TOBB EKONOMİ VE TEKNOLOJİ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

DÜŞÜK-GERİLİMLİ SRAM AYGITLARI İÇİN GERÇEK RASTGELE SAYI ÜRETME VE HATA MODELLEME YÖNTEMLERİ

YÜKSEK LİSANS TEZİ

İsmail Emir YÜKSEL

Bilgisayar Mühendisliği Anabilim Dalı

Tez Danışmanı: Prof. Dr. Oğuz ERGİN

TEMMUZ 2022

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.

İsmail Emir YÜKSEL

İMZA

ÖZET

Yüksek Lisans Tezi

DÜŞÜK-GERİLİMLİ SRAM AYGITLARI İÇİN GERÇEK RASTGELE SAYI ÜRETME VE HATA MODELLEME YÖNTEMLERİ

İsmail Emir YÜKSEL

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

Tez Danışmanı: Prof. Dr. Oğuz ERGİN

Tarih: TEMMUZ 2022

Tezin ilk çalışması olarak, SRAM'ler üzerinde deneysel düşük gerilim çalışmalarından çıkarılan gerçek hataları kullanarak ilk yaklaşık düşük gerilim hata modelini oluşturan bir altyapı olan MoRS'yi öneriyoruz. SRAM tabanlı bellekler, heterojen cihazlar, örneğin, GPU'lar, FPGA'lar, ASIC'ler dahil olmak üzere çeşitli bilgi işlem cihazları için yüksek performans elde etmek için çok önemli bileşenlerdir. Bu heterojen yapılar üzerinde çalışan Derin Sinir Ağları (DSA) gibi modern iş yükleri, verimli hızlandırma için büyük ölçüde yonga üzerindeki bellek mimarisine bağlıdır. Enerji tasarrufu sağlamanın yaygın yöntemlerinden biri, besleme voltajının nominal seviyenin altına düşürülmesidir. Bu tür sistemler, belirli bir voltaj sınırına kadar arızaya neden olmadan güvenli bir şekilde voltajı düşürülebilir. Bu güvenli aralığa voltaj koruma bandı da denir. Bununla birlikte, sistemin frekansı düşürmeden koruma bandı seviyesinin altındaki voltaja düşürmek, zamanlamaya dayalı hatalara neden olur. Sistemin dayanıklılığını değerlendirmek için MoRS tarafından üretilen hataları DSA hızlandırıcısının yonga üzerindeki belleğine enjekte ediyoruz. MoRS, tamamen rastgele oluşturulmuş bir hata enjeksiyon yaklaşımına kıyasla gerçek deney hatalarına daha yakın olurken, yüksek zamanlı ek yük deneylerine ihtiyaç duymadan basitlik avantajına sahiptir. DSA'nın ağırlık değişkenlerini SRAM'lere eşleyerek popüler DSA iş yüklerinde yaptığımız deneyi değerlendirip MoRS ile gerçek veriler arasındaki sınıflandırma yüzdesinin farkını ölçeriz. Sonuçlarımız, gerçek deneyler ile MoRS'un çıktısı arasındaki maksimum farkın %6,21, gerçek deneyler ile rastgele hata enjeksiyon modeli arasındaki maksimum farkın ise %23.2 olduğunu göstermektedir. Gerçek

verilere sınıflandırma yüzdesi açısından, MoRS'nin çıktısı, rastgele hata enjeksiyonu yaklaşımından 3.21x daha iyi performans gösterir.

