kasırga Mikroişlemciler Laboratuvarı'ndaki çalışma arkadaşlarıma, destekleriyle her zaman yanımda olan annem Nilgün Olgun ve babam Selami Olgun'a ve tezimi dikkatlice okuyup düzenlememe yardım eden Şevval İzmirli'ye teşekkürlerimi sunarım.

İÇİNDEKİLER

ŞEKİL LİSTESİ

Sayfa

ÇİZELGE LİSTESİ

Sayfa

KISALTMALAR

DRAM : Dynamic Random Access Memory
TRNG : True Random Number Generator
GRSÜ : Gerçek Rastgele Sayı Üreteci
SHA : Secure Hash Algorithm

SEMBOL LİSTESİ

Bu çalışmada kullanılmış olan simgeler açıklamaları ile birlikte aşağıda sunulmuştur.

Simgeler	**Açıklama**
Gb/s	gigabit saniye (aktarım hızı)
Mb/s	megabit saniye (aktarım hızı)
MT/s	mega transfer saniye (aktarım hızı)
ns	nanosaniye
μs	mikrosaniye
ms	milisaniye
V_{DD}	besleme gerilimi
V_{th}	eşik gerilimi
$tRAS$	DRAM satır gecikmesi
tRP	DRAM önyükleme gecikmesi
$tRRD$	DRAM sütun komutu gecikmesi

1. GİRİŞ

Gerçek rastgele sayılar, kriptografi, bilimsel benzetim ve yapay öğrenme dahil olmak üzere çok çeşitli uygulamalarda kullanılmaktadır [1–20]. Bu uygulamalar genellikle çalışma koşullarındaki (örn. sıcaklık ve gerilim dalgalanmaları) değişikliklere karşı dirençli, yüksek hızlı bir gerçek rastgele sayı üreteci (GRSÜ, *-ing. TRNG*) gerektirmektedir [21].

Ne yazık ki tüm bilgisayar sistemleri bu tür uygulamaları etkin bir şekilde çalıştırma kabiliyetini sağlayan özel *GRSÜ* donanımı ile donatılmamaktadır. Bu sorunu çözmek için birçok çalışma bugün çoğu sistemde bulunabilen donanım bileşenlerini kullanarak gerçek rastgele sayı üreteçleri sağlamaya çalışmaktadır (örneğin, DRAM [22–26] ve SRAM [27–29]).

Gerçek rastgele sayılar üretmek için entropi kaynağı olarak DRAM'i kullanmak, yüksek performanslı sunucular, düşük güçlü uç aygıtlar ve bellek içinde hesaplama yapan sistemler [30] gibi birçok sisteme gerçek rastgele sayı üretme yeteneği kazandırmak için, DRAM'in ana bellek olarak yaygın bir şekilde benimsenmesinden ötürü umut verici bir yaklaşımdır. Son teknoloji DRAM-tabanlı GRSÜler ya (i) temel olarak yavaş gerçekleşen fiziksel süreçlerden faydalanarak rastgele sayı ürettiklerinden yüksek gecikmelere sahiptirler (örneğin, tutma hatası [25, 31–33], DRAM başlangıç değerleri [34]), ya da (ii) özel olarak seçilmiş DRAM satırlarının küçük bir kısmını entropi elde etmek için kullandıklarından düşük hızda çalışmaktadırlar (*tRCD* hataları-tabanlı [22], *tRP* hataları-tabanlı [24]).

Bu tezde bahsi geçen çalışmamızdaki amacımız gerçek DRAM aygıtlarında desteklenen, hızlı (birim zamanda çok rastgele sayı) ve düşük gecikmeli (ilk rastgele sayının üretilmesi için gereken zaman kısa) bir GRSÜ geliştirmektir. Bu amaç doğrultusunda gerçek DRAM aygıtlarında (SK Hynix tarafından üretilmiş) dikkatlice tasarlanmış DRAM komut dizilerini yürüterek yeni gözlemlediğimiz dört ardışık DRAM satırının peş peşe açılmasını sağlayan Dörtlü Etkinleştirme (QUAC)[1] davranışından yararlanmaktayız.

Çalışmamızın ana fikri QUAC işlemlerini düşük gecikmeli ve yüksek hızlı DRAM-tabanlı gerçek rastgele sayı üretimi için kullanmaktır. Birbirine zıt veriler saklayan DRAM satırları (örneğin, iki satırda '0' ve diğer iki satırda '1') QUAC ile açıldığında, DRAM bit hattındaki net gerilim güvenilir eşiğin üstüne çıkamadığından, DRAM bit hattı algılama yükselteçleri bit hattı üzerindeki gerilimi rastgele bir şekilde '0' veya '1' değerlerine yakınsatmaktadır. QUAC işlemlerini kullanarak çok sayıda bit hattı algılama yükseltecinde yarı-kararlı bir durum oluşturulabilir, bu da düşük gecikmeli ve yüksek hızda gerçek rastgele sayı üretmek için kullanılabilir.

[1] Tezin devamında, tezden üretilen yayında kullandığımız gibi, Dörtlü Etkinleştirme işlemlerinden bahsederken QUAC (*-ing. Quadruple Activation*) kısaltmasını kullanmaktayız.

Tüm bilgisayar sistemlerinde yaygın bir şekilde kullanılan DRAM aygıtlarını kullanarak hem birim zamanda çok sayıda rastgele sayı (yüksek hız) üretmek hem de ilk rastgele sayının üretilmesi için gereken zamanı en az tutmak (düşük gecikme) için QUAC-TRNG'yi geliştirmekteyiz. QUAC-TRNG ardışık olarak QUAC işlemleri uygulamakta ve bu işlemlerin sonuçlarını bir kriptografik özet işlevine sokarak gerçek rastgele sayı üretmektedir. QUAC-TRNG ile rastgele sayı üretilmesi (bir yineleme) beş temel adımda gerçekleşmektedir: QUAC-TRNG (i) birlikte açılacak dört DRAM satırını belirler, (ii) bu satırları birbirine zıt veriler ile doldurur, (iii) standart DRAM komutları kullanarak bir QUAC işlemi gerçekleştirir, (iv) işlemin sonuçlarını DRAM algılama yükselteçlerinden okur, (v) SHA-256 kriptografik özet işlevini kullanarak QUAC sonuçlarını işler ve 256-bit boyutunda bir gerçek rastgele sayıyı çıktı olarak verir.

136 DRAM aygıtı kullanarak gerçekleştirdiğimiz deneylerimizin sonucunda QUAC-TRNG'nin bir yinelemede ortalama 7664 bit gerçek rastgele sayı ürettiğini ve her yinelemenin ortalama 1940 ns sürdüğünü gözlemlemekteyiz. Var olan DRAM-tabanlı GRSÜler [22–26, 34, 35] ile kıyaslandığında QUAC-TRNG (i) daha düşük gecikme ile rastgele sayı üretmekte ve (ii) daha hızlı rastgele sayı üretmektedir. Ayrıca QUAC-TRNG'nin güvenilirliğini (ürettiği rastgele sayıların istatistiksel olarak ne kadar nitelikli olduğunu) NIST istatistiksel test paketinin (NIST STS [36]) tüm testlerini geçtiğini göstererek ölçmekteyiz.

QUAC-TRNG'nin ve var olan iki son teknoloji DRAM-tabanlı GRSÜlerin [22, 24] başarımını karşılaştırmaktayız. Son teknoloji çalışmaları iki farklı yapılandırma ile incelemekteyiz: (i) temel yapılandırmada çalışmaları önerildikleri şekilde yapılandırmakta, (ii) geliştirilmiş yapılandırmada QUAC-TRNG ile daha adil bir karşılaştırma sunabilmek için SHA-256 işlevini kullanarak DRAM işlem (hatalı *tRP* ve *tRCD* erişimi) sonuçlarını işlemekteyiz. Sonuçlarımız QUAC-TRNG'nin son teknoloji çalışmaların temel ve geliştirilmiş sürümlerinden sırayla 15,08 ve 1,41 kat daha hızlı rastgele sayı ürettiğini göstermektedir. Ayrıca QUAC-TRNG'nin artan DRAM veri yolu hızı ile daha iyi ölçeklendiğini, en iyi son teknoloji DRAM-tabanlı GRSÜ'ye göre 12 GT/s aktarım hızında 2,03 kat daha hızlı rastgele sayı ürettiğini göstermekteyiz. Son olarak, QUAC-TRNG'nin gerçek bir sisteme düşük işlemci devre alanı, küçük başarım ve bellek kapasitesi kayıpları ile nasıl uygulanabileceğini tartışmaktayız.

Bu çalışma ile literatüre sunduğumuz katkılar aşağıda listelenmektedir:

- Dikkatlice tasarlanmış DRAM komut dizilerinin ardışık dört DRAM satırını neredeyse aynı anda açabileceğini ilk defa gözlemlemekteyiz (QUAC). QUAC işlemlerinin bit hattı algılama yükselteçlerini yarı-kararlı bir duruma sokabileceğini göstermekte ve bundan faydalanarak gerçek rastgele sayı üretmekteyiz.
- Gerçek DRAM aygıtlarında uygulanabilen, QUAC işlemlerini taban alan yeni bir yüksek hızlı gerçek rastgele sayı üreteci olan QUAC-TRNG'yi geliştirmekteyiz. QUAC-TRNG yüksek hızda rastgele sayı üretmek için (i) DRAM algılama yükselteçlerinin sağladığı paralellikten ve (ii) SHA-256'nın sağladığı rastgelelik niteliksi artışından faydalanarak son teknoloji DRAM-tabanlı rastgele sayı üreteçlerinden çok daha hızlı rastgele sayı üretmektedir.
- QUAC-TRNG'nin yüksek nitelikte gerçek rastgele sayı ürettiğini, QUAC-

TRNG tarafından üretilen rastgele sayıların NIST istatistiksel test paketindeki rastgelelik testlerinin [36] tamamını geçtiğini gözlemleyerek göstermekteyiz.

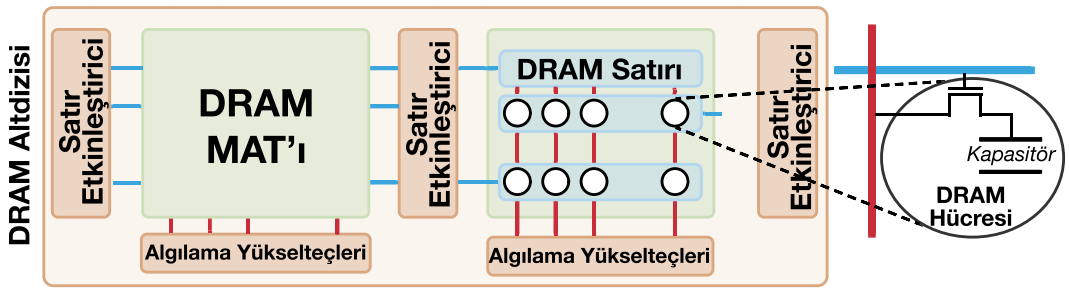
- QUAC-TRNG'nin son teknoloji DRAM-tabanlı GRSÜlerden [22, 24] daha hızlı olduğunu, son teknoloji GRSÜlerin temel ve geliştirilmiş sürümlerinden 15,08 ve 1,41 kat daha hızlı gerçek rastgele sayı ürettiğini gözlemleyerek göstermekteyiz.
- 17 DRAM modülündeki 136 DRAM aygıtını kullanarak QUAC-TRNG'nin ayrıntılı bir nitelendirmesini gerçekleştirmekteyiz. Nitelendirme çalışmamız (i) QUAC-TRNG'nin gerçek DRAM aygıtlarında uygulanabilir olduğunu ve (ii) QUAC işlemlerinin sağladığı rastgeleliğin zamanla sabit kaldığını göstermektedir.

2. TEMEL BİLGİLER

2.1 DRAM Yapısı ve Organizasyonu

DRAM-tabanlı ana bellek hiyerarşik bir şekilde düzenlenmiştir. Bir işlemci bir ya da daha fazla bellek kanalı ile birden çok DRAM modülüne bağlanır. Her bellek kanalının kendine has adres, komut ve veri hatları vardır. Bir DRAM modülü birden çok DRAM yongasını barındırır. Her DRAM yongası birbirinden bağımsız çalışan birden çok DRAM kümesinden (*-ing.* DRAM bank) oluşur. Bazı DRAM standartları birden çok DRAM kümesini DRAM küme grupları içinde barındırır [37, 38]. Tipik olarak DRAM modülleri ile işlemci arasındaki veri aktarımları bir önbellek satırı (512-bit) boyutunda gerçekleşir.

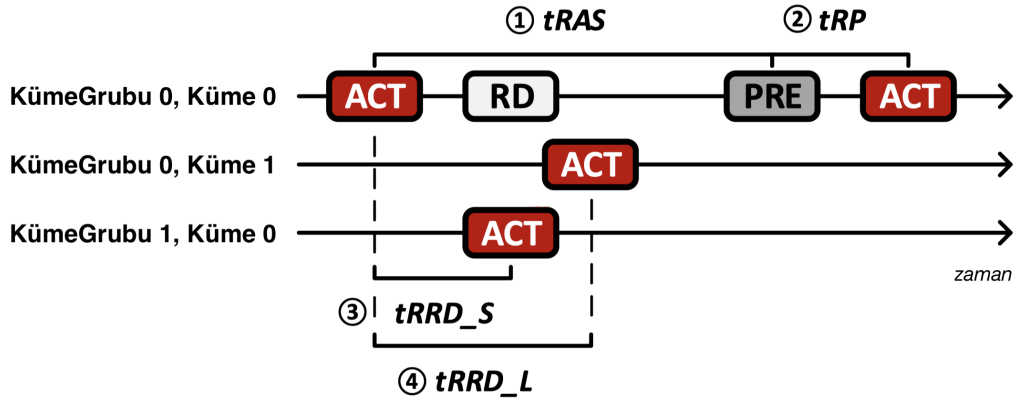
DRAM Küme Organizasyonu. Bir DRAM kümesi birden çok DRAM altdizisinden (*-ing.* DRAM subarray) oluşur [39–41]. Şekil 2.1'de gösterildiği gibi bir altdizi birden çok satır etkinleştirici (*-ing.* wordline driver) ve algılama yükselteci barındırır. Bir DRAM altdizisi birbirleri arasında satır etkinleştiricileri barındıran birden çok DRAM MAT'ına ayrılır. Bir DRAM MAT'ındaki DRAM hücreleri yatay eksende DRAM satırları ve dikey eksende bit hatları olmak üzere iki boyutlu bir yapıda dizilmiştir.



Şekil 2.1: DRAM altdizi, MAT, ve hücre organizasyonu

DRAM Hücrelerine Erişim. Bir DRAM hücresi veriyi V_{DD} ve toprak gerilimi arasında bir düzeyde, kapasitörünün içinde saklar. Her DRAM hücresi bir bit hattına erişim transistörü ile bağlanmıştır. Tüm satırlar kapalı konumdayken bit hatları $V_{DD}/2$ gerilim düzeyine önceden yüklenir. Bir DRAM hücresine erişmek için hücrenin bulunduğu satırı hedef alan bir ACT (etkinleştir) komutu gereklidir. Etkinleştirme işlemi önce satır hattını (*-ing.* wordline) sürer, böylece satır üzerindeki tüm erişim transistörleri açılır. Açılan transistörler bağlı oldukları DRAM hücrelerinin bit hatları ile yüklerini paylaşmalarına sebep olur. Bunun sonucunda bit hattı gerilimi V_{DD} ya da toprak gerilimine doğru yaklaşır. Bit hattı gerilimindeki değişiklik V_{th} eşik düzeyini geçtiğinde algılama yükselteçleri bit hattı gerilimini V_{DD} ya da 0 gerilimine yükseltir. Ancak bu noktadan sonra hücrelere erişim algılama yükselteçleri üzerinden okuma ve yazma komutları ile gerçekleştirilebilir. Sonrasında PRE (önyükle) komutları ile satırlar kapatılabilir ve bit hattı gerilimleri $V_{DD}/2$ olarak ayarlanır.

DRAM Zamanlama Parametreleri. Bir bellek denetleyicisi, DRAM aygıtına komut gönderirken JEDEC tarafından belirlenen DRAM standartlarındaki (örneğin, DDR4 [37]) zamanlama parametrelerine uymak zorundadır. DRAM komutlarının bir zaman çizelgesi Şekil 2.2'de gösterilmiştir. Ardışık ACT ve PRE komutları birbirlerinden *tRAS* (ACT $\rightarrow$ PRE zamanlama parametresi) kadar ayrılmalıdır. Çünkü bir DRAM satırındaki hücrelere yükün geri doldurulması için en azından *tRAS* kadar süre geçmelidir. Ardışık PRE ve ACT komutları arasında en az *tRP* kadar zaman geçmelidir. Bu süre, bit hattı gerilimini $V_{DD}/2$'ye sürmek için ve satır hatlarını kapatmak için gereklidir. Farklı DRAM küme gruplarını hedefleyen ardışık ACT komutları t_{RRD_S}, aynı DRAM küme gruplarını hedefleyen ardışık ACT komutları ise t_{RRD_L} kadar süre ile birbirlerinden ayrılmalıdırlar. Bu parametreler (t_{RRD_S} ve t_{RRD_L}) tipik olarak kısa sürelidir (DDR4-2666 standardında [37] 3,00 ve 4,90 ns). Bu sayede farklı DRAM kümelerindeki veya küme gruplarındaki satırları açmak için gereken gecikmeler birbiri ardına saklanabilir.



Şekil 2.2: Önemli DRAM komutlarının zaman çizelgesi

DRAM üreticileri bu zamanlama parametrelerini olması gerektiğinden yukarıda tutarak tüm DRAM hücrelerinin doğru çalışmasını sağlar ve üretim verimini artırır [42–46]. Çok sayıda çalışma DRAM davranışını standart olmayan zamanlama parametreleri (örneğin, standarda göre daha düşük) altında inceler ve bu zamanlama parametrelerinin ihlal edilmesi ile gerçek DRAM aygıtlarında erişim gecikmesinin hızlandırılabileceğini [42–51], fiziksel klonlanamayan fonksiyonların (*-ing.* physical unclonable function, PUF) gerçeklenebileceğini [24, 51, 52], rastgele sayı üretilebileceğini [22, 24, 51], kopyalama işleminin yapılabileceğini ve VE/VEYA mantık işlemlerinin yapılabileceğini göstermiştir [53].

2.2 Gerçek Rastgele Sayı Üreteçleri

Gerçek rastgele sayı üreteçleri (GRSÜler) rastgele davranış gösteren fiziksel süreçlerden (örneğin, Brown hareketi, termal gürültü) entropi elde ederek rastgele sayı üretirler [54]. Entropi kaynağı olarak kullanılan bu süreçler tipik olarak rastgele sayıların eğilimli (*-ing.* biased) olarak üretilmesine sebep olur (örneğin, rastgele sayı dizisindeki mantık-1 sayısı mantık-0 sayısından çok daha fazla olabilir). Dolayısıyla pratik GRSÜ tasarımları bu eğilimin giderilmesi için çeşitli sonradan işleme (*-ing.* post-processing) yöntemi kullanırlar (örneğin, özet fonksiyonları [55]). Sonradan işleme yöntemleri-

nin kullanılması GRSÜlerin hızlarını ve gecikmelerini sınırlayabilir, bu sınırlamanın üstesinden gelmek için daha fazla donanım kaynağı (örneğin, ara bellek yapıları) kullanılabilir.