Tezin ikinci çalışması olarak SRAM tabanlı belleklerde düşük voltajdan kaynaklanan hataları kullanan SRAM tabanlı gerçek rastgele sayı üreteci öneriyoruz. Gerçek rastgele sayı üreteçleri (GRSÜ), elektriksel gürültü, termal gürültü ve saat titreşimleri gibi öngörülemeyen fiziksel entropi kaynaklarından üretilmektedir. Ancak, tüm bilgi işlem cihazları bu kaynaklardan entropi çıkarmak için özel donanıma sahip değildir. Bu nedenle, özel donanım aracılığıyla bilgi işlem sistemlerine gerçek rasgele sayı üretme yeteneği sağlamak maliyetlidir. Önceki çalışmalar, çoğu hesaplama sistemlerinde mevcut olan bellek aygıtlarından entropi çıkaran GRSÜ'ler önermektedir. SRAM tabanlı GRSÜ'lerin diğer bellek tabanlı mekanizmalara göre iki büyük avantajı vardır: (i) SRAM aygıtları, tüm çağdaş CMOS tabanlı yongalarda kullanıldığı için her ölçekte bilgi işlem sistemine rasgele sayılar sağlayabilir ve (ii) SRAM bir yonga üstü bellek olduğundan kimse rastgele sayı aktarımlarını dışarıdan gözetleyemez ve bu sayede gerçek rastgele sayıların güvenliğini artırır. SRAM tabanlı GRSÜ'ler bu avantajlar nedeniyle umut verici olsa da, mevcut SRAM tabanlı GRSÜ'ler iki temel sınırlamadan muzdariptir: 1) sık sık güç döngüsüne ihtiyaç duyar, bu da aşırı derecede büyük gecikmelere ile düşük verime neden olur ve 2) önemli bir enerji yüküne sahiptir. Amacımız, bu iki temel sınırlamanın üstesinden gelen SRAM tabanlı bir GRSÜ tasarlamaktır. Bu amaçla, yeni bir *yüksek aktarım hızlı*, *enerji verimli* ve *düşük gecikmeli* SRAM tabanlı GRSÜ olan TuRaN'ı öneriyoruz. TuRaN, SRAM besleme voltajının üretici tarafından önerilen eşiğin altına düştüğünde SRAM hücrelerinin okunmasının rastgele değerlerle sonuçlandığı temel gözleminden yararlanır. TuRaN, düşük voltajlı SRAM'a tekrar tekrar erişerek yüksek hızda gerçek rastgele sayılar üretir hücreleri ve ardından SHA-256 karma işlevini kullanarak ortaya çıkan hataları işleyerek gerçek rastgele sayı üretir. TuRaN'ın gerçek rastgele sayılar ürettiğini göstermek için, iki ticari kullanıma hazır FPGA kartı üzerinde gerçek deneyler yapıyoruz. Yaygın olarak benimsenen NIST STS'yi kullanarak TuRaN tarafından üretilen rastgele sayıların kalitesini değerlendiriyoruz ve TuRaN'ın tüm testleri geçtiğini gözlemliyoruz. TuRaN, (i) 1,6 Gbps (1,812 Gbps) ortalama (maksimum) aktarım hızı, (ii) 0,11nJ/bit enerji tüketimi ve (iii) 278,46μs gecikme süresi ile gerçek rastgele sayılar üretir. TuRaN, en son teknoloji ürünü SRAM tabanlı GRSÜ'lerden sırasıyla 2,26x, 5,09x ve 5,39x aktarım hızı, enerji verimliliği ve gecikme süresi ile daha iyi performans göstermektedir.

Anahtar Kelimeler: Bellek, SRAM, Hata modelleme, Gerçek rastgele sayı.

ABSTRACT

Master of Science

TRUE RANDOM NUMBER GENERATION AND FAULT MODELING METHODS FOR REDUCED-VOLTAGE SRAM DEVICES

İsmail Emir YÜKSEL

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

Supervisor: Prof. Dr. Oğuz ERGİN

Date: JULY 2022

SRAM-based on-chip memories are crucial components for various computing devices including heterogeneous devices, *e.g,* GPUs, FPGAs, and ASICs to achieve high performance. Modern workloads such as Deep Neural Networks (DNNs) running on these heterogeneous fabrics depend highly on the on-chip memory architecture for efficient acceleration. One of the common methods to save energy is undervolting *i.e.,* supply voltage underscaling below the nominal level. Such systems can be safely undervolted without incurring faults down to a certain voltage limit. This safe range is also called a voltage guardband. However, reducing voltage below the guardband level without decreasing frequency causes timing-based faults.

In the first study of this thesis, we propose MoRS, a framework that generates the first approximate undervolting fault model using real faults extracted from experimental undervolting studies on SRAMs to build the model. We inject the faults generated by MoRS into the on-chip memory of the DNN accelerator to evaluate the resilience of the system under the test. MoRS has the advantage of simplicity without any need for high-time overhead experiments while being accurate enough in comparison to a fully randomly-generated fault injection approach. We evaluate our experiment in popular DNN workloads by mapping weights to SRAMs and measuring the accuracy difference between the output of the MoRS and the real data. Our results show that the maximum difference between real fault data and the output fault model of MoRS is 6.21%. In contrast, the maximum difference between real data and the random fault injection model is 23.2%. In terms of average proximity to the real data, the output of MoRS outperforms the random fault injection approach by 3.21x.

True random number generators (TRNGs) rely on unpredictable physical entropy sources such as electrical noise, thermal noise, and clock jitters. However, not all computing devices are equipped with dedicated hardware to extract entropy from these sources. Thus, it is costly to provide true random number generation capability to computing systems via dedicated hardware.

Prior works propose TRNGs that extract entropy from memory devices that are present in most computing systems. SRAM-based TRNGs have two major advantages over other memory-based mechanisms: they can (i) provide random numbers to every scale of computing systems as SRAM is used in all contemporary CMOS-based chips and (ii) enhance the security of the true random numbers because no one can snoop the random number transfers as SRAM is an on-chip memory. Although SRAM-based TRNGs are promising due to these advantages, existing SRAM-based TRNGs suffer from two key limitations: they 1) need frequent power cycling, causing prohibitively large latencies and low throughput and 2) have a significant energy overhead.

Our goal is to design an SRAM-based TRNG that overcomes these two key limitations. To this end, we propose our second contribution, TuRaN, a new *high-throughput*, *energy-efficient*, and *low-latency* SRAM-based TRNG. TuRaN leverages the key observation that reading SRAM cells results in random values when the supply voltage is reduced below the manufacturer-recommended threshold. TuRaN generates random numbers at high throughput by repeatedly accessing undervolted SRAM cells and then post-processing the resulting faults using the SHA-256 hash function. To demonstrate that TuRaN generates true random numbers, we conduct real-world experiments on two commercial off-the-shelf FPGA boards. We evaluate the quality of the random numbers generated by TuRaN using widely-adopted NIST STS and observe that TuRaN passes all tests. TuRaN generates true random numbers with (i) an average (maximum) throughput of $1.6Gbps$ ($1.812Gbps$), (ii) $0.11nJ/bit$ energy consumption, and (iii) $278.46\mu s$ latency. TuRaN outperforms the state-of-the-art SRAM-based TRNGs by $2.26x$, $5.09x$, and $5.39x$ throughput, energy efficiency, and latency, respectively.

Keywords: Memory, SRAM, Fault Modeling, 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, destekleriyle her zaman yanımda olan annem Semiha Yüksel, babam Şaban Yüksel, ablam Nur Banu Yüksel ve biricik yeğenim Yekta Çağan Kırık'a, sundukları üretken ve keyifli çalışma ortamı için Kasırga Mikroişlemciler Laboratuvarındaki çalışma arkadaşlarım Ataberk Olgun, Alperen Bolat, Fatma Nisa Bostancı, Yahya Can Tuğrul ve tezimi dikkatlice okuyup düzenlememe yardım eden Oğuzhan Canpolat ve Şevval İzmirli'ye teşekkürlerimi sunarım.