3. MOTİVASYON

İstatistiksel açıdan güçlü (yüksek nitelikli) rastgele sayılar birçok teknoloji ve uygulama için önem arz etmektedir [1, 3–5, 9, 10, 13–20, 56–60]. Rastgele sayılar özellikle kriptografik haberleşme protokollerinde (örneğin, anahtar ve imza üretimi) ağ üzerindeki iki veya daha fazla aygıt arasında güvenli bir bağlantı kurmak için yaygın bir şekilde kullanılmaktadır. Bu protokoller, kullanıcıların değerli ve gizli verilerine erişmeyi amaçlayan kriptografik saldırılara [2, 18] karşı dayanıklılık için yüksek nitelikli gerçek rastgele sayılara ihtiyaç duymaktadır. Bazı yeni ortaya çıkan anahtar dağıtım protokolleri (örneğin, kuantum anahtar dağıtımı) daha yüksek düzeyde güvenlik garantilemekte, daha çok sayıda ve çeşitte kriptografik saldırıya karşı dayanıklılık sağlamaktadır. Bu protokoller Gb/s düzeyinde hıza sahip GRSÜlere ihtiyaç duymaktadır. GRSÜler kriptografik uygulamalar dışında bilimsel benzetimlerde [9, 58–60], yapay öğrenme [1, 3–5] ve oyun uygulamalarında [10] da kullanılmaktadır.

Yüksek Hızlı GRSÜler. Çok sayıda geçmiş çalışma özel donanım (örneğin, optik [61–64], halka osilatör [21, 65–67], düzensiz devreler [68, 69]) kullanan yüksek hızlı GRSÜ geliştirmektedir. Ne yazık ki bu GRSÜler ya (i) sistemlere tasarım zamanında yerleştirilmelidir, dolayısıyla var olan sistemlerde uygulanamazlar ya da (ii) masraflı oldukları için yaygın kullanımları mümkün değildir. Yüksek hızlı GRSÜlerin zayıf yanlarının üstesinden gelmek için ve yukarıda belirtilen uygulamaları çeşitli bilgisayar sistemlerinde destekleyebilmek için yüksek nitelikli GRSÜler var olan donanımlar kullanılarak geliştirilmelidir.

DRAM-tabanlı GRSÜler. DRAM yongaları günümüz bilgisayar sistemlerinde çok yaygın olduğundan, DRAM-tabanlı GRSÜler gerçek rastgele sayı üretimi için gelecek vadetmektedirler. DRAM-tabanlı GRSÜler, bilgisayar sistemlerine düşük maliyet ve göz ardı edilebilecek çaba ile yerleştirilebilirler [22]. Bu sayede hem günümüz hem de gelecek bilgisayar sistemlerinde gerçek rastgele sayı üretme yeteneği getirilebilir.

PIM ile Sinerji. Bellek içinde hesaplama (*-ing.* processing-in-memory, PIM) yöntemleri, hesaplama işlemlerini bellek yongalarının içinde yaparak sistemde gereksiz veri hareketinin önüne geçer, bu sayede sistem başarımının ve enerji tüketiminin iyileştirilmesini sağlar [70–78]. Geçmiş çalışmalar iş yüküne ve kullanılan bellek aygıtına bağlı, geniş yelpazeli bellek içinde hesaplama sistemleri önermektedir [70–72, 79–116]. Bellek içinde hesaplama sistemlerinde yüksek nitelikli gerçek rastgele sayılara ihtiyaç duyan yeni iş yüklerini (örneğin, güvenlik uygulamaları) yürütebilmek için DRAM-tabanlı GRSÜler kullanılabilir. Bu sayede (i) verimsiz yonga-dışı haberleşmeden (diğer GRSÜ kaynakları ile) kaçınılabilir ve (ii) bellek içinde hesaplama sistemlerinin güvenliği ve gizliliği iyileştirilebilir.

Geçmiş Çalışmaların Eksiklikleri. Geçmiş DRAM-tabanlı GRSÜler ya (i) temel olarak yavaş süreçlere (örneğin, DRAM tutma hataları [25, 31–33], DRAM başlangıç değerleri [34]) dayandıklarından yüksek gecikmelere sahiptirler, ya da (ii) belirli DRAM

satırlarının küçük kısımlarını entropi kaynağı olarak kullandıklarından (örneğin, *tRCD* hatası tabanlı üreteçler [22]) veya çok sayıda algılama yükseltecini yarı kararlı duruma getiremediklerinden (örneğin, *tRP* hatası tabanlı üreteçler [24]) düşük hızda gerçek rastgele sayı üretirler.

DRAM başlangıç değerlerini kullanan GRSÜler [34] rastgele sayı üretmek için DRAM aygıtını yeniden başlatır. Bu mekanizma yüksek hızlı GRSÜler için kullanılamaz çünkü hem (i) bu mekanizmanın rastgele sayı üretme gecikmesi yüksektir hem de (ii) bu mekanizma kesintisiz bir akış şeklinde rastgele sayı üretilmesini mümkün kılmaz. DRAM tutma hatalarını kullanan GRSÜler [23, 25] yeterince entropi elde etmek için çok sayıda DRAM tutma hatasını geniş bir zaman aralığında biriktirir. DRAM hücreleri tutma hatalarından ötürü çok uzun sürede veri kaybeder, bu yüzden bu GRSÜler rastgele sayı üretmek için dakikalarca beklemek zorundadır [117–121]. DRAM satır açma gecikmesi (*tRCD*) tabanlı GRSÜlerin hızları belirli DRAM satırlarının küçük kısımlarından (bir DRAM önbellek satırı, tipik olarak 512 bit) elde edilebilecek entropi ile sınırlıdır. Örneğin D-RaNGe [22] bir satırdaki 65.536 hücrenin en fazla 4 tanesini rastgele sayı üretimi için kullanmaktadır. Ön yükleme gecikmesi (*tRP*) tabanlı GRSÜler [24] bir DRAM satırındaki çok sayıda hücrede paralel bir şekilde hata oluşturabilir. Ancak hatalı hücreler arasında rastgele hatalara sahip hücrelerin oranı çok düşüktür. Bölüm 7.4'te geçmiş DRAM-tabanlı GRSÜlerin ayrıntılı bir incelemesi sunulmaktadır.

Geçmiş çalışmalar üzerinde yaptığımız incelemenin sonucu olarak yüksek hızlı DRAM-tabanlı GRSÜlerin (i) temel olarak hızlı gerçekleşen DRAM hata mekanizmalarını (örneğin, zamanlama hataları) kullanması gerektiğini, (ii) DRAM satırlarındaki tüm hücreleri (mümkün olduğu kadar) entropi kaynağı olarak kullanması gerektiğini ve (iii) çok sayıda algılama yükseltecini yarı kararlı duruma sokması gerektiğini görmekteyiz.

Bu çalışmadaki amacımız var olan DRAM aygıtlarını kullanan, geçmiş DRAM-tabanlı GRSÜlerden daha hızlı ve daha düşük gecikme ile rastgele sayı üreten yeni bir GRSÜ geliştirmektir.

4. DÖRTLÜ AKTİVASYON (QUAC)

Gerçek DRAM aygıtlarında dörtlü etkinleştirme (QUAC) adını verdiğimiz yeni bir davranış gözlemlemekteyiz. Gözlemimize göre bellek denetleyicisinden DRAM aygıtına ardışık gönderilen üç standart DDR4 komutu (ACT $\rightarrow$ PRE $\rightarrow$ ACT), zamanlama parametreleri düşük (örneğin, 2,5 ns) belirlendiğinde, DRAM aygıtında aynı altdizideki bitişik dört DRAM satırının kısa aralıklarla açılmasına sebep olmaktadır. QUAC işlemlerinin başarılı olması için öncelikle ardışık gönderilen ACT komutlarının hedef adreslerinin yalnızca en anlamsız iki biti farklı olmalıdır (örneğin, 0 - 1 - 2 - 3 satırları). Bu şekilde QUAC ile aktive edilebilen her dört DRAM satırı kümesini DRAM bölütü olarak adlandırmaktayız. Ayrıca QUAC işlemlerinin başarılı olması için ACT komutlarının hedef adreslerinin en anlamsız iki bitinin birbirine zıt olması gerektiğini gözlemlemekteyiz. Bir başka deyişle iki ACT komutu DRAM bölütü içindeki 0. ve 3. (ikilik tabanda 00 ve 11) ya da 1. ve 2. (ikilik tabanda 01, 10) satırları hedefleyebilir.

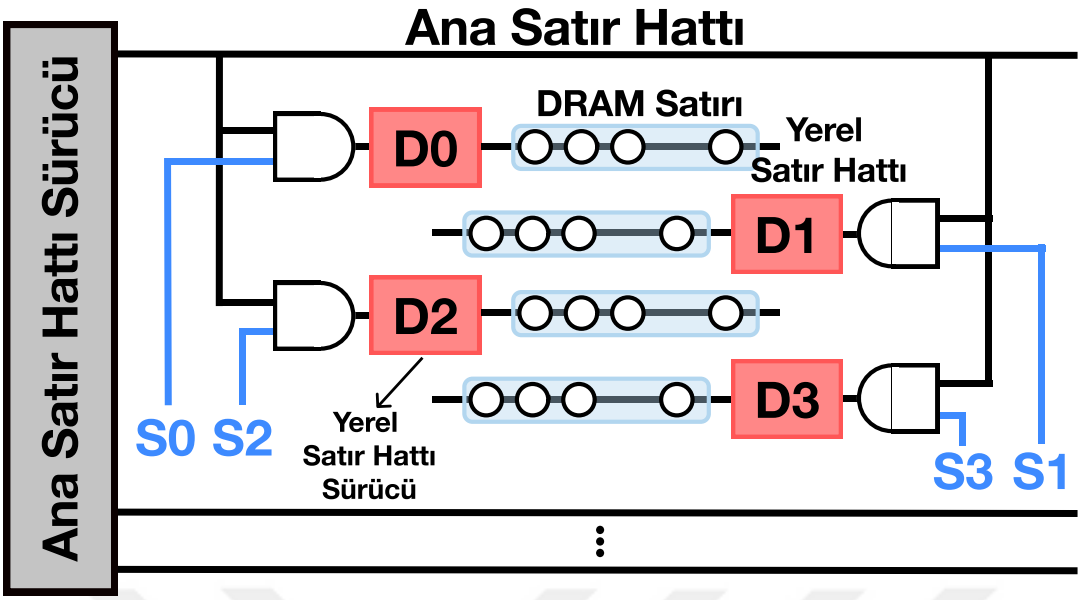
QUAC'ın çalışmasına sebep olan mekanizmayı açıklayabilmek için günümüz DRAM yongalarının mimarisini incelemekteyiz. İnceleme sonucu kurduğumuz hipotez, DRAM satır hatlarının hiyerarşik tasarımının QUAC işlemlerinin DRAM bölütlerindeki dört DRAM satırını açmasına izin verdiğidir. Bölüm 4.1'de bu hipotezimizi destekleyen ve QUAC işlemlerindeki ACT komutlarının hedef adreslerinin neden birbirine zıt bitlerden oluştuğunu açıklayan bir satır adresi kod çözücü (row decoder) tasarımını anlatmaktayız.

4.1 Hiyerarşik Satır Hatları

Yüksek kapasite ve başarım gereksinimleri DRAM üreticilerini yüksek kapasiteli ve düşük gecikmeli DRAM dizi mimarileri tasarlamaya itmektedir [122]. DRAM satır hatlarının hiyerarşik tasarlanması ve organize edilmesi, bu amaçla sıklıkla kullanılan bir tasarım örüntüsüdür [45, 123–125]. Hiyerarşik satır hattı tasarımına sahip bir DRAM MAT Şekil 4.1'de gösterilmektedir.

Hiyerarşik satır hattı tasarımlarında DRAM satır adresleri iki parçaya ayrılır. Satır adresi bitlerinin en anlamlı bitleri bir ana satır hattını (*-ing.* master wordline) açmaya yarar. Her ana satır hattı dört yerel satır hattı sürücüsüne (*-ing.* local wordline driver) bağlıdır (Şekil 4.1'de D0, D1, D2 ve D3). Bu sürücüler MAT içinde ardışık yerleştirilen DRAM satırlarını açmaya yarar. Satır adresi bitlerinin en anlamsız iki biti yerel satır hattı sürücülerinin seçim sinyallerinin (Şekil 4.1'de S0'dan S3'e) değerlerini belirleyerek yerel satır hattı sürücülerini açar ve sonuç olarak bir DRAM satırı açılmış olur.

Açık bir ana satır hattının dört yerel satır hattı sürücüsünü (bir DRAM bölütü) açma potansiyeli vardır. Hipotezimize göre QUAC komutları (ACT-PRE-ACT) S0, S1, S2

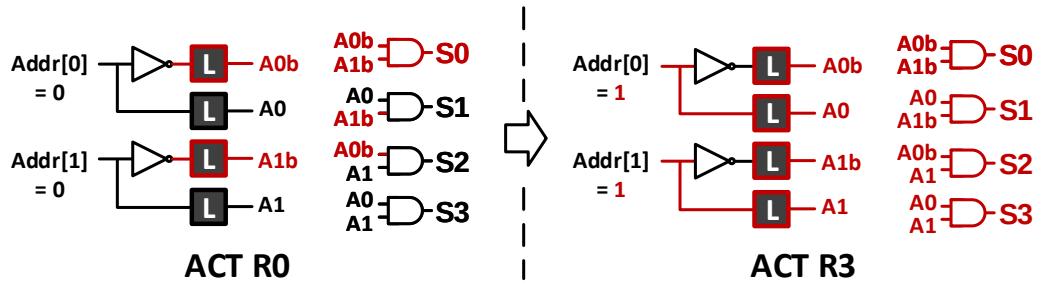


Şekil 4.1: Hiyerarşik satır hattı tasarımına sahip DRAM MAT

ve S3'ü yaklaşık olarak aynı anda açtığından dört bitişik DRAM satırının aynı anda açılmasına sebep olmaktadır.

4.2 Hipoteze Dayalı Satır Adresi Kod Çözücü Devresi

Hipoteze dayalı devremiz DRAM yongası zamanlama parametreleri ihlal edilmiş bir ACT-PRE-ACT DRAM komut serisi aldığında dört ardışık DRAM satırını açabilmektedir. Şekil 4.2, satır adres bitlerinin en anlamsız iki bitini giriş olarak alan devremizin tasarımını göstermektedir.



Şekil 4.2: QUAC'ı destekleyen, hipoteze dayalı satır adresi kod çözücü devresi. Kırmızı gösterilen sinyaller açık (mantık-1), siyah gösterilen sinyaller kapalı (mantık-0) anlamına gelmektedir.

İlk satırı (Satır 0, R0, Addr[1:0] = "00") hedefleyen ACT komutu, (Şekil 4.2'de solda) A0b ve A1b sinyallerini süren devrenin (latch, L) değerini belirler. Bu iki sinyal mantıksal VE işlemine tabi tutulup S0 sinyalini oluştururlar. S0 sinyali R0'ı açan yerel satır hattı sürücüsünü açar. Sıradaki PRE komutu *tRAS* zamanlama parametresi ihlal edildiğinden R0'ı kapatamaz ve A0b ve A1b'yi süren devreyi de başlangıç durumuna getiremez. Dördüncü satırı (Satır 3, R3, Addr[1:0] = "11") hedefleyen ACT komutu A0

ve A1 sinyallerini süren devrenin değerini belirler. İkinci ACT komutu yürütüldükten sonra PRE komutu A0b, A1b, A0 ve A1 değerlerini süren devreleri başlangıç durumuna getiremediğinden tüm denetim sinyalleri (A0, A0b, A1 ve A1b) açık hale gelir. Bu sinyaller S0, S1, S2 ve S3 sinyallerine aynı anda mantık-1 sürülmesine, dolayısıyla R0, R1, R2 ve R3 satırlarının aynı anda açılmasına sebep olur.

Gerçek DRAM yongaları kullanarak gerçekleştirdiğimiz deneyler ile QUAC'ın dört DRAM satırını açtığını gözlemlemekteyiz. Deneyde ilk olarak bir DRAM bölütünü önceden belirlenmiş bir veri örüntüsü ile dolduruyoruz. Sonra bu DRAM bölütünde aynı anda dört satırı açmak için QUAC işlemi gerçekleştiriyoruz. Daha sonra dört satır açıkken algılama yükselteçlerine yeni bir veri örüntüsü yazıyoruz. Son olarak DRAM kümesine PRE komutu gönderiyoruz ve zamanlama parametrelerine uyarak DRAM bölütündeki tüm satırları okuyoruz. Bu işlemlerin sonunda bölütteki dört satırın da yeni veri örüntüsü ile güncellenmiş olduğunu görmekteyiz. Bir DRAM üreticisinin 136 DRAM yongasında geçerli QUAC işlemleri gözlemlemekteyiz.

4.3 Gelecek QUAC Arayüzleri

Günümüz DRAM standart arayüzleri (örneğin, DDR4) QUAC işlemlerini desteklememektedir[2]. Ancak gelecekte üretilen DRAM yongaları QUAC'ın temel aldığı davranıştan yararlanarak düşük maliyetli ve yüksek hızda gerçek rastgele sayı üretimini gerçekleştirebilirler.

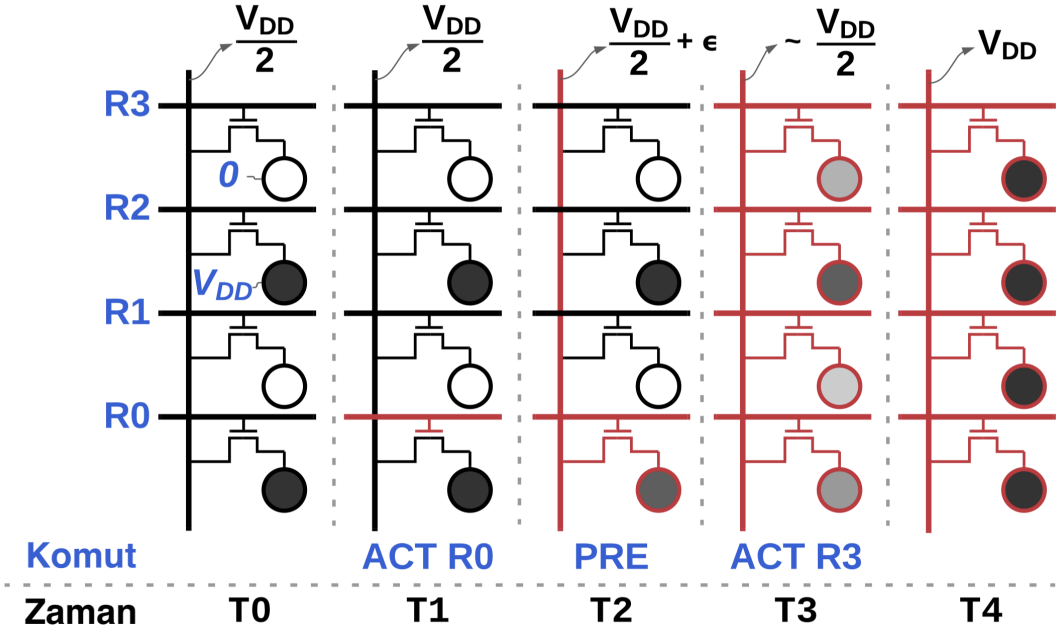
[2]QUAC işlemleri için DRAM standartlarında bir komut bulunmamaktadır, ancak çalışmamızda gerçekleştirdiğimiz gibi zamanlama parametreleri güvenli eşiklerin altına indirilerek QUAC işlemleri gerçek DRAM cihazlarında çalıştırılabilir.