İÇİNDEKİLER

ŞEKİL LİSTESİ

Sayfa

ÇİZELGE LİSTESİ

Sayfa

KISALTMALAR

ASIC : Application-Specific Integrated Circuit
CMOS : Complementary Metal Oxide Semiconductor
CPU : Central Processing Unit
DRAM : Dynamic Random Access Memory
DSA : Derin Sinir Ağları
FPGA : Field-Programmable Gate Array
GPU : Graphics Processing Unit
GRSÜ : Gerçek Rastgele Sayı Üreteci
HBM : High Bandwidth Memory
IoT : Internet of Things
NIST : National Institute of Standards and Technology
RSÜ : Rastgele Sayı Üreteci
SRAM : Static Random Access Memory
SRSÜ : Sözde Rastgele Sayı Üreteci
STS : Statistical Test Suite
TRNG : True Random Number Generator

SEMBOL LİSTESİ

Bu çalışmada kullanılmış olan simgeler açıklamaları ile birlikte aşağıda sunulmuştur.

Simgeler	**Açıklama**
Gb/s (Gbps)	gigabit saniye (aktarım hızı)
Mb/s (Mbps)	megabit saniye (aktarım hızı)
nJ	nano-Joule
mm^2	milimetre kare
ns	nanosaniye
μs	mikrosaniye
ms	milisaniye
V_{min}	Minimmum güvenli voltaj seviyesi
V_{crash}	En düşük çalışma voltaj seviyesi

1. GİRİŞ

SRAMler geleneksel olarak dallanma öngörücüsü (CPUlarda), yazmaç öbeği (GPUlarda), yonga üzerinde arabellek (FPGAlar ve ASICler gibi donanım hızlandırıcılarda) gibi farklı bilgi işlem sistemlerinde farklı bileşenlerin yapı taşıdır. Bununla birlikte, SRAMlerin güç tüketimi, toplam sistem gücüne önemli ölçüde katkıda bulunur. Örneğin, GPUlar [1] üzerindeki önceki çalışmalar, GPUlardaki yazmaç öbeğininin toplam gücün %15-20'sini tükettiğini göstermektedir. Modern sıra dışı yürütüm yapan çekirdekler [2] üzerine yapılan bir başka çalışma da mikroişlemcinin SRAMın yaygın olduğu ön tarafın CPU'nun toplam gücünün yaklaşık %33'üne kadarını tükettiğini göstermektedir. FPGAlar [3] üzerindeki DNN hızlandırıcıları üzerinde yapılan diğer çalışmalar, yonga üzerindeki SRAMlerin toplam gücün %27'sini tükettiğini göstermektedir.

Herhangi bir temel donanımın toplam güç tüketimi, doğrudan besleme voltajıyla ilgili olduğundan, voltajın düşürülmesi, güçten tasarruf etmek için etkili bir çözümdür [4, 5]. Bu teknik, CPUlar [6, 7, 8, 9, 10], GPUlar [11] FPGAlar [3, 12, 13, 14, 15] ve ASICler [16, 17, 18] ve ayrıca bellek aygıtları olan DRAMlar [19, 20, 21], HBMler [22], SRAMlar [23, 24], Flash Diskleri [25, 26, 27, 28] gibi çeşitli cihazlarda yaygın olarak kullanılmaktadır. Bununla birlikte, voltaj düşürüldüğünde, düşük voltaj seviyelerinde artan devre gecikmesi nedeniyle güvenilik sorunları ortaya çıkabilir. Çoğu ticari cihazda, nominal voltaj ile minimum güvenli voltaj, *yani* V_{min} arasında zamanlama hata gözlemlenmeyen aralık olan bir koruma bandı vardır. Bu voltaj koruma bandının altında devre gecikmelerinin bir sonucu olarak hatalar meydana gelir.

V_{min}'ın altında daha fazla voltaj düşürülmesi, güç tüketimini daha da azaltsa da, düşük voltaj hataları nedeniyle sistem güvenilirliğini tehlikeye atar. V_{min} ile en düşük çalışma voltaj seviyesi, *yani* V_{crash}, arasındaki bu belirli bölge, kritik bölge [3] olarak adlandırılır. Bu bölgede sistem hala çalışabilse de, agresif bir şekilde voltaj düşürüldüğünden sistemde hatalar meydana gelmektedir.

Geçmişte birkaç hatayı engelleme tekniği önerildi [3, 16, 17, 29, 30]. Ancak, bu teknikler ya yüksek efor gerektiren mühendislik [3, 30] ya da tamamen rastgele hata yerleştirme tekniği [16, 17] ile gerçeklendi. Bu yaklaşımlardan rastgele hata yerleştirme tekniği yeterince doğru sonuç vermemektedir. Bir diğer yaklaşım olan gerçek deneyler ise yüksek efor gerektiğinden pratik bir çözüm sunmamaktadır. Çözümümüz, hem rastgele hata yerleştirmeunun hem de gerçek hata deneylerin avantajlarını sunar.

Bu tezin ilk çalışması olarak, SRAMler için ilk yaklaşık voltaj düşük ölçekli modeli oluşturan bir altyapı olan MoRS'u öneriyoruz. MoRS üç adımdan oluşmaktadır: 1) Deney, 2) Davranış Çıkarımı ve 3) Model Oluşturma. Deney adımında, MoRS, önceki çalışmamıza [3] dayalı olarak SRAMlerin herkese açık düşük voltajlı hata haritası verilerini kullanır. Davranış Çıkarımı adımında, düşük gerilimli SRAM

bloklarının karakteristik hata davranışı öznitelikleri çıkarılmaktadır. Düşük voltaj kaynaklı hataların rastgele oluşmadığını gözlemlemekteyiz. Bu hatalar birbirleriyle ilişkilidir ve hataların nerede olduğunu etkilemektedir. Satır tabanlı ve sütun tabanlı yaklaşımlarla hataların oluşumunu inceliyoruz ve ardışık hatalı bit hücreleri arasındaki mesafeyi, hem satırlardaki hem de sütunlardaki her bit hatasının sayısını ve SRAM bloğu başına toplam hatalı satır ve sütun sayısını ve dağılımlarını analiz ediyoruz. Bu ince taneli öznitelikler karakteristik davranışlar göstermektedir. Bu adımda, karakteristik öznitelikleri çıkarırız ve bunları iki profilde sınıflandırırız: kaba taneli ve ince taneli profil oluşturma. Bu sınıflandırmanın arkasındaki sebep, rastgele hata yerleştirme çalışmalarının sadece kaba taneli özellikleri, bit hatalarının sayısını ve hatalı SRAM bloklarının sayısını kullanmasıdır. MoRS'un Karma Model olarak adlandırdığımız çıktısı, hem ince taneli hem de kaba taneli özellikleri kullanır ve son adım olarak yaklaşık bir model elde etmek için özel bir olasılıksal modelleme algoritması uygular.

Son yıllarda, DSAlarde SRAMların güç tüketimi önemli ölçüde arttığı gözlemlenmiştir [16, 3, 17]. Yonga üzerindeki SRAMların önemli bir rol oynadığı DSA hızlandırıcıları için MoRS'u değerlendiriyoruz. MoRS, belirli bir etki alanı ile sınırlı olmayan ve diğer önceki çalışmalardan farklı olarak, dallanma öngörücü birimleri, yazmaç öbekleri, önbellekler ve SRAM'a dayalı diğer bellekler gibi birçok alanda potansiyel olarak kullanılabilen hata modelleri üretir. MoRS'u değerlendirmek için, kaba taneli özelliklere düzgün bir rastgele dağılım işlevi uygulayarak bir temel model (rastgele model) oluşturuyoruz. Rastgele Model olarak adlandırdığımız bu temel, standart bir hata yerleştirme şemasıdır ve önceki çalışmalarda [31, 17, 16, 32] kullanılmıştır. Ayrıca, her bir yapay model (Rastgele Model ve Karma Model) ile gerçek veriler arasındaki doğruluk farkını görmek için gerçek deney verilerini işliyoruz.