5. QUAC-TRNG

QUAC-TRNG art arda QUAC işlemleri gerçekleştirerek yüksek hızda gerçek rastgele sayı üretmektedir.

5.1 QUAC ile Rastgele Sayı Üretimi

Bir DRAM bölütünde R0 ve R2 satırlarındaki hücreler başlangıç durumunda yüklü (V_{DD}) olduğunda ve R1 ve R3 satırlarındaki hücreler başlangıç durumunda yüksüz (0) olduğunda, bir QUAC işleminin nasıl rastgele sayı oluşturduğu Şekil 5.1'de gösterilmektedir.



Şekil 5.1: QUAC işlemi sırasında bir DRAM bölütü içindeki bir bit hattındaki durum değişikliklerinin zaman diyagramı. Çizikli dikey çizgiler bir durum değişikliğini belirtmektedir.

T0 anında bit hattı $V_{DD}/2$'ye önyüklenmiştir. T1'de R0'ın satır hattını hızlı bir şekilde ACT komutu gönderek açmaktayız. T2'de ACT komutunu yarıda kesen bir PRE komutu göndermekteyiz. Bu sırada R0 satırındaki DRAM hücresi yükünü bağlı olduğu bit hattı ile paylaşmakta, dolayısı ile yükünün bir kısmını kaybetmektedir ($< V_{DD}$). PRE komutu aktif olan satırı kapatıp bit hatlarını önyükleyemeden T3'te R3'ü hedef alan bir ACT komutu daha göndermekteyiz. Bu son ACT komutu PRE komutunu yarıda kesmekte ve R1, R2 ve R3 satırlarını aynı anda açmaktadır. R0 ilk ACT komutu tarafından açılmış olduğundan, ikinci ACT komutu sonunda dört satır birden

açılmış olmaktadır. QUAC işlemi dört satırı birden açtığı için bu satırlar üzerindeki dört ayrı hücre bit hattı gerilimine katkı sağlamaktadır. QUAC işleminden sonra (T4 anında) bit hattı gerilimi algılama yükselteçleri güvenilir eşiğinin altında kalmaktadır. Böylece algılama yükselteçleri bu gerilim değerini rastgele bir değer olarak örnekler. Şekil 5.1'deki örnekte bit hattı gerilimi rastgele bir şekilde V_{DD} olarak örneklenmiştir.

QUAC'ın nasıl gerçek rastgele sayı ürettiğini açıklamak için bir hipotez kurmaktayız. Hipotezimize göre QUAC rastgele sayıları algılama yükselteçlerinde oluşturmaktadır. QUAC işlemleri her algılama yükseltecini güvenilir eşiklerinin altında (bit hattında neredeyse hiç gerilim değişikliği olmadığı durum) bir gerilim değişikliğini algılamaya zorlamaktadır. Bu koşullar altında algılama yükselteçleri güvenilir bir şekilde bit hattı gerilimini yükseltemez ve termal gürültü [126] sebebiyle kararsız bir şekilde bit hattı gerilimini ya mantık-0 ya da mantık-1 değerine yükseltir.[3] Bit hattı gerilimindeki değişikliğin algılama yükselteçlerinin güvenilir eşiklerini aşmamasını sağlamak için aynı anda açılan dört satırdaki hücreleri birbirine zıt veriler ile doldurmaktayız. Bu sayede bu hücreler bit hattı ile yük paylaşımlarını gerçekleştirdiğinde bit hattı gerilimindeki toplam değişiklik 0'a yakın olmakta, bit hattı gerilimi ise $V_{DD}/2$'ye yakın bir değerde kalmaktadır.

5.2 Mekanizma

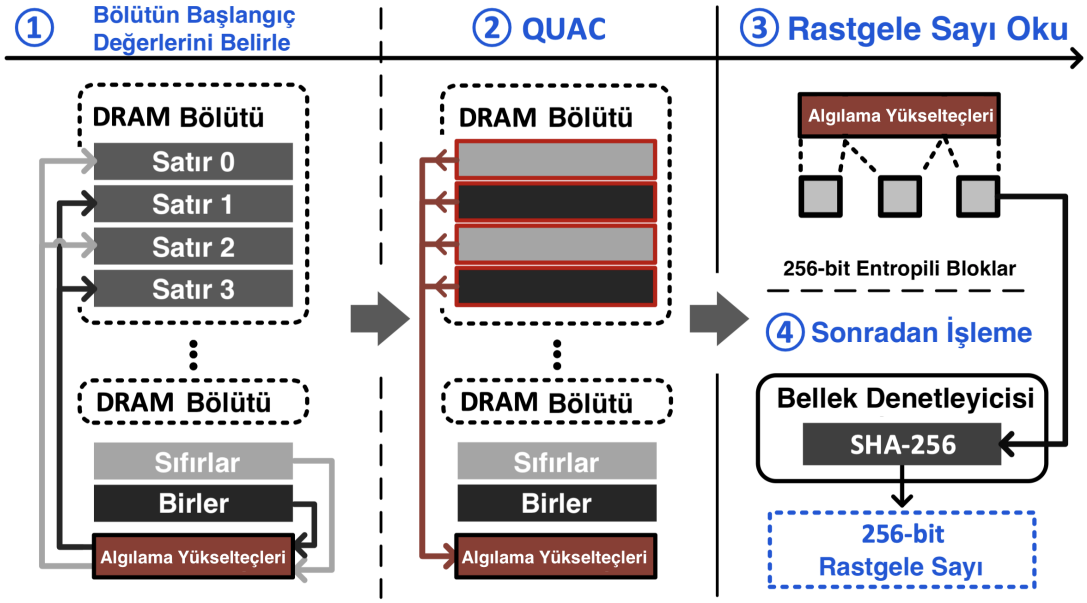
QUAC-TRNG, QUAC işlemleri sonucunda algılama yükselteçlerinde oluşan rastgele sayıları entropi kaynağı olarak kullanarak rastgele sayı üretmektedir. QUAC-TRNG ilk olarak yüksek entropili[4] bir DRAM bölütünde QUAC işlemi çalıştırır. QUAC-TRNG sonrasında SHA-256 kriptografik özet fonksiyonunu [127] kullanarak DRAM bölütündeki algılama yükselteçlerinden okuduğu rastgele sayıları işlemekte ve yüksek nitelikli gerçek rastgele sayılar üretmektedir.

Şekil 5.2'de bir DRAM altdizisinin QUAC-TRNG işlemleri için kullanıldığı zamandaki mantıksal organizasyonu ve QUAC-TRNG'nin 256 bitlik bir rastgele sayı üretmek için yürüttüğü üç aşamalı süreç gösterilmektedir. DRAM altdizisinde altı DRAM satırı, bilgisayar sisteminin herhangi bir başka parçasının aynı anda QUAC işlemlerinin gerçekleştirildiği bölgeye erişmediğine emin olmak için ayrılmaktadır. Bu altı satırın ikisi yalnızca mantık-0 ve yalnızca mantık-1 değerlerini tutmakta ve DRAM bölütlerini hızlıca başlangıç değerleri ile doldurmak için kullanılmaktadır.

QUAC-TRNG 256 bitlik rastgele sayı üretmek için ilk olarak yüksek entropili bir DRAM bölütünü seçmekte ve DRAM-içinde-kopyalama [53, 87] işlemleri kullanarak başlangıç değerleri ile doldurmaktadır①. İkinci olarak DRAM bölütünde bir QUAC işlemi gerçekleştirmekte ve algılama yükselteçlerinde rastgele değerler oluşturmaktadır②. Üçüncü olarak bellek denetleyicisi üzerinden toplamda 256 bitlik Shannon entropisine (Bölüm 6.1.1) sahip bir veri bloğunu okumaktadır③. Son olarak

[3] Tarif ettiğimiz davranışı DRAM bölütlerindeki tüm bit hatlarında gözlemlememekteyiz. Bu davranış yonga üretim sürecinden kaynaklanan değişiklikler (*-ing.* process variation) ile açıklanabilir. Yonga üretim sürecindeki değişiklikler bir DRAM aygıtındaki bit hattı kapasitanslarının, algılama yükselteçleri güvenilir eşiklerinin ve DRAM hücrelerinin kapasitanslarının değişiklik göstermesine sebep olmaktadır.

[4] Yüksek entropili DRAM bölütlerindeki algılama yükselteçlerinde QUAC işlemleri çok sayıda rastgele değer (> 1000 bit) üretir. Bu bölütler Bölüm 6.1.2'de anlatıldığı gibi, tek seferlik bir nitelendirme işlemi sonucunda bulunmaktadır.



Şekil 5.2: QUAC-TRNG mekanizması

da okuduğu bu bloğu SHA-256 kriptografik özet fonksiyonunu kullanarak işlemekte ve 256-bit genişliğinde bir rastgele sayı üretmektedir.

6. GERÇEK DRAM YONGA NİTELENDİRMESİ

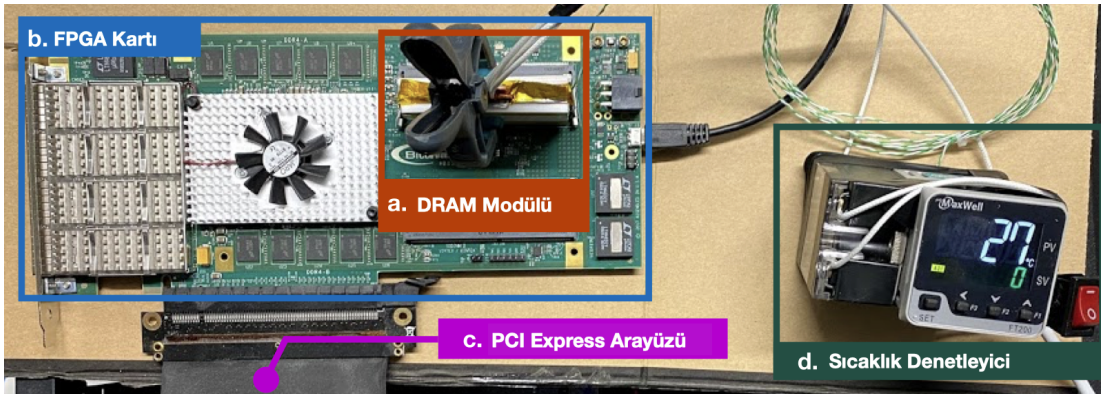
6.1 QUAC işlemlerindeki Rastgelelik

QUAC işlemlerinin entropi niteliklerini farklı veri örüntüleri ve DRAM bölütlerini kullanarak deneysel olarak incelemekteyiz.

6.1.1 Deneysel metodoloji

QUAC işlemleri sonucu oluşan rastgele değerlerin entropilerini nitelendirmek için 17 DRAM modülündeki 136 DRAM yongasında deneyler yürütmekteyiz.

Deney Altyapısı. Deneylerimizde DRAM zamanlama parametreleri üzerinde kesin denetleme kabiliyetine sahip, geçmiş çalışmalarda da kullanılan [128, 129] SoftMC DRAM test platformunun [130] değiştirilmiş bir versiyonunu kullanmaktayız. Bir ana bilgisayara PCIe arayüzü ile bağlı bir FPGA kartına (Şekil 6.1-b) yerleştirilmiş DDR4 modüllerinde (Şekil 6.1-a) deneylerimizi gerçekleştirmekteyiz. DRAM modüllerinin iki tarafında bulunan DRAM yongalarının sıcaklığını da denetlemekteyiz. Bunun için modülleri FPGA kartına dikey olarak takmakta ve bir kauçuk ısıtıcı ve mandal kullanmaktayız. Isıtıcaları denetlemek için bir sıcaklık denetleyicisi (Şekil 6.1-d) kullanmaktayız. Bu denetleyici $\pm 0.1\,^{\circ}\mathrm{C}$ hataya sahip olacak şekilde yongaların sıcaklığını istediğimiz sıcaklıkta tutabilmektedir (aksi belirtilmediği takdirde $50\,^{\circ}\mathrm{C}$).



Şekil 6.1: DDR4 SoftMC deney düzeneği.

Algoritma 1, QUAC kullanarak gerçek rastgele sayı elde etmek için kullandığımız test prosedürünü açıklamaktadır. Bu algoritma üç adımda gerçek rastgele sayı elde etmektedir: adım i) (*2. satır*) DRAM bölütünü başlangıç değerleri ile doldurur, adım ii) (*3-7. satırlar*) DRAM bölütünde bir QUAC işlemi gerçekleştirir, adım iii) (*9-10. satırlar*) algılama yükselteçlerindeki rastgele sayıları okur. DRAM bölütündeki tüm satırları aynı anda açmak için bölütteki birinci ve dördüncü satırları (Row_0 ve Row_3) iki ihlal edilmiş zamanlama parametresi (*tRAS* ve *tRP*) ile açmaktayız. İlk olarak PRE komutunu (*5. satır*) DRAM hücrelerine yükün geri doldurulması için gereken süreden (*tRAS*) daha kısa bir süre içinde göndermekteyiz. Sonra ikinci ACT komutunu (*7. satır*) bit hatları

$V_{dd}/2$ gerilim değerine sahip olamadan (tRP) önce göndermekteyiz. Rastgele sayıları algılama yükselteçlerinden okurken tüm zamanlama parametrelerine uymaktayız.

Algorithm 1: QUAC rastgelelik testi

```
1  DRAM_QUAC_rastgelelik_testi(veri_oruntusu, DRAM_bolutu, DRAM_kumesi):
2      DRAM_bolutu'ndeki tüm satırlara veri_oruntusu yaz
3      ac(DRAM_bolutu : Satir_0)
4      bekle(2,5ns) // tRAS ihlali
5      onyukle(DRAM_kumesi)
6      bekle(2,5ns) // tRP ihlali
7      ac(DRAM_bolutu : Satir_3)
8      bekle(tRCD)
9      foreach AY in DRAM_bolutu:   // algılama yükselteçlerini oku
10         AY'deki rastgele değeri kaydet
```

Shannon Entropy. Shannon entropisi [131] bir sinyalin sahip olduğu bilgiyi ölçmek için kullanılmaktadır. Çalışmamızda Shannon entropisini QUAC işlemleri sonucu DRAM algılama yükselteçlerinde oluşan rastgeleliği ölçmek için kullanmaktayız. Bir algılama yükseltecinin Shannon entropisini Eşitlik 6.1'deki gibi hesaplamaktayız. Bu eşitlikte $p(x_1)$ algılama yükseltecinden mantık-0 değeri, $p(x_2)$ ise algılama yükseltecinden mantık-1 değeri okuma olasılığını belirtmektedir. Bir bit dizisinin toplam Shannon entropisi (kısaca entropi) o dizideki rastgele bit sayısıymış gibi düşünülebilir.

$$H(x) = -\sum_{i=1}^{2} p(x_i)\log_2 p(x_i) \tag{6.1}$$

6.1.2 QUAC'taki entropiyi ölçmek için kullanılan metodoloji

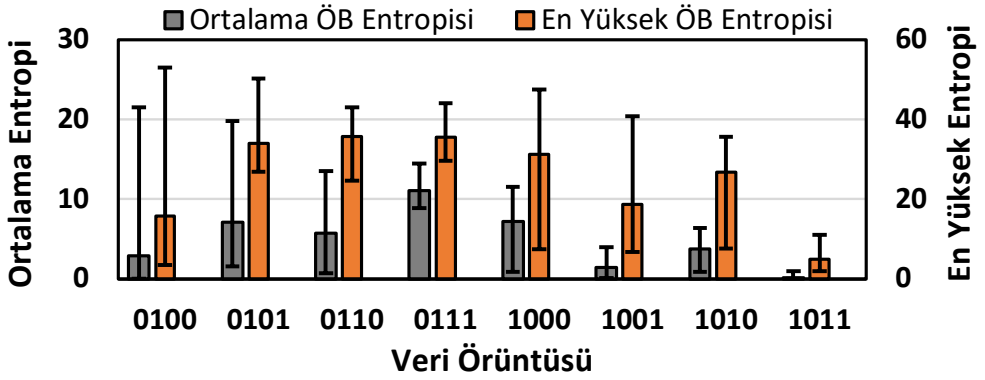
QUAC tarafından her algılama yükseltecinde oluşturulan rastgele bit dizilerininin entropisini ölçmekteyiz. Bunun için Algoritma 1'de gösterildiği gibi ardışık olarak QUACişlemleri yürütmekte ve her algılama yükseltecinden elde ettiğimiz 1000-bit boyutunda bit dizilerinin entropilerini Eşitlik 6.1'e göre hesaplamaktayız. Bu incelemeyi toplam 8K DRAM bölütünde (32K DRAM satırı) 16 farklı veri örüntüsü kullanarak gerçekleştirmekteyiz. Bir DRAM bölütündeki bir bit hattının entropisini, bağlı olduğu algılama yükseltecinden elde ettiğimiz bit dizileri ile hesaplamaktayız.

6.1.3 Veri örüntüsü bağımlılığı

DRAM bölütlerinin başlangıç değerlerini belirlemek için kullanılan veri örüntülerinin QUAC işlemlerinin sonuçlarına etkisini incelemekteyiz. Her DRAM önbellek bloğunun (512 bit hattı) entropisini, önbellek bloğundaki tüm bit hatlarının entropisini toplayarak hesaplamaktayız. Buna bağlı olarak iki metrik tanımlamaktayız: *(i) ortalama önbellek bloğu entropisi* ve *(ii) maksimum önbellek bloğu entropisi*[5]. Ortalama önbellek bloğu entropisini bir DRAM modülündeki tüm önbellek bloklarının ortalama entropisi olarak hesaplamaktayız. Maksimum önbellek bloğu entropisini ise bir DRAM modülündeki en yüksek önbellek bloğunun entropisi olarak hesaplamaktayız. Şekil 6.2 bu iki metriğin test ettiğimiz 17 modüldeki değerlerinin ortalamasını göstermektedir. Hata çubukları, metriklerin test edilen modüller arasında aldığı en düşük ve en yüksek

[5]Önbellek bloğu entropisi, her blok toplamda 512 bit hattına sahip olduğundan teorik olarak 512'ye kadar çıkabilir.

değerleri göstermektedir. Daha yüksek entropi algılama yükselteçlerinde daha rastgele davranışa işaret etmektedir. Şekil 6.2'de algılama yükselteçlerinde yeterince entropi oluşturmayan veri örüntüleri gösterilmemektedir.

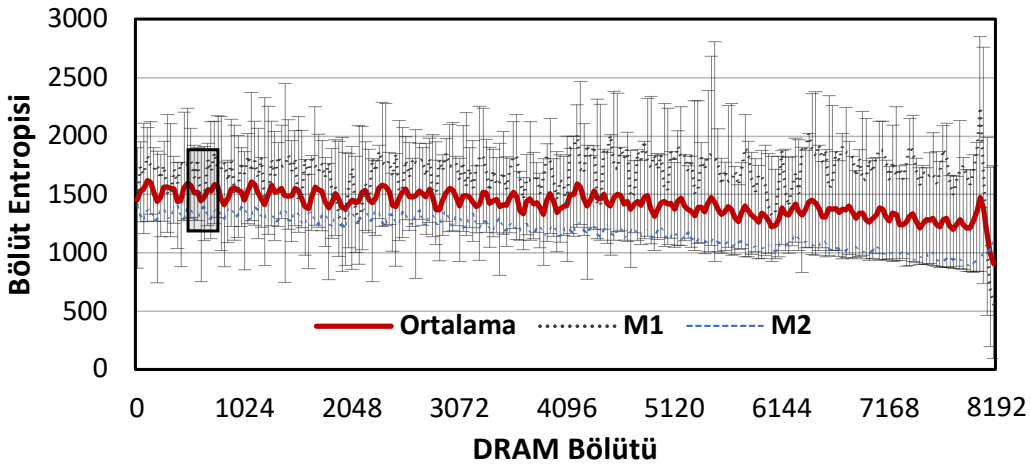


Şekil 6.2: Ortalama (gri çubuklar, sol Y-ekseni) ve maksimum (turuncu çubuklar, sağ Y-ekseni) DRAM önbellek bloğu entropileri.

Şekil 6.2 üzerinde üç gözlem yapmaktayız. İlk olarak, ortalama entropi veri örüntüsüne göre değişmektedir. "0111" veri örüntüsü için ortalama entropi en yüksek değeri (11,07 bit) almaktayken "1011" için ise en düşük değeri (0,17 bit) almaktadır. İkinci olarak, "0111" ve "1000" veri örüntülerinin ortalamada en yüksek entropiye yol açtığını gözlemlemekteyiz. Bu, QUAC tarafından açılan ilk DRAM satırının ($Satir_0$) başlangıç değeri ile sonradan açılan üç DRAM satırının başlangıç verisinin birbirine zıt olduğu durumda rastgeleliğin arttığını göstermektedir. Çünkü açılan ilk DRAM satırındaki hücrelerin bit hatları ile yük paylaşmak için diğer üç satırdaki hücrelere göre daha fazla vakti vardır. Hipotezimize göre bit hattı gerilimleri, bu durumda daha çok sayıda algılama yükseltecinde yarı kararlı durum oluşmasına sebep olmaktadır. Üçüncü olarak, önbellek bloğu entropisinin "0100" veri örüntüsü ile 53,0 bite kadar çıkabildiğini gözlemlemekteyiz. Bu gözlem |yonga tasarımındaki farklılıklar [45] ve üretim esnasında gerçekleşen, üretim sürecinden kaynaklanan değişikliklerin bir kombinasyonu ile açıklanabilir. Örneğin, DRAM hücrelerinin kapasitansları DRAM bölütleri arasında farklılık gösterebilir ve bu sebeple bazı DRAM bölütleri "0100" gibi veri örüntülerini tercih edebilir. Bir diğer deyişle, bu tür bölütlerde QUAC işlemi gerçekleştirmek (bölütün başlangıç değerleri "0100" örüntüsü ile belirlendiğinde) bit hatlarındaki gerilimi algılama yükselteçlerinin güvenilir eşiğinin altında tutuluyor olabilir.

6.1.4 Entropinin DRAM dizilerinde dağılımı

Entropinin DRAM dizilerinde dağılımını, farklı DRAM bölütlerindeki değerlerine bakarak incelemekteyiz. Bu incelemede bir bölütün entropisini, o bölütteki tüm bit hatlarının entropilerinin toplamı olarak hesaplamaktayız. Şekil 6.3, her 136 DRAM yongasındaki bir DRAM kümesi içindeki DRAM bölütlerinin entropilerinin, bölütlerin başlangıç değerleri en yüksek ortalama entropiye yol açan veri örüntüsü ("0111") ile belirlendiğinde nasıl değiştiğini göstermektedir. Bu şekilde üç eğri gösterilmektedir. Kırmızı eğri test edilen yongalar arasındaki ortalama entropiyi, hata çubukları ise DRAM bölütleri arasında gözlemlenen en yüksek ve en düşük entropiyi göstermektedir. Siyah (noktalı) ve mavi (çizgili) eğriler ise yongalar arasında gözlemlenen iki ana entropi değişiklik trendini göstermektedir.



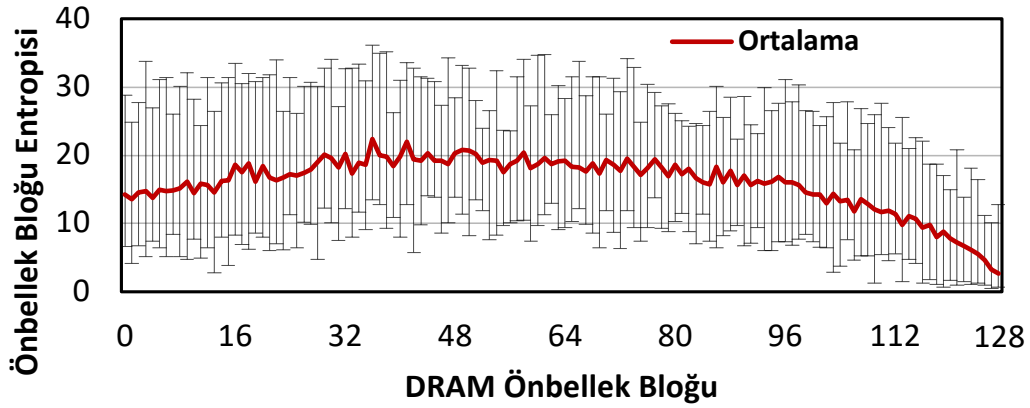
Şekil 6.3: 17 modüldeki (136 yonga) ortalama bölüt entropisi. X-ekseni DRAM bölüt numarasını, Y-ekseni ise bölüt entropisini göstermektedir. İki modüle özel bölüt entropilerini siyah (M1 modülü) ve mavi (M2 modülü) eğrileri kullanarak göstermekteyiz.

Şekil 6.3'den üç gözlem yapmaktayız. İlk olarak, DRAM bölüt entropisinin modülden modüle değişiklik gösterdiğini gözlemlemekteyiz. Örneğin, 640. bölüt (şekilde siyah kutunun ortası) M1 modülünde etrafındaki bölütlere göre önemli ölçüde daha düşük entropiye sahipken (yerel minimum entropi) M2 modülünde etrafındaki bölütlere göre daha yüksek entropiye sahip (yerel maksimum entropi) olmaktadır. Bu iki modülün (M1 ve M2) devre tasarımlarının aynı olduğu varsayılırsa, entropi davranışındaki bu değişiklik üretim sürecinden kaynaklanan sistematik farklılıklar [132] ve/veya üretim sonrası DRAM satır tamiri[6] yöntemi ile açıklanabilir [45, 88, 133–145]. İkinci olarak, DRAM bölüt entropisinin dağılımının bir dalga şeklinde olduğunu gözlemlemekteyiz. DRAM bölüt entropisi, bölüt numarası (bir diğer deyişle, DRAM satır adresi) arttıkça artmakta ve azalmaktadır. Bu dağılım örüntüsü üretim sürecinden kaynaklanan sistematik farklılıklar ve DRAM dizisinin yerel mimari özellikleri ile açıklanabilir. Örneğin, bir DRAM bölütünün entropisi, o bölütün algılama yükselteçlerinden uzaklığı ile bağlantılı olabilir. Üçüncü olarak, test edilen DRAM modüllerinin çoğunluğunda bölüt entropisi, bölüt numarası 8000'e giderken önemli ölçüde artmaktadır ve sonrasında DRAM kümesinin sonuna doğru azalmaktadır. Bu davranış üretim sürecinden kaynaklanan sistematik farklılıklar veya DRAM kümesinin mikromimari özellikleri ile açıklanabilir. Örneğin, DRAM kümesinin sonundaki DRAM alt dizileri, diğer alt dizilerden daha farklı boyuta sahip olabilir. Bu da bazı bölütlerin algılama yükselteçlerinden daha uzağa yerleştirilmesine sebep olabilir.

Bir önbellek bloğunun entropisini, o bloktaki tüm bit hatlarının entropilerinin toplamı olarak hesaplamaktayız. Bu incelemede de en yüksek ortalama entropiye yol açan veri örüntüsü ile DRAM bölütlerinin başlangıç değerlerini belirlemekte ve en yüksek entropiye sahip DRAM bölütündeki tüm önbellek bloklarının entropilerini hesaplamaktayız. Şekil 6.4 en yüksek entropiye sahip DRAM segmentindeki tüm önbellek bloğu entropilerini göstermektedir. Şekle göre önbellek bloğu entropisi DRAM bölütünün ortalarına doğru zirve yapmakta ve DRAM bölütünün kenarlarına yaklaştıkça azalmaktadır. Bu da yüksek numaralı önbellek bloklarındaki bit hatlarının, düşük ve orta

[6]DRAM aygıtlarındaki hatalı satırlar, üretim veriminin artırılması için üretim sonrasında yedek satırlarla yer değiştirilmektedir.

numaralı bloklardakine göre daha az olduğunu göstermektedir.



Şekil 6.4: Tüm DRAM modüllerinde en yüksek entropiye sahip DRAM bölütlerinin önbellek bloklarının ortalama entropisi. Hata çubukları bu değerlerin tüm DRAM modüllerinde aldıkları aralığı göstermektedir.

Entropi nitelendirme incelememizin sonucunda QUAC işlemlerinin yarattığı entropinin DRAM dizisine (DRAM kümeleri, bölütleri, önbellek blokları) düzensiz bir şekilde yayıldığını gözlemlemekteyiz. Bu gözlem DRAM kümesinin mikromimari özellikleri (örneğin, bölütlerin algılama yükselteçlerinden uzaklıkları), üretim sürecinden kaynaklanan sistematik farklılıklar [132] ve üretim sonrası satır tamiri ile açıklanabilir.

6.2 QUAC ile Gerçek Rastgele Sayı Üretilmesi

QUAC işlemlerinin, art arda uygulandıklarında DRAM algılama yükselteçlerinde gerçek rastgele değerler oluşturduğunu göstermek için bir SoftMC deneyi gerçekleştirmekteyiz. Deneyimiz üç adımdan oluşmaktadır: *(i)* DRAM bölütünün başlangıç değerleri bir veri örüntüsü ile belirlenir, *(ii)* DRAM bölütünde bir QUAC işlemi yürütülür ve algılama yükselteçlerinde rastgele sayılar oluşturulur, *(iii)* DRAM bölütünden veri okunur. Deneyimiz ile her yinelemede tüm DRAM bölütlerindeki tüm algılama yükselteçlerinden birer bit elde etmekteyiz. Toplamda bir milyon yineleme gerçekleştirip her algılama yükseltecinden 1 Mb boyutunda rastgele bit dizileri elde etmekteyiz. Entropi incelememizin sonuçlarına göre QUAC tarafından tüm algılama yükselteçlerinde üretilen rastgele değerler ya mantık-1 ya da mantık-0 değerine eğilim göstermektedir. Bu sebeple çeşitli sonradan işleme yöntemleri (Von Neumann Corrector [146] ve SHA-256 [127]) kullanmaktayız.
QUAC ile üretilen bit dizilerindeki rastgeleliği iyileştirmek (eğilimi azaltmak) için Von Neumann Corrector (VNC) [146] yöntemini kullanmaktayız. VNC ilk olarak diziyi iki bitlik gruplara bölmekte, sonrasında ise üç dönüşümden birini uygulamaktadır: *(i)* eğer iki bit de aynı değere sahipse diziden çıkar, *(ii)* dizideki ilk bit mantık-0, ikinci bit mantık-1 ise iki bitlik grubu diziden çıkar yerine mantık-1 ekle, *(iii)* dizideki ilk bit mantık-1, ikinci bit mantık-0 ise iki bitlik grubu diziden çıkar yerine mantık-0 ekle. Örneğin, “0010” bit dizisi VNC uygulandıktan sonra “0” olmaktadır.
NIST istatistiksel testlerini (STS) [36] kullanarak gerçek rastgele sayı üretecimizin rastgeleliğini test etmekteyiz. NIST STS, test edilen bir bit dizisinin rastgele olduğunu belirten özel bir sıfır hipotezininin (H_0) doğruluğunu denetleyen birçok istatistiksel test

barındırmaktadır. STS rastgele değerin üzerinde yürüttüğü her istatistiksel test için bir p-value çıktı vermektedir. Eğer bir p-value, α ile gösterilen belirli bir önem düzeyinden yüksekse o test için H_0 doğru olmaktadır. Bir diğer deyişle eğer bir testin p-value değeri α'dan yüksek ise o teste göre girdi olarak verilen sayı rastgeledir. Testlerimizde α'yı, NIST STS tanımlamasında önerilen önem düzeyi aralığına ([0,01, 0,001]) uygun bir şekilde 0,001 olarak belirlemekteyiz.

Çizelge 6.1: NIST STS rastgelelik testleri sonuçları.

NIST STS Testi	**VNC*** **(p-value)**	**SHA-256 (p-value)**
monobit	0,430	0,500
frequency_within_block	0,408	0,528
runs	0,335	0,558
longest_run_ones_in_a_block	0,564	0,533
binary_matrix_rank	0,554	0,548
dft	0,538	0,364
non_overlapping_template_matching	>0,999	0,488
overlapping_template_matching	0,513	0,410
maurers_universal	0,493	0,387
linear_complexity	0,483	0,559
serial	0,355	0,510
approximate_entropy	0,448	0,539
cumulative_sums	0,356	0,381
random_excursion	0,164	0,466
random_excursion_variant	0,116	0,510

*VNC: Von Neumann Corrector

Bir DRAM bölütündeki her algılama yükseltecinden (her DRAM bölütünde 64K tane vardır) bit dizileri toplamaktayız. Her DRAM modülünde 8K DRAM bölütü test etmekteyiz. Testlerimizin sonucunda ise 22 algılama yükseltecinden elde edilen 1Mbit bit dizilerinin tüm NIST STS testlerini geçtiğini görmekteyiz.
Tablo 6.1'de tüm 15 NIST STS testini geçen iki çeşit bit dizisi için ortalama p-value değerlerini göstermekteyiz: *(i)* Von Neumann Corrector çıktısı ile elde edilen bit dizileri, *(ii)* Bölüm 5.2'de anlatılan, kriptografik özet fonksiyonu ile elde edilen bit dizileri. Rastgelelik testlerinin sonucunda QUAC'ın gerçek rastgele sayı dizilerinden ayırt edilemeyecek rastgele sayılar ürettiğini gözlemlemekteyiz. Kriptografik özet fonksiyonu ile elde edilen sonuçları Bölüm 7.1'de tartışmaktayız.

7. QUAC-TRNG'NİN DEĞERLENDİRMESİ

QUAC-TRNG'nin *(i)* yüksek nitelikli gerçek rastgele bit dizileri ürettiğini ve *(ii)* son teknoloji DRAM GRSÜlerinden daha iyi başarım gösterdiğini göstermek için gerçek DRAM yongaları kullanarak deneyler ve benzetimler yürütmekteyiz.

7.1 QUAC-TRNG'nin Niteliği

QUAC-TRNG'nin yüksek nitelikli rastgele bit dizileri ürettiğini göstermek için deneysel olarak üç DRAM modülünden (24 DRAM yongası) toplam dokuz adet 1 Gb bit dizisi elde etmekteyiz.[7] Sonuçlarımız tüm bit dizilerinin bütün NIST STS testlerini geçtiğini göstermektedir.

Her bir bit dizisini beş aşamada elde etmekteyiz: *(i)* DRAM bölütünün başlangıç değerlerini en yüksek ortalama entropiye yol açan veri örüntüsü ("0111") ile belirlemekte, *(ii)* DRAM bölütünde bir QUAC işlemi gerçekleştirmekte, *(iii)* DRAM bölütünü okumakta, *(iv)* DRAM önbellek bloğu entropi nitelendirmesinde elde ettiğimiz sonuçlara göre okunan DRAM bölütünü her biri toplam 256-bit Shannon entropisine sahip olacak sürekli parçalara bölmekte ve *(v)* her 256-bit entropilik bloğu SHA-256 özet fonksiyonuna girdi olarak verip 256-bitlik rastgele sayılar elde etmekteyiz.

Her yüksek entropili DRAM bölütünden topladığımız 1 Gb bit dizilerini 1 Mblik parçalara bölmekte ve her DRAM bölütü için NIST STS testlerini kullanarak 1024 rastgele sayı dizisi test etmekteyiz. Test sonuçları rastgele sayı dizilerinin $99,28\%$'inin geçtiğini göstermektedir. Bu geçme oranı NIST standartında belirtilen kabul edilebilirlik sınırından $(98,84\%)$ yüksektir [36].[8] Tablo 6.1, sütun "SHA-256", bu testler ile elde ettiğimiz tüm p-değerlerini göstermektedir. Bu deneyimizden QUAC-TRNG'nin yüksek nitelikli rastgele sayı dizileri ürettiği sonucuna varmaktayız.

7.2 QUAC-TRNG'nin Hızı

XT'nin bir DRAM modülünde elde edeceği rastgele sayı üretme hızını analitik olarak *(i)* en yüksek entropiye sahip bölütteki 256 bitlik girdi bloklarının sayısı (*SGB*: SHA Girdi Blokları) ve *(ii)* bir QUAC işleminin gecikmesine (G) bağlı hesaplamaktayız. QUAC-TRNG bir DRAM kümesinde G nanosaniyede bir $256 \times SGB$ rastgele bit üretmektedir. QUAC-TRNG'nin hızı ise buna göre $(256 \times SGB)/(G \times 10^{-9})$ saniye başına rastgele bit olarak hesaplanmaktadır.

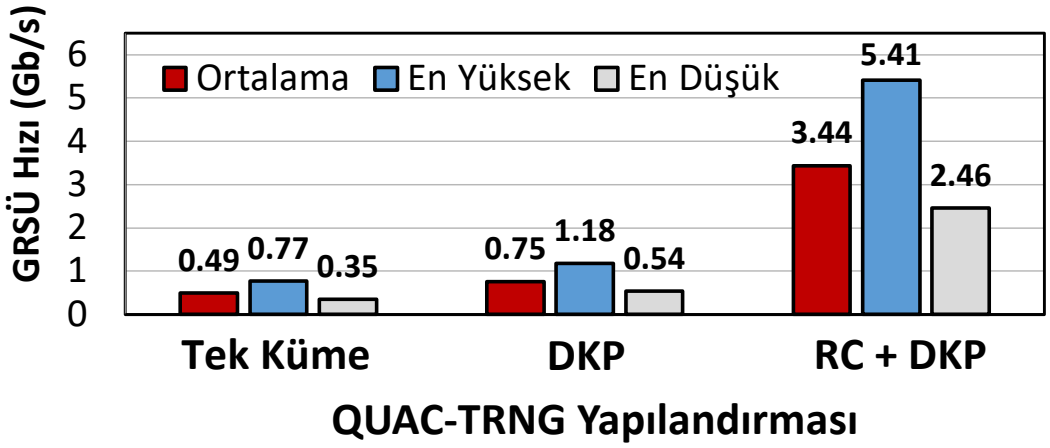
Her modülün *SGB* değerini modüldeki en yüksek entropiye sahip DRAM bölütü ile direkt olarak, $\lfloor bolut\_entropisi/256 \rfloor$ şeklinde hesaplamaktayız. L değerini ise (i) DRAM bölütündeki satırların başlangıç değerlerini veri örüntüleri ile belirlemek için, (ii) QUAC işlemi gerçekleştirmek için ve (iii) rastgele değerleri DRAM algılama yükselteçlerinden okumak için gereken DRAM komutlarını zamanlayarak hesaplamaktayız.

[7] Deneylerimizde makul bir yürütme zamanına sahip olmak için toplam dokuz bit dizisi elde etmekteyiz.

[8] $(1-\alpha) \pm 3\sqrt{\alpha(1-\alpha)/k}$ formülüne göre. K rastgele sayı dizisi sayısını (1024) ve α önem düzeyini $(0,005)$ göstermektedir.