Değerlendirme metodolojimizde DSA ağırlıklarını SRAM bloklarına eşliyoruz. Ağırlıkları SRAM bloklarına eşlediğimizde, en anlamlı bitten başlayarak eşleme, en anlamsız bitten başlayarak eşleme, bitlerin ilk yarısı en anlamlı bitten başlayarak, bitlerin diğer yarısı en anlamsız bitten başlayarak ve bu durumun tam tersi oalcak şekilde birçok farklı eşleme algoritması uyguluyoruz.

DSAlarde enerji tasarrufu yapmanın başka bir yöntemi de nicemlemedir. Nicemleme yöntemimiz, ağırlıkların hassasiyetini sırasıyla 32-bit'ten 16-bit, 8-bit, 4-bit ve 1-bit'e düşürmektedir. Düşük voltaj gerçekleştirirken, bazı istenmeyen bitler değerini terse çevrilebilir ve ardından karşılık gelen değer kayan virgullü sayıların tabanında sonsuz veya NaN olur. Bu değerlerden kaçınmak için maskeleme teknikleri kullanılır [33]. Sonsuzluğu veya NaN değerini 1 veya 0 olarak maskeleriz. Farklı ağırlık eşlemelerinin, nicemlemelerin ve maskeleme seçeneklerinin ayrı ayrı ve birlikte davranışını DSAların sınıflandırma sonundaki doğruluk yüzdesindeki değişimle inceleriz. Sınıflandırma aşamasında önceden eğitilmiş LeNeT-5 [34] ve cuda-convnet [35] DSA sistemleri üzerinde yaptığımız deneyleri değerlendiriyoruz. Her voltaj seviyesi için sınıflandırma sonucundaki doğruluk yüzdesini inceliyoruz. Deneylerimiz, MoRS tarafından oluşturulan yapay modelin, farklı DSA karşılaştırmalarında gerçek verilere benzer trendi olduğunu ve %96,4'lük bir doğruluğa sahip olduğunu göstermektedir. Ayrıca Rastgele Modelin gerçek verilere yeterince yakın olmadığını %23'e varan doğruluk farkıyla görüyoruz. Sınıflandırma sonucu olarak oluşan doğruluk yüzdeleri açısından

ortalama olarak, Karma Modelimizin gerçek verilerle %3,6 farka sahip olduğunu ve temel rastgele modelden ortalama olarak 3 ila 7 kat daha yakın olduğunu bulduk.

Rastgele sayı üreteçleri (RSÜler), kimlik doğrulama ve iletişim protokolleri gibi birçok güvenlik ve şifreleme uygulamasında rastgele sayılar kullanıldığından güvenli sistemlerin temel yapı taşlarıdır [36, 37]. Güvenliğin yanı sıra rasgele sayılar, sayısal simülasyonlar ve modelleme, endüstriyel testler ve makine öğrenimi gibi birçok başka uygulamada kullanılır [38, 39, 40, 41]. Rastgele sayı üreteçlerinin öngörülemezliği, güvenlik açısından kritik sistemleri çeşitli türlerdeki saldırılara karşı korumak için önemlidir [42, 43]. Rastgele sayılar üretmek için iki ana yaklaşım vardır: sözde rasgele sayı üreteçleri (SRSÜler) ve gerçek rastgele sayı üreteçleri (GRSÜler).

SRSÜler, *tohum (-ing. seed)* [44, 45, 46] adı verilen bir başlangıç değeriyle belirlenen sayı dizilerini oluşturmak için matematiksel formüller veya önceden hesaplanmış listeler [47] kullanır. Diğer yandan, GRSÜler rastgele fiziksel olayları (ör. elektriksel gürültü [48, 49], atmosferik gürültü [50, 51], termal gürültü [52], saat titreşimleri [40], bellek içindeki gürültü [53]) deterministik olmayan gerçekten rasgele sayılar üretmek için kullanır. SRSÜlerin aksine, GRSÜler tarafından üretilen rastgele sayı dizileri, entropi kaynakları deterministik olmadığı için tahmin edilemez ve tekrar aynı sayı üretilemez. Bu nedenle, GRSÜler tarafından oluşturulan rastgele değerler öngörülebilir bir tohum değerine bağlı olmadığından, güvenlik açısından kritik uygulamalar, güvenli çalışmayı garanti etmek için SRSÜler yerine GRSÜleri kullanır.