QUAC-TRNG'nin gecikmesinin (L) büyük bir kısmı dört DRAM satırının başlangıç değerlerini belirlemek için gereken DRAM komutlarının yürütülmesinden oluşmaktadır. Bu başlangıç değerlerini belirlemek için gereken zamanı azaltmak ve dolayısıyla QUAC-TRNG'nin hızını artırmak için iki iyileştirme uygulamaktayız. İlk olarak DRAM küme düzeyindeki paralellikten faydalanmak için QUAC komutlarını farklı DRAM kümelerinde aynı anda yürütmekteyiz. DDR4'e özel olarak DRAM komutlarını DRAM küme grupları arasında dağıtmakta, kısa süreli DRAM ACT-ACT (*tRRD_S*) zamanlama parametresinden faydalanmaktayız. İkinci olarak ComputeDRAM'de [53] tanıtılan RowClone-tabanlı [87] DRAM içinde kopyalama işlemleri ile DRAM bölütlerinin başlangıç değerlerini belirlemekteyiz. DRAM içinde kopyalama gerçekleştirerek başlangıç değerleri belirleme gecikmesini oldukça azaltmaktayız.

Şekil 7.1, QUAC-TRNG'nin gerçek rastgele sayı üretme hızını üç yapılandırma için göstermektedir: *(i)* Tek Küme (*-ing. One Bank*), QUAC-TRNG'nin yalnızca bir DRAM kümesi kullandığımızdaki hızını, *(ii)* DKP (Küme Grubu Paralelliği, *-ing. Bank Group Parallelism*) QUAC-TRNG'nin farklı DRAM küme gruplarını kullanarak DRAM komut gecikmelerini birbirlerinin arkasında sakladığımızdaki hızını, *(iii)* RC (RowClone) + DKP QUAC-TRNG'nin DRAM bölütlerinin başlangıç değerlerini DRAM içinde kopyalama işlemlerini kullanarak belirlediğimizde ve küme grubu paralelliğinden faydalandığımızdaki hızını göstermektedir. Şekilde tüm modüller için ortalama, en yüksek ve en düşük hız değerlerini göstermekteyiz.

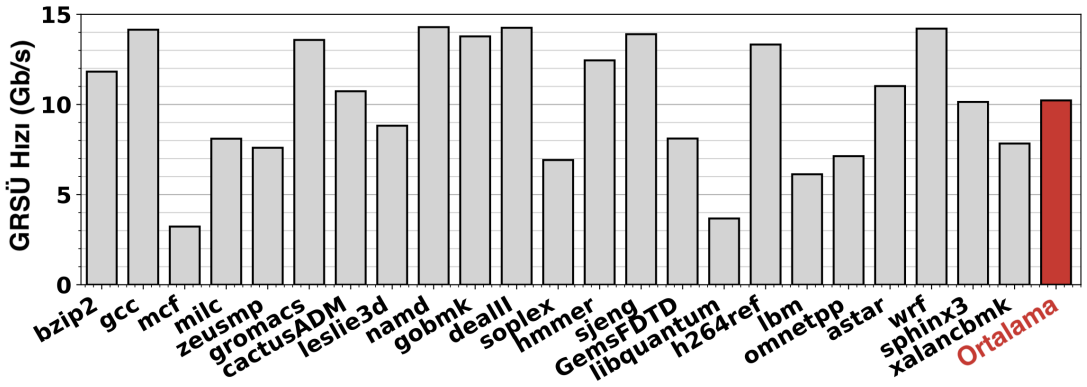


Şekil 7.1: QUAC-TRNG'nin üç yapılandırmasının (Tek Küme, DKP, RC + DKP) rastgele sayı üretim hızı

Ortalamada *Tek Küme*'nin 0,49 Gb/s, *DKP*'nin 0,75 Gb/s, ve *RC + BGP*'nin 3,44 Gb/s hızında rastgele sayı ürettiğini gözlemlemekteyiz. QUAC-TRNG'nin gerçek rastgele sayı üretim hızı, her modülün en yüksek entropiye sahip olan bölütü farklı olduğundan DRAM modülleri arasında farklılık göstermektedir. QUAC-TRNG'nin hız incelemesi sonunda DRAM içinde kopyalama işlemlerinin QUAC-TRNG'nin yüksek hızda rastgele sayı üretmesi için gerekli olduğu sonucuna varmaktayız.

7.3 Sistem Başarım incelemesi

QUAC-TRNG'nin bir sistemde beraber çalıştığı uygulamalara ayrılan toplam bellek hızını azaltmadan erişebileceği en yüksek rastgele sayı üretim hızını ölçmek için SPEC2006 programlarından elde ettiğimiz bellek izlerini kullandığımız bir benzetim deneyi gerçekleştirmekteyiz. Deneyimizde dört DDR4 DRAM kanalına sahip 3,2 GHz çekirdek içeren bir sistemin benzetimini Ramulator [147, 148] ile gerçekleştirip her bellek kanalının ne kadar süre boyunca herhangi bir bellek isteğine hizmet etmediğini (bir diğer deyişle, boşta) bulmaktayız. Bellek kanallarının boş geçirdiği zamanları QUAC-TRNG için gereken DDR4 komutları ile doldurmaktayız. Şekil 7.2, QUAC-TRNG'nin her SPEC2006 iş parçası yürütülürkenki başarımını (rastgele sayı üretim hızını) göstermektedir.[9] QUAC-TRNG ortalama 10,2 Gb/s hızla rastgele sayı üretmektedir. QUAC-TRNG, iş parçasından iş parçasına en yüksek 14,3 Gb/s, en düşük 3,22 Gb/s rastgele sayı üretim hızına sahip olabilmektedir. DRAM kanallarının boş geçireceği zamanları kullanarak, QUAC-TRNG'nin ortalamada Bölüm 7.2'de bulunan empirik ortalama hızın (13,76 Gb/s) %74,13'ünü elde edebildiğini gözlemlemekteyiz.



Şekil 7.2: SPEC2006 programları için boş DRAM arayüz çevrimlerinde elde edilebilecek TRNG hızı

7.4 Önceki Çalışmalar ile Karşılaştırma

Bu bölümde QUAC-TRNG'yi yüksek hızlı ($> 100Mb/s$) DRAM rastgele sayı üreteçleri ile karşılaştırmaktayız. Her çalışmanın gerçek rastgele sayı üretim hızını ve gecikmesini Bölüm 7.3'te tarif edilen dört DRAM kanallı benzetim sistemine ölçeklendirmekteyiz. incelememizin özeti, Bölüm 10'da tartıştığımız düşük hızlı ($< 100Mb/s$) DRAM gerçek rastgele sayı üreteçlerini de içerecek bir şekilde Tablo 7.1'de gösterilmektedir.

QUAC-TRNG'yi, iki son teknoloji yüksek hızlı DRAM GRSÜ [22, 24] ile titizlikle karşılaştırmaktayız. Karşılaştırdığımız GRSÜler için *(i)* en yüksek rastgele sayı üretim hızını ve *(ii)* 256-bit rastgele sayı üretmek için en düşük gecikmeyi hesaplamaktayız. Bunu gerçekleştirmek için GRSÜlerin gerektirdiği DRAM komut dizilerini sıkışık bir şekilde (bir diğer deyişle, komutlar zamanlama parametrelerini ihlal etmeyecek ve olabildiğinde küçük zamanlama parametreleri ile birbirlerinden ayrılacaklar) zamanlamaktayız.

[9] Tüm kanallarda her küme grubundan bir küme kullanmaktayız.

Çizelge 7.1: Önceki DRAM GRSÜlerin QUAC-TRNG ile karşılaştırılmasının özeti.

Çalışma	Entropi Kaynağı	GRSÜ Hızı	256-bit GRSÜ Gecikmesi
QUAC-TRNG	Dörtlü ACT	13,76 Gb/s	274 ns
Talukder+ [24]	Ön Yükleme Hatası	0,68 - 6,13 Gb/s	249 ns - 201 ns
D-RaNGe [22]	Satır Açma Hatası	0,92 - 9,73 Gb/s	260 ns - 36 ns
D-PUF [23]	Tutma Hatası	0,20 Mb/s	40 s
DRNG [34]	Başlangıç Değeri	N/A	700 us
Keller+ [25]	Tutma Hatası	0,025 Mb/s	40 s
Pyo+ [26]	DRAM Komut Dizileri	2,17 Mb/s	112,5 us

7.4.1 D-RaNGe

D-RaNGe açma gecikmesi (*tRCD*) tamamlanmadan önce DRAM önbellek bloklarının okunması ile oluşan hatalardan faydalanarak gerçek rastgele sayı üretmektedir [22]. D-RaNGe'in hızını ve gecikmesini iki yapılandırma altında incelemekteyiz: *(i) D-RaNGe-Temel (-ing. D-RaNGe-Basic)* yapılandırmasında D-RaNGe'i önerildiği gibi değerlendirmekte, *(ii) D-RaNGe-Geliştirilmiş (-ing. D-RaNGe-Enhanced)* yapılandırmasında *tRCD* hatalarının entropisini gerçek DDR4 aygıtları kullanarak nitelendirip sonradan işleme kullanan D-RaNGe'in hızını tahmin etmekteyiz.

D-RaNGe-Temel. Bir DRAM önbellek bloğunda önce açma hatası oluşturup sonra o bloğu okumak için gereken DDR4 komutlarını sıkışık bir şekilde zamanlayarak D-RaNGe-Temel'in hızını hesaplamaktayız. incelememizde D-RaNGe-Temel'i DDR4 aygıtlardaki küme grubu paralelliğinden faydalanması için iyileştirmekteyiz. D-RaNGe bir DRAM önbellek bloğunda dört taneye kadar gerçek rastgele sayı üretici hücre gözlemlemektedir. incelememizde iyimser bir şekilde gözlemlenmiş en yüksek rastgeleliğin (bir önbellek bloğunda dört bit) oluşacağını varsaymakta ve buna göre D-RaNGe-Temel'in hızını hesaplamaktayız. DRAM içinde kopyalama işlemlerini kullanarak D-RaNGe'i hızlandırmamaktayız çünkü D-RaNGe'in yalnızca bir DRAM önbellek bloğunun başlangıç değerini belirlemesi yeterlidir (DRAM yazma komutları ile hızlıca gerçekleştirilebilir) ve bu yüzden DRAM içinde kopyalama işlemlerinin yüksek paralelliğinden faydalanması gerekmemektedir. Bu gözlemlerimizden ve varsayımlarımızdan kaynakla D-RaNGe-Temel'in hızını 916,9 Mb/s, 256-bit rastgele sayı üretim gecikmesini ise 260 nanosaniye olarak hesaplamaktayız.

D-RaNGe-Geliştirilmiş. D-RaNGe-Geliştirilmiş'in gerçek rastgele sayı üretim hızını hesaplamak için 17 gerçek DDR4 modülündeki 136 DDR3 yongasını SoftMC ile test etmekte ve açma gecikmesi hatalarının ortalama önbellek bloğu entropisini bulmaktayız. Test edilen bir yonga için her DRAM kümesindeki bir DRAM önbellek bloğu başına testimiz her yinelemede: *(i)* bir DRAM satırını "0" veri örüntüsü ile doldurmakta (D-RaNGe'de en yüksek rastgeleliğe yol açmaktadır [22]) ve *(ii)* DRAM aygıtına düşük *tRCD* ile erişim gerçekleştirmektedir. Bu deneyi *1000* kez tekrar edip her önbellek bloğunun entropisini hesaplamaktayız. Her DRAM modülü için en yüksek önbellek bloğu entropisini kaydetmekteyiz. Her DRAM modülündeki en yüksek entropinin ortalamasını hesaplamakta ve 256-bit entropi toplayabilmek için D-RaNGe-Geliştirilmiş'in DRAM aygıtına kaç kez erişmesi gerektiğini bulmaktayız. D-RaNGe-

Geliştirilmiş bir önbellek bloğundan ortalama 46,55 bit entropi elde etmektedir. Buna bağlı olarak D-RaNGe-Geliştirilmiş'in 256-bit rastgele sayı üretmesi için *6* düşük *tRCD*'li erişim yapması yeterlidir. Daha adil bir karşılaştırma için QUAC-TRNG ile aynı sonradan işleme mekanizmasını (SHA-256) D-RaNGe-Geliştirilmiş'e de uygulamaktayız. Sonradan işleme gerçekleştiren D-RaNGe-Geliştirilmiş 9,73 Gb/s hıza ulaşabilmektedir. D-RaNGe-Geliştirilmiş'in 256-bit rastgele sayı üretim gecikmesi 36 nanosaniye olmaktadır (SHA-256 gecikmesi dahil). SHA-256 sonradan işleme fonksiyonunun, D-RaNGe'in bir önbellek bloğunun daha büyük kısımlarını entropi kaynağı olarak kullanmasına izin verdiğinden hızını büyük ölçüde artırabileceğini gözlemlemekteyiz.

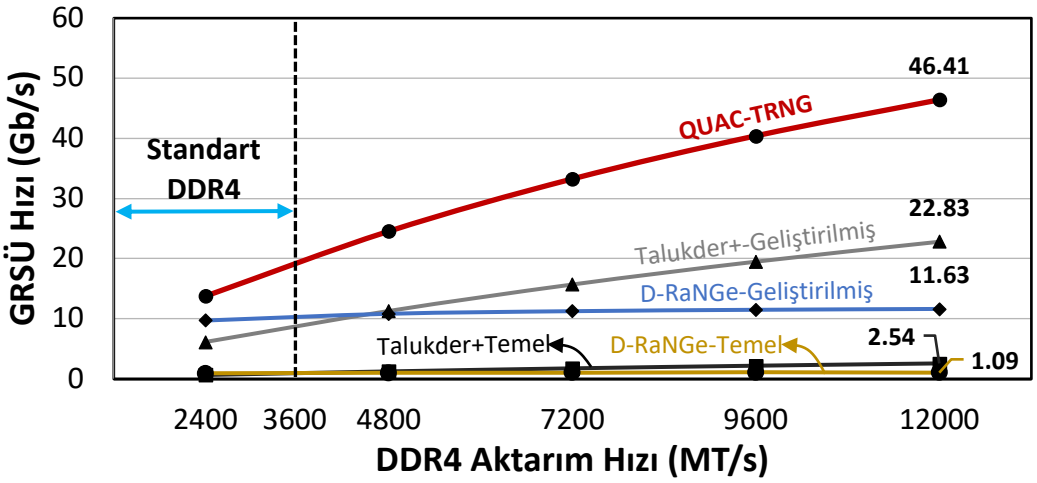
7.4.2 Talukder'in çalışması (Talukder+)

Talukder+ bit hatları $V_{DD}/2$ değerine ön yüklenmeden DRAM satırlarının açılması ile oluşan hataları (ön yükleme gecikmesi hatası) kullanarak gerçek rastgele sayı üretmeyi önermektedir. Yazarlar SHA-256 kullanarak DRAM aygıtından elde edilen bit dizilerini sonradan işlemektedir. Talukder+'ın mekanizması (i) ön yükleme gecikmesi hatalarını birden çok DRAM satırında gerçekleştirmekte, (ii) rastgele oluşan hataları birden çok hücrede biriktirmekte, (iii) bu hücreleri okumakta ve (iv) okunan veriyi SHA-256 özet fonksiyonunu kullanarak sonradan işlemektedir. Talukder+'ın algoritmasını küme grubu düzeyi paralellikten faydalanacak şekilde iyileştirmekte ve ön yükleme gecikmesi hatalarını oluşturmadan önce DRAM satırlarının başlangıç değerlerini belirlemek için DRAM içinde kopyalama işlemlerini kullanmaktayız. Talukder+'nın mekanizmasının hızını iki yapılandırmada incelemekteyiz: (i) *Talukder+-Temel* yapılandırmasında mekanizmanın hızını yazarların rastgele hücre incelemesini kullanarak hesaplamakta, (ii) *Talukder+-Geliştirilmiş* yapılandırmasında ise mekanizmanın hızını ön yükleme gecikmesi hatalarının gerçek DDR4 aygıtlarında oluşturduğu entropiyi nitelendirerek hesaplamaktayız.

Talukder+-Temel. Talukder+-Temel'in GRSÜ hızını yazarların gösterdiği sonuçlardan hesaplamaktayız. Yazarlar her DRAM satırında ortalama 130,6 rastgele hücre bulunduğunu raporlamaktadır. SHA-256 fonksiyonuna giriş olarak 256-bit entropili bir blok verebilmek için Talukder+ mekanizması 3 DRAM satırı okumalıdır. Buna bağlı olarak Talukder+ mekanizmasının rastgele sayı üretim hızı 681,2 Mb/s, 256-bit rastgele sayı üretim gecikmesi ise 249 nanosaniyedir.

Talukder+-Geliştirilmiş. Talukder+-Geliştirilmiş'in GRSÜ hızını hesaplamak için SoftMC kullanarak 17 gerçek DDR4 modülündeki 136 DDR4 yongalarını kullanarak ön yükleme gecikmesi hatalarındaki ortalama satır entropisini ölçmekteyiz. Her DRAM modülündeki en yüksek satır entropisini ve bu en yüksek satır entropilerinin her modül için ortalamasını bulup Talukder+-Geliştirilmiş'in 256 bitlik entropiye sahip SHA-256 girdi bloklarını oluşturabilmesi için kaç DRAM satırı okuması gerektiğini hesaplamaktayız. Talukder+-Geliştirilmiş düşük *tRP* erişimlerinden sonra DRAM satırlarından ortalama 1023,64 bit entropi elde edebilmektedir. Talukder+-Geliştirilmiş her DRAM satırından ortalama 3 SHA-256 girdi bloğu elde edebilmektedir. Buna bağlı olarak Talukder+-Geliştirilmiş'in hızını 6,13 Gb/s, 256-bit rastgele sayı üretim gecikmesini ise 201 nanosaniye olarak hesaplamaktayız.

Şekil 7.3 Talukder+-Temel/Geliştirilmiş, D-RaNGe-Temel/Geliştirilmiş ve QUAC-TRNG'nin ortalama hızını göstermektedir. Ayrıca bu mekanizmaların farklı DDR4 veri aktarım hızlarına (MT/s) iz düşümü de gösterilmektedir.

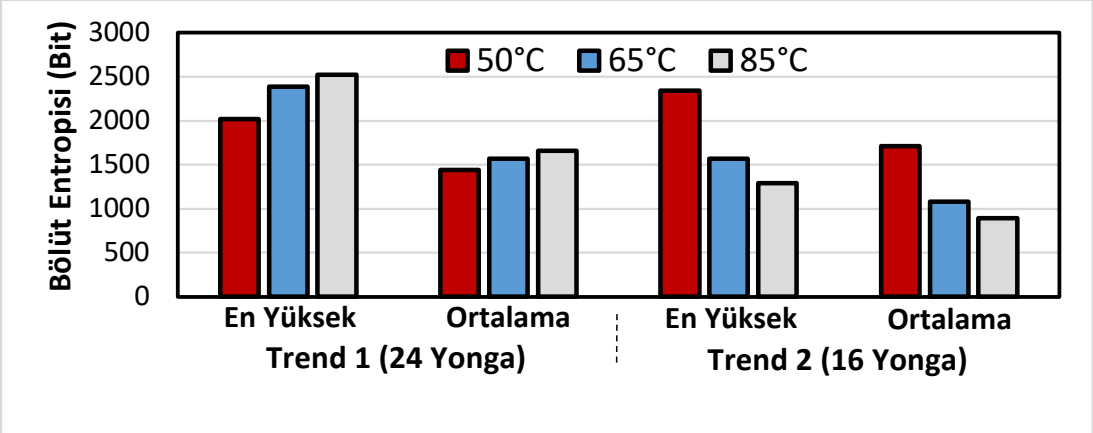


Şekil 7.3: DRAM GRSÜlerin DDR4 veri aktarım hızına iz düşümü. Grafikte DDR4 standartlarının ötesinde aktarım hızları da gösterilmektedir.

Figür üzerinde iki gözlem gerçekleştirmekteyiz. İlk olarak, D-RaNGe artan DRAM aktarım hızından faydalanamamaktadır çünkü D-RaNGe'in yüksek hızda rastgele sayı üretmesi için sürekli DRAM aygıtında açma gecikmesi hatası oluşturması gerekmektedir. Bu yüzden D-RaNGe'in hızı DRAM erişim hızı ile sınırlanmaktadır ve artan DRAM aktarım hızı ile ölçeklenmemektedir. İkinci olarak, Talukder+ ve QUAC-TRNG'nin DRAM aktarım hızına ölçeklenebilir olduğunu gözlemlemekteyiz. QUAC-TRNG Talukder+ ve D-RaNGe'in temel (geliştirilmiş) yapılandırmalarından DDR4 2400 MT/s aktarım hızında sırasıyla **20,20**× (2,24×) ve **15,08**× (1,41×) daha hızlı rastgele sayı üretmektedir. Gelecek 12 GT/s aktarım hızında ise QUAC-TRNG Talukder+ ve D-RaNGe'in geliştirilmiş yapılandırmalarından **2,03**× ve **3,99**× hızlı gerçek rastgele sayı üretmektedir.
QUAC-TRNG, D-RaNGe ve Talukder+'a kıyasla daha uzun gecikmeye sahip olsa da bu gecikme üretilen rastgele sayıların bir ara belleğe kaydedilmesi ile saklanabilir. Özel GRSÜ devrelerine sahip günümüz bilgisayarları halihazırda rastgele sayıları kaydetmek için ara bellekler bulundurmaktadırlar [149]. QUAC-TRNG, son teknoloji diğer DRAM GRSÜlere kıyasla bu ara bellekleri çok daha hızlı doldurabilir.

8. HASSASLIK İNCELEMESİ

Sıcaklığa Bağımlılık. Bit hattı entropisini 40 gerçek DRAM yongasını kullanarak 50°C, 65°C ve 85°C sıcaklıklarda kaydederek sıcaklığın QUAC işlemlerindeki entropiye etkisini incelemekteyiz. Sıcaklığa bağımlılık ile ilgili iki trend gözlemlemekteyiz: *Trend-1*, bit hattı entropisi sıcaklıkla beraber artmaktadır (24 yongada), *Trend-2*, bit hattı entropisi sıcaklık arttıkça azalmaktadır (16 yongada). Trend-1 ve trend-2 davranışına sahip yongalarda ayrı ayrı en yüksek ve ortalama DRAM bölütü entropisini (bir DRAM bölütündeki tüm bit hattı entropilerinin toplamı) hesaplamaktayız. Şekil 8.1, 50°C, 65°C ve 85°C sıcaklıklarda gözlemlediğimiz en yüksek ve ortalama bölüt entropilerini göstermektedir.



Şekil 8.1: Farklı sıcaklıklarda en yüksek ve ortalama bölüt entropileri

QUAC işlemlerindeki entropinin sıcaklığa bağlı değiştiğini gözlemlemekteyiz. 50°C, 65°C ve 85°C sıcaklıklarda trend-1 davranışına sahip yongalarda en yüksek (ortalama) entropinin sırasıyla 2019,6 (1442,0), 2389,8 (1569,5) ve 2520,1 (1659,6) olduğunu gözlemlemekteyiz. Aynı sıcaklıklarda trend-2 davranışına sahip yongalarda ise en yüksek (ortalama) entropinin sırasıyla 2344,2 (1710,6), 1565,8 (1083,1) ve 1293,5 (892,5) olduğunu gözlemlemekteyiz. Sonuç olarak sıcaklığa bağlı entropinin değişmesinden ötürü QUAC-TRNG'yi barındıracak sistemlerin sıcaklık değişikliklerine dikkat etmesini önermekteyiz.

SHA-256 girdi bloklarında sıcaklık değişimlerinde 256-bit toplam entropiyi yakalayabilmek için bellek denetleyicisi bir *sütun adresi kümesi* listesi tutmaktadır. Bu listenin başlangıç değerleri, DRAM bölütlerinin farklı sıcaklıklarda entropilerinin bir seferlik nitelendirmesi ile belirlenmektedir. QUAC-TRNG, DRAM sıcaklığına (örn., sıcaklık sensörleri ile ölçülen [37]) göre bu listedeki bir elemana erişip bir sütun adresi kümesi elde etmektedir. Bu kümedeki her adres, DRAM aygıtında toplam 256-bit entropiye sahip sürekli bir önbellek blok adres uzayının başlangıcına işaret etmektedir. QUAC-TRNG bu kümeleri kullanarak yüksek entropili DRAM bölütünden okunan veriyi 256-bit entropiye sahip parçalara ayırmaktadır. Bu sayede QUAC-TRNG, SHA-256 girdi bloklarının her sıcaklıkta daima en az 256-bit entropiye sahip olmasını sağlamaktadır.

Zamana Bağımlılık.
QUAC-TRNG'nin ürettiği rastgele sayıların niteliğinin zamana bağlı değişip değişmediğini anlamak için 30 günlük bir zaman diliminin başında ve sonunda QUAC işlemleri ile üretilen entropiyi 40 DRAM yongası kullanarak ölçmekteyiz. DRAM bölütlerinin başlangıç değerleri en yüksek entropiye yol açan veri örüntüsü ("0111", Bölüm 6.1.2) ile belirlendiğinde ortalama bölüt entropisi önemli ölçüde değişmemektedir. 30 günlük zaman diliminin başlangıcında ve sonunda 8192 bölütün ortalama entropileri, beş modül içinde ortalama (en yüksek, en düşük) %2,4 (%5,2, %0,9) değişiklik göstermektedir. Sonuç olarak QUAC işlemlerince üretilen entropinin bir aylık zaman içinde kayda değer ölçüde değişmediğini gözlemlemekteyiz. Dolayısıyla QUAC-TRNG'de nitelendirme ile elde edilen entropi değerleri en azından bir ay geçerli olmaktadır. En kötü durumda QUAC-TRNG'nin DRAM aygıtlarını ayda bir kez nitelendirmesi gerekmektedir.

9. QUAC-TRNG'NİN GERÇEK BİR SİSTEMDE UYGULANMASI

Bu bölümde QUAC-TRNG'nin gerçek bir sistemde nasıl uygulanabileceğini tartışmaktayız. QUAC-TRNG, rastgele sayıları (i) dört farklı küme grubundaki yüksek entropili DRAM bölütlerinde QUAC işlemleri gerçekleştirerek ve (ii) QUAC işlemlerinin sonuçlarını SHA-256 özet fonksiyonu ile sonradan işleyerek üretmektedir.

Sonradan İşleme. QUAC-TRNG bir kriptografik özet fonksiyonu kullanarak QUAC işlemleri ile üretilen rastgele bit dizilerini sonradan işlemektedir. QUAC-TRNG'yi SHA-256 kullanarak değerlendirmemizin sebebi SHA-256'nın güvenli bir kriptografik özetk fonksiyonu olması ve donanımda düşük alan ve gecikme maliyeti ile verimli bir şekilde gerçeklenebilmesidir [150–152]. Bu, SHA-256'yı bellek denetleyicisinde gerçeklenmeye uygun kılmaktadır. Değerlendirmemizde SHA-256'nın maliyetini geçmiş bir çalışmanın raporladığı sonuçlara göre hesaplamaktayız [150]. Buna göre SHA-256 devresi 65 saat vuruşu gecikmeye (5,15 GHz saat vuruş sıklığında) sahiptir, 19,7 Gb/s hız ile sonradan işleme gerçekleştirebilmektedir ve 7 nm işlem teknolojisinde 0,001 mm^2 devre alanı gerektirmektedir.

QUAC-TRNG Kullanıcı Uygulama Arayüzü. QUAC-TRNG, QUAC işlemlerini kullanarak rastgele sayı üretmektedir. QUAC işlemleri gerçekleştirmek için bellek denetleyicisinin düşük *tRAS* ve *tRP* zamanlama parametreleri kullanarak DRAM aygıtına bir ACT $\rightarrow$ PRE $\rightarrow$ ACT komut dizisi göndermesi gerekmektedir. Bir QUAC isteği geldiğinde bellek denetleyicisi DRAM aygıtının uygun olup olmadığını (ör., diğer uygulamalara ayrılan DRAM bant genişliğini düşürmeden QUAC işlemleri gönderilip gönderilemediğini) denetlemekte ve bu komut dizisini düşük zamanlama parametreleri ile DRAM aygıtına göndermektedir. Bahsi geçen bu işlevsellik bellek denetleyici devresine küçük bir durum makinesi eklenerek gerçekleştirilebilir. Uygulamaların rastgele isteklerindeki gecikmeleri azaltmak için D-RaNGe'de önerildiği gibi bellek denetleyicisi belirli zaman aralıkları ile QUAC işlemleri gerçekleştirip üretilen rastgele sayıları bir ara bellekte saklayabilir [22]. Bu sayede bir uygulamanın rastgele sayı isteği, DRAM komutları göndermeden ara bellekten rastgele sayıların okunması ile hızlıca karşılanabilir.

QUAC-TRNG'nin gerçek bir sistemde kullanılması için sistem tasarımcısının QUAC-TRNG arayüzünü kullanıcı uygulamalarına açması gerekmektedir. Bunun için çok sayıda yöntem bulunmaktadır. Örneğin bellek eşlemeli denetim yazmaçlarını kullanmak, yardımcı işlemci ve giriş çıkış buyruklarını kullanmak, buyruk kümesi mimarisine özel eklemeler yapmak. Tasarım hedeflerine uygun en iyi yöntemi uygulamayı sistem tasarımcısına bırakmaktayız.

Bellek Kaybı. QUAC-TRNG dört DRAM küme grubundaki birer kümede az sayıda DRAM satırını rastgele sayı üretimi için ayırmaktadır. QUAC-TRNG (i) bir DRAM bölütünü (dört DRAM satırı) QUAC işlemleri gerçekleştirmek için ve (ii) iki DRAM satırını DRAM bölütlerinin başlangıç değerlerini belirleyen işleçleri saklamak için ayırmaktadır. DDR4 bant genişliğinden tamamiyle faydalanabilmek için QUAC-TRNG dört farklı küme grubundaki bölütleri aynı anda açmakta ve ardışık bir şekilde bu küme gruplarında rastgele bit dizilerini okumaktadır. Tüm bunlar için

QUAC-TRNG'nin toplamda her DRAM kanalında 24 DRAM satırı ayırması gerekmektedir. Bu da 192 KiB ayrılmış alana ya da 8 GiB bir DDR4 modülünün kapasitesinin %0,002'sine denk gelmektedir.

Devre Alanı Gereksinimi. QUAC-TRNG dört küme grubundaki yüksek entropili bölütlerin başlangıcına işaret etmek için *dört DRAM satır adresi* ve DRAM içinde kopyalama işlemlerinin işleçlerine işaret etmek için *sekiz DRAM satır adresi* kaydetmektedir. QUAC-TRNG ayrıca bir DRAM bölütünde 256 bit toplam entropiye sahip bit hatlarını takip etmek için, birbirleri ile çakışmayan önbellek bloğu boyutundaki aralıkların başlangıcına işaret eden *11 DRAM sütun adresi kaydetmektedir*. Bu önbellek blok aralıkları sistem sıcaklığına göre değişmektedir. Devre alanı gereksinimini hesaplarken QUAC-TRNG'nin 10 farklı sıcaklık aralığında çalışacak şekilde yapılandırıldığını varsaymaktayız. QUAC-TRNG satır ve sütun adreslerini kaydetmek için toplam 1316 bit saklama alanına ihtiyaç duymaktadır. Bu saklama alanı için ihtiyaç duyulacak devre alanını CACTI [153] kullanarak modellemekte ve gereken alanın 0,0003 mm^2 olduğunu bulmaktayız. SHA-256 hızlandırıcı çekirdeği dahil edildiğinde, QUAC-TRNG'nin bir 7nm teknolojide gerçeklenmesi için 0,0014 mm^2 devre alanı gerekmektedir. Bu da 7nm teknolojisinde geliştirilmiş bir modern işlemcinin yonga alanının %0,04'üne karşılık gelmektedir [154, 155].

10. GEÇMİŞ ÇALIŞMALAR

Bilindiği kadarıyla QUAC-TRNG, *(i)* dörtlü etkinleştirme (QUAC) işlemlerinin DRAM yongalarında algılama yükselteçlerini yarı kararlı duruma sürükleyerek rastgele sayılar oluşturmasını gösteren ve (ii) bu davranıştan faydalanarak yeni bir gerçek rastgele sayı üreteci tasarlayan ilk çalışmadır. Bölüm 7.4'te iki son teknoloji yüksek hızlı DRAM GRSÜ ile ayrıntılı karşılaştırma gerçekleştirmekteyiz. Bu bölümde diğer geçmiş çalışmalardan bahsetmekteyiz.

10.1 Düşük Hızlı DRAM GRSÜleri

Pyo et al. [26] (Tablo 7.1, Pyo+), DRAM komut zamanlamasındaki tahmin edilemezliği entropi kaynağı olarak kullanarak rastgele sayı üretmektedir. Bölüm 7.3'te tarif ettiğimiz sistemde 8-bit rastgele sayı üretmek için geçen işlemci saat vuruş sayısını (45000) kullanarak Pyo+'nın gerçek rastgele sayı üretim hızını 2,17 Mb/s, gecikmesini ise 112,5us olarak hesaplamaktayız.
Tutma-tabanlı GRSÜler [23, 25], (i) DRAM yenileme işlemlerini duraklatarak entropi kaynağı olarak kullanılan tutma hatalarının [117] oluşmasına sebep olmakta, (ii) DRAM dizisinin tutma hatalarını barındıran parçalarını okumakta, (iii) okunan veriyi özet fonksiyonları kullanarak sonradan işlemekte ve bir rastgele sayı elde etmektedirler.
D-PUF [23] (Tablo 7.1, D-PUF), DRAM aygıtını 4 MiB büyüklüğünde parçalara ayırıp yeterli miktarda entropi oluşturana kadar DRAM yenileme işlemlerini duraklatmaktadır. D-PUF her parçadan okuduğu veriyi SHA-256 özet fonksiyonunu kullanarak sonradan işlemektedir. Bu işlem ile rastgele sayı üretilmesi en az 40 saniye gecikmeye sahip olmaktadır. Dört bellek kanallı, toplam 128 GiB belleğe sahip olan bir sistemde optimistik olarak D-PUF'un hızını hesaplamaktayız. Hesaplamamızda 128 GiB verinin okunması için gereken zamanı da göz ardı etmekteyiz. Tüm DRAM'in %1'i (yaklaşık 327 4 MiB büyüklüğünde parça) rastgele sayı üretimi için ayrıldığında D-PUF 0,002 Mb/s hızda rastgele sayı üretebilmektedir. Tüm 32K parça kullanıldığında bile D-PUF'un hızı yalnızca 0,20 Mb/s olabilmektedir.
Keller+ [25] (Tablo 7.1, Keller+), DRAM aygıtını 1 MiB büyüklüğünde parçalara ayırıp 320 saniye boyunca DRAM yenileme işlemlerini duraklatmaktadır. D-PUF [23] için yaptığımız analize benzer bir analiz ile Keller+'nın 256-bit gerçek rastgele sayı üretim gecikmesini 320 saniye olarak ve hızını yalnızca 0,025 Mb/s olarak hesaplamaktayız.
DRAM Başlangıç Değeri GRSÜleri [34] (Tablo 7.1, DRNG), DRAM aygıtına güç verildiği anda DRAM hücrelerinde oluşan değerleri kullanarak rastgele sayı üretmektedir. Bu GRSÜler hızlı bir şekilde rastgele sayı üretmek için kullanılamaz çünkü DRAM aygıtının kapatılıp açılması gerekmektedir. Bu kategorideki GRSÜlerin gecikmesini DDR4 başlangıç mekanizmasının [156] gecikmesi ile 700 mikrosaniye olacak şekilde hesaplamaktayız.
Bu bölümde anlatılan tüm DRAM GRSÜler çok yavaş rastgele sayı üretmekte ve yüksek gecikmelere sahip olmaktadır. Yavaş GRSÜler günümüz GRSÜ uygulamaları-

nın (örneğin, makine öğrenmesi, kriptografi, bilimsel benzetimler [1, 3–5, 9, 10, 13–20, 56–60]) yüksek hız ihtiyacını karşılamakta başarısız olmaktadır. Buna karşı QUAC-TRNG bu uygulamaların yüksek hız ihtiyacını karşılayabilmektedir.

10.2 Özel Donanım Gerektiren DRAM Tabanlı Olmayan GRSÜler

Çok sayıda geçmiş çalışma özel donanım gerektiren yüksek hızlı gerçek rastgele sayı üreteçleri tasarlamaktadır [21, 28, 61, 62, 64–69, 126, 157–162]. Ne yazık ki özel donanım gerektiren GRSÜlerin özellikle düşük maliyetli bilgisayar sistemlerine ve gelecek bellek içinde hesaplama sistemlerine tümlenmesi masraflıdır. Var olan bazı bilgisayar sistemlerindeki GRSÜler [149, 163, 164] (i) özel donanım gerçekleme için devre alanı tüketmekte (örneğin halka osilatörleri [165]) ve (ii) düşük hızla rastgele sayı üretmektedir. Örneğin, yakın zamanda piyasaya sürülen AMD Zen3 işlemcileri 3,18 Gb/s (çekirdek başına, 4 GHz saat vuruş sıklığında) hızla rastgele sayı üretebilmektedir [166]. Bu da QUAC-TRNG'nin dört kanallı bir sistemde rastgele sayı üretim hızının yalnızca %23,11'ine denk gelmektedir.

10.3 DRAM Aygıtlarında Aynı Anda Birden Çok Satırın Açılması

Ambit [70] ve ComputeDRAM [53]. Seshadri et al. [41, 70, 86, 167] DRAM'de üç satırın aynı anda açılması fikrini öne sürmektedir ve bu işlemin açılan üç satıra çoğunluk fonksiyonunu uygulamak için kullanılabileceğini göstermektedir. ComputeDRAM [53] benzer bir işlemin var olan DRAM aygıtlarında zamanlama parametrelerini ihlal ederek gerçekleştirilebileceğini göstermektedir. Bu çalışmaların sunduğu tabanın üzerine inşa ederek dörtlü etkinleştirme (QUAC) yöntemini öne sürmekteyiz. QUAC temel olarak farklı bir davranışa yol açmakta ve DRAM algılama yükselteçlerinde rastgele sayı oluşturmaktadır. Bu davranıştan faydalanarak yüksek hızda ve düşük gecikme ile rastgele sayı üretmekteyiz.
CROW [108] ve MCR-DRAM [168] aynı veriyi içeren birden çok satırın aynı anda açılması ile erişim gecikmesini azaltan yeni bir DRAM devresi tasarlamaktadır. **RowClone [87]** iki DRAM satırının art arda açılması ile veri kopyalanmasını sağlamaktadır. Bu mekanizmalar *(i)* DRAM yongalarına değişiklik gerektirmekte ve *(ii)* gerçek rastgele sayı üretmemektedir.