Modern sistemler, uygulamalara gerçek rastgele sayılar sağlamak için özel donanım GRSÜleri kullanır. Ancak, tüm modern bilgi işlem cihazlarının özel donanım GRSÜleri yoktur (ör. IoTler, mobil sistemler [54, 55]). Bu cihazlarda gerçek rastgele sayıları etkinleştirmek için önceki çalışmalar, SRAMlar [56, 57, 58, 59, 54, 55], DRAMlar [60, 61, 62] ve FLASH bellekler [63, 64] gibi mevcut bellek cihazlarını kullanan GRSÜ mekanizmaları önermektedir. Bu cihazlar arasında SRAM, (i) tüm ticari bilgi işlem cihazlarında bulunan tek bellek cihazıdır ve (ii) yonga üzerinde olduğundan ve üretilen rastgele sayıları yongaya aktarmak için herhangi bir yonga dışı bağlantı gerektirmediğinden daha güvenlidir. Bu avantajlar, SRAM tabanlı GRSÜleri çağdaş CMOS tabanlı cihazlar için umut verici entropi kaynakları haline getirmektedir.

SRAM tabanlı GRSÜler [56, 57, 65, 66, 67, 58, 68, 55, 69, 70, 59, 54] üzerindeki önceki çalışmalar, rastgele değerleri oluşturmak için SRAMların başlangıç değerlerini *(-ing. start-up values)* entropı kaynağı olarak kullanır. Bazı SRAM hücrelerinin başlangıç değerleri, her açılıştaki gürültüye bağlı olarak öngörülemeyen bir değere sahip olabilir. Başlangıç değerlerinden gerçek rastgele sayı üretmek için SRAM'ın kapatılıp açılması gerekmektedir, bu duruma güç çevrimi *(-ing power-up cycle)* da denir. Ne yazık ki, bu mevcut SRAM tabanlı GRSÜler, pahalı güç çevrimi ve çok sayıda okuma erişimi gerçekleştirmeleri nedeniyle önemli dezavantajlardan muzdariptir [71]: *(i)* sürekli çalışmayı sürdüremezler, *(ii)* yüksek gecikmeye sahiptir, *(iii)* yüksek aktarım hızı elde edemez ve *(iv)* yüksek enerji tüketir. Bu dezavantajlarla birlikte, mevcut SRAM tabanlı TRNGleri modern cihazlarda kullanmak pratik değildir.

Bu tezin ikinci çalışmasındaki amacımız, bu dezavantajların üstesinden gelmek ve düşük gecikme süresinde sürekli yüksek aktarım hızında işlev sağlarken gerçek rastgele sayılar üretmek için yaygın cihazlarda uygulanması pratik olan enerji verimli

SRAM tabanlı bir GRSÜ önermektir.

Bir SRAM hücresinin erişim gecikmesinin, düşük voltajda arttığı gözleminden yararlanıyoruz. SRAM bloklarının besleme voltajının üretici tarafından önerilen sınırın altında ölçeklendirilmesinin ve bit hataları elde etmek için nominal gecikme kullanarak SRAM hücrelerine erişmenin rastgelelik sergilediğini gözlemledik. Bu rastgelelik sergileyen hücreleri GRSÜ'müz için entropi kaynağı olarak kullanıyoruz.