11. SONUÇ

Var olan sistemlerde düşük maliyetle gerçeklenebilen, yüksek hızlı ve düşük gecikmeli DRAM GRSÜ QUAC-TRNG'yi öne sürmekteyiz. QUAC-TRNG'nin ana fikri, yeni gözlemlediğimiz bir davranış olan ve aynı anda dört DRAM satırının açılmasına sebep olan QUAC'ı kullanarak birden çok DRAM algılama yükselteçlerinde paralel bir şekilde yarı kararlı durum oluşturmaktır. QUAC-TRNG'nin ürettiği rastgele bit dizilerinin tüm 15 NIST STS testini geçtiğini ve yüksek nitelikli gerçek rastgele sayıları 3,44 Gb/s hızla ürettiğini, 136 gerçek DRAM yongasını kullanarak gerçekleştirdiğimiz deneyler sonucu göstermekteyiz. QUAC-TRNG'yi temel (önerildiği gibi) ve geliştirilmiş (hızlandırılmış) yapılandırmaları altında incelediğimiz iki geçmiş çalışma ile karşılaştırmaktayız. QUAC-TRNG, son teknoloji DRAM GRSÜlerine kıyasla sırasıyla temel ve geliştirilmiş yapılandırmaları için 15,08 kat ve 1,41 kat yüksek başarım göstermektedir. QUAC-TRNG DRAM bant genişliği ile iyi ölçeklenmektedir ve gelecek DRAM aktarım hızlarında geçmiş çalışmaların geliştirilmiş yapılandırmalarından 2,03 kat yüksek başarım göstermektedir. Sonuç olarak QUAC-TRNG güvenilir bir şekilde yüksek hız ve düşük gecikme ile gerçek rastgele sayı üretmektedir.

Kaynaklar

[1] Darrell Whitley. A Genetic Algorithm Tutorial. *Statistics and Computing*, 1998.

[2] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion. *JMLR*, 2010.

[3] W. F. Schmidt, M. A. Kraaijveld, and R. P. W. Duin. Feedforward Neural Networks with Random Weights. In *ICPR*, 1992.

[4] Le Zhang and P.N. Suganthan. A Survey of Randomized Algorithms for Training Neural Networks. *Information Sciences*, 2016.

[5] Y. Miché, B. Schrauwen, and A. Lendasse. Machine Learning Techniques based on Random Projections. In *ESANN*, 2010.

[6] Vittorio Bagini and Marco Bucci. A Design of Reliable True Random Number Generator for Cryptographic Applications. In *CHES*, 1999.

[7] Mohammed Bakiri, Christophe Guyeux, Jean-François Couchot, and Abdelkrim Kamel Oudjida. Survey on Hardware Implementation of Random Number Generators on FPGA: Theory and Experimental Analyses. *CSR*, 2018.

[8] Andrea Röck. Pseudorandom Number Generators for Cryptographic Applications. Master's thesis, Paris-Lodron-Universität Salzburg, 2005.

[9] Xiongfeng Ma, Xiao Yuan, Zhu Cao, Bing Qi, and Zhen Zhang. Quantum Random Number Generation. *Quantum Inf.*, 2016.

[10] Mario Stipčević and Çetin Kaya Koç. True Random Number Generators. In *Open Problems in Mathematics and Computational Science*. 2014.

[11] Mahmood Barangi, Joseph S Chang, and Pinaki Mazumder. Straintronics-Based True Random Number Generator for High-Speed and Energy-Limited Applications. In *IEEE Trans. Magn*, 2016.

[12] Sha Tao and Elena Dubrova. TVL-TRNG: Sub-Microwatt True Random Number Generator Exploiting Metastability in Ternary Valued Latches. In *ISMVL*, 2017.

[13] Zvi Gutterman, Benny Pinkas, and Tzachy Reinman. Analysis of the Linux Random Number Generator. In *SP*, 2006.

[14] Vincent von Kaenel and Toshinari Takayanagi. Dual True Random Number Generators for Cryptographic Applications Embedded on a 200 Million Device Dual CPU SOC. In *CICC*, 2007.

[15] J. Kim, H. Nili, N. D. Truong, T. Ahmed, J. Yang, D. S. Jeong, S. Sriram, D. C. Ranasinghe, S. Ippolito, H. Chun, and O. Kavehei. Nano-Intrinsic True Random Number Generation: A Device to Data Study. *IEEE TCAS*, 2019.

[16] Milos Drutarovsky and Pavol Galajda. A Robust Chaos-based True Random Number Generator Embedded in Reconfigurable Switched-Capacitor Hardware. In *Radioelektronika*, 2007.

[17] Sammy HM Kwok and Edmund Y Lam. FPGA-based High-speed True Random Number Generator for Cryptographic Applications. In *TENCON*, 2006.

[18] Abdelkarim Cherkaoui, Viktor Fischer, Laurent Fesquet, and Alain Aubert. A Very High Speed True Random Number Generator with Entropy Assessment. In *CHES*, 2013.

[19] Teng Zhang, Minghui Yin, Changmin Xu, Xiayan Lu, Xinhao Sun, Yuchao Yang, and Ru Huang. High-speed True Random Number Generation Based on Paired Memristors for Security Electronics. *Nanotechnology*, 2017.

[20] Quintessence Labs. Random Number Generators White Paper, 2015.

[21] Kaiyuan Yang, David Blaauw, and Dennis Sylvester. An All-digital Edge Racing True Random Number Generator Robust Against PVT Variations. In *JSSC*, 2016.

[22] J. S. Kim, M. Patel, H. Hassan, L. Orosa, and O. Mutlu. D-RaNGe: Using Commodity DRAM Devices to Generate True Random Numbers with Low Latency and High Throughput. In *HPCA*, 2019.

[23] Soubhagya Sutar, Arnab Raha, and Vijay Raghunathan. D-PUF: An Intrinsically Reconfigurable DRAM PUF for Device Authentication in Embedded Systems. In *CASES*, 2016.

[24] B. M. S. Bahar Talukder, J. Kerns, B. Ray, T. Morris, and M. T. Rahman. Exploiting DRAM Latency Variations for Generating True Random Numbers. In *ICCE*, 2019.

[25] Christoph Keller, Frank Gurkaynak, Hubert Kaeslin, and Norbert Felber. Dynamic Memory-based Physically Unclonable Function for the Generation of Unique Identifiers and True Random Numbers. In *ISCAS*, 2014.

[26] Changwoo Pyo, Sungil Pae, and Gyungho Lee. DRAM as Source of Randomness. In *IET*, 2009.

[27] Daniel E Holcomb, Wayne P Burleson, and Kevin Fu. Initial SRAM State as a Fingerprint and Source of True Random Numbers for RFID Tags. In *RFID*, 2007.

[28] Daniel E. Holcomb, Wayne P. Burleson, and Kevin Fu. Power-Up SRAM State as an Identifying Fingerprint and Source of True Random Numbers. *ToC*, 2009.

[29] Vincent van der Leest, Erik van der Sluis, Geert-Jan Schrijen, Pim Tuyls, and Helena Handschuh. Efficient Implementation of True Random Number Generator Based on SRAM PUFs. In *Cryptography and Security: From Theory to Applications*. 2012.

[30] UPMEM. Introduction to UPMEM PIM. Processing-in-memory (PIM) on DRAM accelerator (White Paper), 2018.

[31] Fatemeh Tehranipoor, Wei Yan, and John A Chandy. Robust Hardware True Random Number Generators using DRAM Remanence Effects. In *HOST*, 2016.

[32] Soubhagya Sutar, Arnab Raha, Devadatta Kulkarni, Rajeev Shorey, Jeffrey Tew, and Vijay Raghunathan. D-PUF: An Intrinsically Reconfigurable DRAM PUF for Device Authentication and Random Number Generation. In *TECS*, 2018.

[33] Maryam S Hashemian, Bhanu Singh, Francis Wolff, Daniel Weyer, Steve Clay, and Christos Papachristou. A Robust Authentication Methodology Using Physically Unclonable Functions in DRAM Arrays. In *DATE*, 2015.

[34] Charles Eckert, Fatemeh Tehranipoor, and John A Chandy. DRNG: DRAM-based Random Number Generation Using its Startup Value Behavior. In *MWSCAS*, 2017.

[35] Khaled Humood, Baker Mohammad, and Heba Abunahla. DTRNG: Low Cost and Robust True Random Number Generator Using DRAM Weak Write Scheme. In *ISCAS*, 2021.

[36] Lawrence Bassham, Andrew Rukhin, Juan Soto, James Nechvatal, Miles Smid, Stefan Leigh, M Levenson, M Vangel, Nathanael Heckert, and D Banks. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. Special Publication (NIST SP), 2010.

[37] JEDEC. DDR4. *JEDEC Standard JESD79–4*, 2012.

[38] JEDEC. Graphics Double Data Rate (GDDR5) SGRAM Standard. 2016.

[39] Yoongu Kim, Vivek Seshadri, Donghyuk Lee, Jamie Liu, and Onur Mutlu. A Case for Exploiting Subarray-level Parallelism (SALP) in DRAM. In *ISCA*, 2012.

[40] Kevin K Chang, Donghyuk Lee, Zeshan Chishti, Alaa R Alameldeen, Chris Wilkerson, Yoongu Kim, and Onur Mutlu. Improving DRAM Performance by Parallelizing Refreshes with Accesses. In *HPCA*, 2014.

[41] Vivek Seshadri and Onur Mutlu. In-DRAM Bulk Bitwise Execution Engine. arXiv:1905.09822, 2020.

[42] Donghyuk Lee, Yoongu Kim, Gennady Pekhimenko, Samira Khan, Vivek Seshadri, Kevin Chang, and Onur Mutlu. Adaptive-Latency DRAM: Optimizing DRAM Timing for the Common-Case. In *HPCA*, 2015.

[43] Kevin K Chang, Abhijith Kashyap, Hasan Hassan, Saugata Ghose, Kevin Hsieh, Donghyuk Lee, Tianshi Li, Gennady Pekhimenko, Samira Khan, and Onur Mutlu. Understanding Latency Variation in Modern DRAM Chips: Experimental Characterization, Analysis, and Optimization. In *SIGMETRICS*, 2016.

[44] Kevin K Chang, Abdullah Yaglikci, Saugata Ghose, Adity Agrawal, Niladrish Chatterjee, Abhijith Kashyap, Donghyuk Lee, Mike O'Connor, Hasan Hassan, and Onur Mutlu. Understanding Reduced-Voltage Operation in Modern DRAM Devices: Experimental Characterization, Analysis, and Mechanisms. In *SIGMETRICS*, 2017.

[45] Donghyuk Lee, Samira Khan, Lavanya Subramanian, Saugata Ghose, Rachata Ausavarungnirun, Gennady Pekhimenko, Vivek Seshadri, and Onur Mutlu. Design-Induced Latency Variation in Modern DRAM Chips: Characterization, Analysis, and Latency Reduction Mechanisms. In *SIGMETRICS*, 2017.

[46] Jeremie S Kim, Minesh Patel, Hasan Hassan, and Onur Mutlu. Solar-DRAM: Reducing DRAM Access Latency by Exploiting the Variation in Local Bitlines. In *ICCD*, 2018.

[47] Karthik Chandrasekar, Sven Goossens, Christian Weis, Martijn Koedam, Benny Akesson, Norbert Wehn, and Kees Goossens. Exploiting Expendable Process-Margins in DRAMs for Run-Time Performance Optimization. In *DATE*, 2014.

[48] Donghyuk Lee. *Reducing DRAM Latency at Low Cost by Exploiting Heterogeneity*. PhD Dissertation, Carnegie Mellon University, 2016.

[49] Donghyuk Lee. *Reducing DRAM Latency at Low Cost by Exploiting Heterogeneity*. PhD Dissertation, Carnegie Mellon University, 2016.

[50] Kevin K. Chang. *Understanding and Improving Latency of DRAM-Based Memory Systems*. PhD Dissertation, Carnegie Mellon University, 2017.

[51] J. S. Kim. *Improving DRAM Performance, Security, and Reliability by Understanding and Exploiting DRAM Timing Parameter Margins*. PhD Dissertation, Carnegie Mellon University, 2020.

[52] J. S. Kim, M. Patel, H. Hassan, and O. Mutlu. The DRAM Latency PUF: Quickly Evaluating Physical Unclonable Functions by Exploiting the Latency-Reliability Tradeoff in Modern Commodity DRAM Devices. In *HPCA*, 2018.

[53] Fei Gao, Georgios Tziantzioulis, and David Wentzlaff. ComputeDRAM: In-Memory Compute Using Off-the-Shelf DRAMs. In *MICRO*, 2019.

[54] Çetin Kaya Koç. About Cryptographic Engineering. In *Cryptographic Engineering*. 2009.

[55] Ronald Rivest. The MD5 Message-Digest Algorithm. In *RFC*, 1992.

[56] Patrick J Clarke, Robert J Collins, Philip A Hiskett, Paul D Townsend, and Gerald S Buller. Robust Gigahertz Fiber Quantum Key Distribution. *Applied Physics Letters*, 2011.

[57] XiaoMing Lu, LiJun Zhang, YongGang Wang, Wei Chen, DaJun Huang, Deng Li, Shuang Wang, DeYong He, ZhenQiang Yin, Yu Zhou, et al. FPGA Based Digital Phase-coding Quantum Key Distribution System. *Science China Physics, Mechanics & Astronomy*, 2015.

[58] Thomas E Hull and Alan R Dobell. Random Number Generators. *SIAM Review*, 1962.

[59] R.C. Botha. *The Development of a Hardware Random Number Generator for Gamma-ray Astronomy*. PhD Dissertation, North-West University, 2005.

[60] P. Davis and P. Rabinowitz. Some Monte Carlo Experiments in Computing Multiple Integrals. *Mathematical Tables and Other Aids to Computation*, 1956.

[61] Zhang Limeng, Baowu Pan, Guangcan Chen, Lu Guo, Dan Lu, and Lingjuan Zhao. 640-Gbit/s Fast Physical Random Number Generation Using a Broadband Chaotic Semiconductor Laser. *Scientific Reports*, 2017.

[62] Tobias Gehring, Cosmo Lupo, Arne Kordts, Dino Solar Nikolic, Nitin Jain, Tobias Rydberg, Thomas B. Pedersen, Stefano Pirandola, and Ulrik L. Andersen. Ultra-Fast Real-Time Quantum Random Number Generator with Correlated Measurement Outcomes and Rigorous Security Certification. arXiv:1812.05377, 2020.

[63] Kazusa Ugajin, Yuta Terashima, Kento Iwakawa, Atsushi Uchida, Takahisa Harayama, Kazuyuki Yoshimura, and Masanobu Inubushi. Real-time fast physical random number generator with a photonic integrated circuit. *Optics Express*, 2017.

[64] André Stefanov, Nicolas Gisin, Olivier Guinnard, Laurent Guinnard, and Hugo Zbinden. Optical Quantum Random Number Generator. In *J. Mod. Opt*, 2000.

[65] X. Wang, F. Liang, P. Miao, Y. Qian, and G. Jin. 10-Gbps True Random Number Generator Accomplished in ASIC. In *RT*, 2016.

[66] Takehiko Amaki, Masanori Hashimoto, and Takao Onoye. An Oscillator-based True Random Number Generator with Process and Temperature Tolerance. In *DAC*, 2015.

[67] Marco Bucci, Lucia Germani, Raimondo Luzzi, Alessandro Trifiletti, and Mario Varanonuovo. A High-speed Oscillator-based Truly Random Number Source for Cryptographic Applications on a Smart Card IC. In *TC*, 2003.

[68] Fabio Pareschi, Gianluca Setti, and Riccardo Rovatti. A Fast Chaos-based True Random Number Generator for Cryptographic Applications. In *ESSCIRC*, 2006.

[69] M. Degaldo-Restituto, F. Medeiro, and A. Rodriguez-Vazquez. Nonlinear switched-current CMOS IC for random signal generation. *Electronics Letters*, 1993.

[70] Vivek Seshadri, Donghyuk Lee, Thomas Mullins, Hasan Hassan, Amirali Boroumand, Jeremie Kim, Michael A Kozuch, Onur Mutlu, Phillip B Gibbons, and Todd C Mowry. Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology. In *MICRO*, 2017.

[71] Saugata Ghose, Kevin Hsieh, Amirali Boroumand, Rachata Ausavarungnirun, and Onur Mutlu. Enabling the Adoption of Processing-in-Memory: Challenges, Mechanisms, Future Research Directions. *arXiv:1802.00320*, 2018.

[72] Vivek Seshadri and Onur Mutlu. Simple Operations in Memory to Reduce Data Movement. In *Advances in Computers*. 2017.

[73] Geraldo Francisco Oliveira, Juan Gómez-Luna, Saugata Ghose, Lois Orosa, Nandita Vijaykumar, Ivan Fernandez, Mohammad Sadrosadati, and Onur Mutlu. DAMOV: A New Methodology and Benchmark Suite for Evaluating Data Movement Bottlenecks. arXiv:2105.03725, 2021.

[74] Juan Gómez-Luna, Izzat El Hajj, Ivan Fernández, Christina Giannoula, Geraldo F. Oliveira, and Onur Mutlu. Benchmarking a New Paradigm: Understanding a Modern Processing-in-Memory Architecture. arXiv:2105.03814, 2021.

[75] Amirali Boroumand, Saugata Ghose, Geraldo F. Oliveira, and Onur Mutlu. Polynesia: Enabling effective hybrid transactional/analytical databases with specialized hardware/software co-design. arXiv:2103.00798, 2021.

[76] Amirali Boroumand, Saugata Ghose, Minesh Patel, Hasan Hassan, Brandon Lucia, Rachata Ausavarungnirun, Kevin Hsieh, Nastaran Hajinazar, Krishna T Malladi, Hongzhong Zheng, et al. CONDA: Efficient Cache Coherence Support for Near-Data Accelerators. In *ISCA*, 2019.

[77] Onur Mutlu, Saugata Ghose, Juan Gómez-Luna, and Rachata Ausavarungnirun. A Modern Primer on Processing in Memory. arXiv:2012.03112, 2020.

[78] Saugata Ghose, Amirali Boroumand, Jeremie S Kim, Juan Gómez-Luna, and Onur Mutlu. Processing-in-Memory: A Workload-Driven Perspective. *IBM JRD*, 2019.

[79] Yaohua Wang, Lois Orosa, Xiangjun Peng, Yang Guo, Saugata Ghose, Minesh Patel, Jeremie S. Kim, Juan Gómez Luna, Mohammad Sadrosadati, Nika Mansouri Ghiasi, and Onur Mutlu. FIGARO: Improving System Performance via Fine-Grained In-DRAM Data Relocation and Caching. In *MICRO*, 2020.

[80] Christina Giannoula, Nandita Vijaykumar, Nikela Papadopoulou, Vasileios Karakostas, Ivan Fernandez, Juan Gómez-Luna, Lois Orosa, Nectarios Koziris, Georgios Goumas, and Onur Mutlu. SynCron: Efficient Synchronization Support for Near-Data-Processing Architectures. In *HPCA*, 2021.

[81] Berkin Akin, Franz Franchetti, and James C. Hoe. Data Reorganization in Memory Using 3D-Stacked DRAM. In *ISCA*, 2015.

[82] Shaizeen Aga, Supreet Jeloka, Arun Subramaniyan, Satish Narayanasamy, David Blaauw, and Reetuparna Das. Compute Caches. In *HPCA*, 2017.

[83] Junwhan Ahn, Sungpack Hong, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi. A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing. In *ISCA*, 2015.

[84] Junwhan Ahn, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi. PIM-Enabled Instructions: a Low-overhead, Locality-aware Processing-in-Memory Architecture. In *ISCA*, 2015.

[85] Donghyuk Lee, Saugata Ghose, Gennady Pekhimenko, Samira Khan, and Onur Mutlu. Simultaneous Multi-Layer Access: Improving 3D-Stacked Memory Bandwidth at Low Cost. In *TACO*, 2016.

[86] Vivek Seshadri, Kevin Hsieh, Amirali Boroum, Donghyuk Lee, Michael A. Kozuch, Onur Mutlu, Phillip B. Gibbons, and Todd C. Mowry. Fast Bulk Bitwise AND and OR in DRAM. *IEEE CAL*, 2015.

[87] Vivek Seshadri, Yoongu Kim, Chris Fallin, Donghyuk Lee, Rachata Ausavarungnirun, Gennady Pekhimenko, Yixin Luo, Onur Mutlu, Phillip B Gibbons, Michael A Kozuch, and Todd Mowry. RowClone: Fast and Energy-Efficient In-DRAM Bulk Data Copy and Initialization. In *MICRO*, 2013.

[88] Vivek Seshadri, Thomas Mullins, Amirali Boroumand, Onur Mutlu, Phillip B Gibbons, Michael A Kozuch, and Todd C Mowry. Gather-Scatter DRAM: In-DRAM Address Translation to Improve the Spatial Locality of Non-Unit Strided Accesses. In *MICRO*, 2015.

[89] Zhiyu Liu, Irina Calciu, Maurice Herlihy, and Onur Mutlu. Concurrent Data Structures for Near-Memory Computing. In *SPAA*, 2017.

[90] Ashutosh Pattnaik, Xulong Tang, Adwait Jog, Onur Kayiran, Asit K Mishra, Mahmut T Kandemir, Onur Mutlu, and Chita R Das. Scheduling Techniques for GPU Architectures with Processing-in-Memory Capabilities. In *PACT*, 2016.

[91] Oreoluwatomiwa O Babarinsa and Stratos Idreos. JAFAR: Near-Data Processing for Databases. In *SIGMOD*, 2015.

[92] Amin Farmahini-Farahani, Jung Ho Ahn, Katherine Morrow, and Nam Sung Kim. NDA: Near-DRAM Acceleration Architecture Leveraging Commodity DRAM Devices and Standard Memory Modules. In *HPCA*, 2015.

[93] Mingyu Gao, Grant Ayers, and Christos Kozyrakis. Practical Near-Data Processing for In-Memory Analytics Frameworks. In *PACT*, 2015.

[94] Mingyu Gao and Christos Kozyrakis. HRL: Efficient and Flexible Reconfigurable Logic for Near-Data Processing. In *HPCA*, 2016.

[95] Syed Minhaj Hassan, Sudhakar Yalamanchili, and Saibal Mukhopadhyay. Near Data Processing: Impact and Optimization of 3D Memory System Architecture on the Uncore. In *MEMSYS*, 2015.

[96] Kevin Hsieh, Eiman Ebrahimi, Gwangsun Kim, Niladrish Chatterjee, Mike O'Connor, Nandita Vijaykumar, Onur Mutlu, and Stephen W Keckler. Transparent Offloading and Mapping (TOM): Enabling Programmer-Transparent Near-Data Processing in GPU Systems. In *ISCA*, 2016.

[97] Amir Morad, Leonid Yavits, and Ran Ginosar. GP-SIMD Processing-in-Memory. In *TACO*, 2015.

[98] Zehra Sura, Arpith Jacob, Tong Chen, Bryan Rosenburg, Olivier Sallenave, Carlo Bertolli, Samuel Antao, Jose Brunheroto, Yoonho Park, Kevin O'Brien, et al. Data Access Optimization in a Processing-in-Memory System. In *CF*, 2015.

[99] Dongping Zhang, Nuwan Jayasena, Alexander Lyashevsky, Joseph L Greathouse, Lifan Xu, and Michael Ignatowski. TOP-PIM: Throughput-Oriented Programmable Processing in Memory. In *HPDC*, 2014.

[100] Kevin Hsieh, Samira Khan, Nandita Vijaykumar, Kevin K Chang, Amirali Boroumand, Saugata Ghose, and Onur Mutlu. Accelerating Pointer Chasing in 3D-Stacked Memory: Challenges, Mechanisms, Evaluation. In *ICCD*, 2016.

[101] Amirali Boroumand, Saugata Ghose, Brandon Lucia, Kevin Hsieh, Krishna Malladi, Hongzhong Zheng, and Onur Mutlu. LazyPIM: An Efficient Cache Coherence Mechanism for Processing-in-Memory. In *CAL*, 2017.

[102] Kevin K Chang, Prashant J Nair, Donghyuk Lee, Saugata Ghose, Moinuddin K Qureshi, and Onur Mutlu. Low-Cost Inter-Linked Subarrays (LISA): Enabling Fast Inter-Subarray Data Movement in DRAM. In *HPCA*, 2016.

[103] Jeremie S Kim, Damla Senol Cali, Hongyi Xin, Donghyuk Lee, Saugata Ghose, Mohammed Alser, Hasan Hassan, Oguz Ergin, Can Alkan, and Onur Mutlu. GRIM-Filter: Fast Seed Location Filtering in DNA Read Mapping Using Processing-in-memory Technologies. *BMC Genomics*, 2018.

[104] Amirali Boroumand, Saugata Ghose, Berkin Akin, Ravi Narayanaswami, Geraldo F. Oliveira, Xiaoyu Ma, Eric Shiu, and Onur Mutlu. Mitigating Edge Machine Learning Inference Bottlenecks: An Empirical Study on Accelerating Google Edge Models. arXiv:2103.00768, 2021.

[105] Amirali Boroumand, Saugata Ghose, Youngsok Kim, Rachata Ausavarungnirun, Eric Shiu, Rahul Thakur, Daehyun Kim, Aki Kuusela, Allan Knies, Parthasarathy Ranganathan, and Onur Mutlu. Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks. In *ASPLOS*, 2018.

[106] Vivek Seshadri. *Simple DRAM and Virtual Memory Abstractions to Enable Highly Efficient Memory Systems*. PhD Dissertation, Carnegie Mellon University, 2016.

[107] Onur Mutlu, Saugata Ghose, Juan Gómez-Luna, and Rachata Ausavarungnirun. Processing Data Where it Makes Sense: Enabling In-Memory Computation. *Microprocessors and Microsystems*, 2019.

[108] Hasan Hassan, Minesh Patel, Jeremie S. Kim, A. Giray Yaglikci, Nandita Vijaykumar, Nika Mansouri Ghiasi, Saugata Ghose, and Onur Mutlu. CROW: A Low-Cost Substrate for Improving DRAM Performance, Energy Efficiency, and Reliability. In *ISCA*, 2019.

[109] G. Singh, D. Diamantopoulos, C. Hagleitner, J. Gomez-Luna, S. Stuijk, O. Mutlu, and H. Corporaal. NERO: A Near High-Bandwidth Memory Stencil Accelerator for Weather Prediction Modeling. In *FPL*, 2020.

[110] Ivan Fernandez, Ricardo Quislant, Christina Giannoula, Mohammed Alser, Juan Gómez-Luna, Eladio Gutiérrez, Oscar Plata, and Onur Mutlu. NATSA: A Near-Data Processing Accelerator for Time Series Analysis, 2020.

[111] Young-Cheon Kwon, Suk Han Lee, Jaehoon Lee, Sang-Hyuk Kwon, Je Min Ryu, Jong-Pil Son, O Seongil, Hak-Soo Yu, Haesuk Lee, Soo Young Kim, et al. 25.4 A 20nm 6GB Function-In-Memory DRAM, Based on HBM2 with a 1.2 TFLOPS Programmable Computing Unit Using Bank-Level Parallelism, for Machine Learning Applications. In *ISSCC*, 2021.

[112] Fabrice Devaux. The True Processing in Memory Accelerator. In *HCS*, 2019.

[113] Shuangchen Li, Cong Xu, Qiaosha Zou, Jishen Zhao, Yu Lu, and Yuan Xie. Pinatubo: A Processing-in-Memory Architecture for Bulk Bitwise Operations in Emerging Non-Volatile Memories. In *DAC*, 2016.

[114] Ping Chi, Shuangchen Li, Cong Xu, Tao Zhang, Jishen Zhao, Yongpan Liu, Yu Wang, and Yuan Xie. PRIME: A Novel Processing-in-Memory Architecture for Neural Network Computation in ReRAM-Based Main Memory. In *ISCA*, 2016.

[115] Lois Orosa, Yaohua Wang, Ivan Puddu, Mohammad Sadrosadati, Kaveh Razavi, Juan Gómez-Luna, Hasan Hassan, Nika Mansouri-Ghiasi, Arash Tavakkol, Minesh Patel, Jeremie Kim, Vivek Seshadri, Uksong Kang, Saugata Ghose, Rodolfo Azevedo, and Onur Mutlu. Dataplant: Enhancing system security with low-cost in-dram value generation primitives. arXiv:1902.07344, 2019.

[116] Lois Orosa, Yaohua Wang, Ivan Puddu, Mohammad Sadrosadati, Kaveh Razavi, Juan Gómez-Luna, Hasan Hassan, Nika Mansouri-Ghiasi, Arash Tavakkol, Minesh Patel, Jeremie Kim, Vivek Seshadri, Uksong Kang, Saugata Ghose, Rodolfo Azevedo, and Onur Mutlu. CODIC: A Low-cost Substrate for Enabling Custom In-DRAM Functionalities and Optimizations. In *ISCA*, 2021.

[117] Samira Khan, Donghyuk Lee, Yoongu Kim, Alaa R Alameldeen, Chris Wilkerson, and Onur Mutlu. The Efficacy of Error Mitigation Techniques for DRAM Retention Failures: A Comparative Experimental Study. In *SIGMETRICS*, 2014.

[118] Ravi K Venkatesan, Stephen Herr, and Eric Rotenberg. Retention-aware Placement in DRAM (RAPID): Software Methods for Quasi-non-volatile DRAM. In *HPCA*, 2006.

[119] Minesh Patel, Jeremie S Kim, and Onur Mutlu. The Reach Profiler (REAPER): Enabling the Mitigation of DRAM Retention Failures via Profiling at Aggressive Conditions. In *ISCA*, 2017.

[120] Jamie Liu, Ben Jaiyen, Richard Veras, and Onur Mutlu. RAIDR: Retention-Aware Intelligent DRAM Refresh. In *ISCA*, 2012.

[121] Moinuddin K Qureshi, DaeHyun Kim, Samira Khan, Prashant J Nair, and Onur Mutlu. AVATAR: A Variable-Retention-Time (VRT) Aware Refresh for DRAM Systems. In *DSN*, 2015.

[122] Onur Mutlu. Memory Scaling: A Systems Architecture Perspective. In *IMW*, 2013.

[123] N. Chatterjee, M. OConnor, D. Lee, D. R. Johnson, S. W. Keckler, M. Rhu, and W. J. Dally. Architecting an Energy-Efficient DRAM System for GPUs. In *HPCA*, 2017.

[124] Aniruddha N. Udipi, Naveen Muralimanohar, Niladrish Chatterjee, Rajeev Balasubramonian, Al Davis, and Norman P. Jouppi. Rethinking DRAM Design and Organization for Energy-Constrained Multi-Cores. In *ISCA*, 2010.

[125] DRAM Power Model. `https://www.rambus.com/energy/`, .

[126] Mudit Bhargava, Kaship Sheikh, and Ken Mai. Robust True Random Number Generator Using Hot-Carrier Injection Balanced Metastable Sense Amplifiers. In *HOST*, 2015.

[127] FIPS, PUB. 180-2: Secure hash standard (SHS). *US Department of Commerce, National Institute of Standards and Technology (NIST)*, 2012.

[128] Jeremie S. Kim, Minesh Patel, A. Giray Yağlıkçı, Hasan Hassan, Roknoddin Azizi, Lois Orosa, and Onur Mutlu. Revisiting RowHammer: An Experimental Analysis of Modern DRAM Devices and Mitigation Techniques. In *ISCA*, 2020.

[129] Pietro Frigo, Emanuele Vannacci, Hasan Hassan, Victor van der Veen, Onur Mutlu, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. TRRespass: Exploiting the Many Sides of Target Row Refresh. In *S&P*, 2020.

[130] H. Hassan, N. Vijaykumar, S. Khan, S. Ghose, K. Chang, G. Pekhimenko, D. Lee, O. Ergin, and O. Mutlu. SoftMC: A Flexible and Practical Open-Source Infrastructure for Enabling Experimental DRAM Studies. In *HPCA*, 2017.

[131] Claude Elwood Shannon. A Mathematical Theory of Communication. *Bell System Technical Journal*, 1948.

[132] V. Mehrotra. *Modeling the Effects of Systematic Process Variation of Circuit Performance*. PhD Dissertation, Massachusetts Institute of Technology, 2001.

[133] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors. In *ISCA*, 2014.

[134] Robert T Smith, James D Chlipala, JOHN FM Bindels, Roy G Nelson, Frederick H Fischer, and Thomas F Mantz. Laser Programmable Redundancy and Yield Improvement in a 64K DRAM. *JSSC*, 1981.

[135] Masashi Horiguchi. Redundancy Techniques for High-Density DRAMs. In *ISIS*, 1997.

[136] B. Keeth and R.J. Baker. *DRAM Circuit Design: A Tutorial*. Wiley, 2001.

[137] Kiyoo Itoh. *VLSI Memory Chip Design*. Springer, 2001.

[138] Jamie Liu, Ben Jaiyen, Yoongu Kim, Chris Wilkerson, and Onur Mutlu. An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms. In *ISCA*, 2013.

[139] Uksong Kang, Hak-soo Yu, Churoo Park, Hongzhong Zheng, John Halbert, Kuljit Bains, S Jang, and Joo Sun Choi. Co-Architecting Controllers and DRAM to Enhance DRAM Process Scaling. In *The Memory Forum*, 2014.

[140] Samira Khan, Donghyuk Lee, and Onur Mutlu. PARBOR: An Efficient System-Level Technique to Detect Data-Dependent Failures in DRAM. In *DSN*, 2016.

[141] Samira Khan, Chris Wilkerson, Zhe Wang, Alaa R Alameldeen, Donghyuk Lee, and Onur Mutlu. Detecting and Mitigating Data-Dependent DRAM Failures by Exploiting Current Memory Content. In *MICRO*, 2017.

[142] Andrei Tatar, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. Defeating Software Mitigations Against Rowhammer: A Surgical Precision Hammer. In *RAID*, 2018.

[143] Alessandro Barenghi, Luca Breveglieri, Niccolò Izzo, and Gerardo Pelosi. Software-Only Reverse Engineering of Physical DRAM Mappings for Rowhammer Attacks. In *IVSW*, 2018.

[144] Lucian Cojocar, Jeremie Kim, Minesh Patel, Lillian Tsai, Stefan Saroiu, Alec Wolman, and Onur Mutlu. Are We Susceptible to Rowhammer? An End-to-End Methodology for Cloud Providers. In *S&P*, 2020.

[145] Minesh Patel, Jeremie Kim, Taha Shahroodi, Hasan Hassan, and Onur Mutlu. Bit-Exact ECC Recovery (BEER): Determining DRAM On-Die ECC Functions by Exploiting DRAM Data Retention Characteristics. In *MICRO*, 2020.

[146] John von Neumann. Various Techniques Used in Connection with Random Digits. In *Monte Carlo Method*, NBS Applied Mathematics Series. 1951.

[147] Yoongu Kim, Weikun Yang, and Onur Mutlu. Ramulator: A Fast and Extensible DRAM Simulator. In *CAL*, 2016.

[148] Ramulator Source Code. `https://github.com/CMU-SAFARI/ramulator`, .

[149] AMD. AMD Random Number Generator. `https://www.amd.com/system/files/TechDocs/amd-random-number-generator.pdf`, .

[150] Luca Baldanzi, Luca Crocetti, Francesco Falaschi, Matteo Bertolucci, Jacopo Belli, and Luca Fanucci. Cryptographically Secure Pseudo-Random Number Generator IP-Core Based on SHA2 Algorithm. *Sensors*, 2020.

[151] A. Satoh and T. Inoue. ASIC Hardware Focused Comparison for Hash Functions MD5, RIPEMD-160, and SHS. In *ITCC*, 2005.

[152] Fast Hashing Cores. `https://www.heliontech.com/fast_hash.htm`.

[153] CACTI: An integrated cache and memory access time, cycle time, area, leakage, and dynamic power model. https://www.hpl.hp.com/research/cacti/.

[154] AMD. AMD Zen2 Microarchitecture. https://en.wikichip.org/wiki/amd/microarchitectures/zen_2, .

[155] David Suggs, Mahesh Subramony, and Dan Bouvier. The AMD "Zen 2" Processor. *Hot Chips*, 2020.

[156] SK Hynix. DDR4 SDRAM Device Operation.

[157] Craig S Petrie and J Alvin Connelly. A Noise-based IC Random Number Generator for Applications in Cryptography. In *Trans. Circuits Syst. I*, 2000.

[158] Sanu K Mathew, Suresh Srinivasan, Mark A Anders, Himanshu Kaul, Steven K Hsu, Farhana Sheikh, Amit Agarwal, Sudhir Satpathy, and Ram K Krishnamurthy. 2.4 Gbps, 7 mW All-digital PVT-variation Tolerant True Random Number Generator for 45 nm CMOS High-performance Microprocessors. In *JSSC*, 2012.

[159] Ralf Brederlow, Ramesh Prakash, Christian Paulus, and Roland Thewes. A Low-power True Random Number Generator using Random Telegraph Noise of Single Oxide-traps. In *ISSCC*, 2006.

[160] Carlos Tokunaga, David Blaauw, and Trevor Mudge. True Random Number Generator with a Metastability-based Quality Control. In *JSSC*, 2008.

[161] DJ Kinniment and EG Chester. Design of an On-chip Random Number Generator using Metastability. In *ESSCIRC*, 2002.

[162] Jeremy Holleman, Seth Bridges, Brian P Otis, and Chris Diorio. A 3mu W CMOS True Random Number Generator with Adaptive Floating-Gate Offset Cancellation. *JSSC*, 2008.

[163] ARM. ARM True Random Number Generator (TRNG) Technical Reference Manual Revision r0p0. https://developer.arm.com/documentation/100976/0000/Introduction/Features.

[164] Benjamin Jun and Paul Kocher. The Intel Random Number Generator (White Paper). *Cryptography Research Inc.*, 1999.

[165] Liao Ning, Jiang Ding, Bai Chuang, and Zou Xuecheng. Design and Validation of High Speed True Random Number Generators Based on Prime-length Ring Oscillators. *The Journal of China Universities of Posts and Telecommunications*, 2015.

[166] Agner Fog. Lists of instruction latencies, throughputs and micro-operation breakdowns for Intel, AMD, and VIA CPUs. https://www.agner.org/optimize/instruction_tables.pdf.

[167] Vivek Seshadri, Donghyuk Lee, Thomas Mullins, Hasan Hassan, Amirali Boroumand, Jeremie Kim, Michael A Kozuch, Onur Mutlu, Phillip B Gibbons, and Todd C Mowry. Buddy-RAM: Improving the Performance and Efficiency of Bulk Bitwise Operations Using DRAM. arXiv:1611.09988, 2016.

[168] Jungwhan Choi, Wongyu Shin, Jaemin Jang, Jinwoong Suh, Yongkee Kwon, Youngsuk Moon, and Lee-Sup Kim. Multiple Clone Row DRAM: A Low Latency and Area Optimized DRAM. In *ISCA*, 2015.