Bu amaçla, bu tezin ikinci çalışması olarak SRAM'ın besleme voltajını agresif bir şekilde düşürerek ve ortaya çıkan hataları bir şifreleme karma işlevi kullanıp işleyerek gerçek rastgele sayılar üretmek için SRAM hücrelerini bir entropi kaynağı olarak kullanan yeni bir SRAM tabanlı GRSÜ olan TuRaN'ı öneriyoruz. TuRaN'ı etkinleştirmek için ilk olarak, tek seferlik bir işlem olan düşük gecikmeli bir karakterizasyon adımı kullanarak düşük voltajda, yüksek entropiye sahip SRAM satırlarını belirlemek için SRAM'ı karakterize ediyoruz. Karakterizasyondan sonra, TuRaN karakterizasyon sonrası belirlenen yüksek entropili satırlarda art arda düşük voltaj tabanlı hataları oluşturarak devamlı bir şekilde gerçek rastgele sayılar üretir.

Önceki SRAM tabanlı GRSÜlerle karşılaştırıldığında, TuRaN, başlangıç değerleri yerine SRAM hücrelerinde düşük voltaj hatalarından yararlandığından ve SRAM hücrelerinde daha yüksek entropiye ulaştığından, TuRaN *(i)* devamlı çalışmasını sürdürür, *(ii)* daha düşük gecikmeye sahiptir, *(iii)* daha yüksek aktarım hızı elde eder ve *(iv)* daha az enerji tüketir.

Toplamda 560 blok SRAM içeren Xilinx ZC702 FPGA kartının [72] iki özdeş numunesi üzerinde deneylerimizi ve karakterizasyonumuzu değerlendiriyoruz. Düşük voltaj seviyelerinde çalışan karakterizasyonumuz, frekans, veri örüntüsünün ve sıcaklığın etkisini de hesaba katar. Her parametrenin altındaki satırların entropisini inceliyoruz. Ayrıca frekans ve voltaj arasındaki ilişkiyi entropi açısından analiz ediyoruz.

Deneylerimiz TuRaN'ın standart NIST STS [73] kullanarak gerçek rastgele sayılar ürettiğini gösteriyor. Deneysel sonuçlarımız TuRaN'ın maksimum aktarım hızının $1.812 Gbps$, ortalamanın ise $1.6 Gbps$ olduğunu gösteriyor. TuRaN, gerçek rastgele bit başına $0.11 nJ$ enerji tüketirken $278.46 \mu s$ gecikme süresine sahiptir.

Sistem değerlendirmemizde, son teknoloji ürünü bir CPU [74] benzeri bir sistemi simüle ediyoruz. Bu tür sistemler için TuRaN'ın iki entegrasyon mekanizmasını ele alıyoruz. TuRaN'ı modern sistemlerde etkinleştirmek için, L1 veri ve L2 önbelleklerini Drowsy Cache [75] olarak değiştirmemiz gerekiyor, çünkü bu, her bir önbellek satırının voltaj seviyesini ihmal edilebilir bir maliyetle ölçeklendirmemize izin veriyor. Gerçek rastgele sayılar üretmek için önbelleklerin boşta kalma döngülerinden yararlanırız. Önbellekler rastgele sayılar üretmek için yeterli boşta döngüye sahip olduğunda, karakterizasyon sonrası belirlenen yüksek entropili önbellek satırının voltajını düşürüp ardından rastgele bit akışını elde etmek için okuruz. Bundan sonra, ek alan kullanmamak adına SHA-256 işlevini gerçekleştirmek için CPU'yu kullanırız. Bu iki senaryoyu, gem5 [76] kullanarak SPEC2006 [77] iş yüklerinde değerlendiriyoruz ve TuRaN'ın L1 veri önbelleğinde (L2 önbellek) $4,03 Gbps$ ortalama verimle gerçek rastgele sayılar ürettiğini buluyoruz ($10,95 Gbps$) ve entegrasyon için $0,00165 mm^2$ ($0,0111 mm^2$) yonga alanı ek yüküne sahipken, ortalama olarak %4,86 (%1,92)

performans düşüşüne neden olmaktadır.

Bu tez ile literatüre sunduğumuz katkılar aşağıda listelenmektedir:

- Tezin ilk çalışması: MoRS
  - Ortalama olarak %3,6'lık bir doğruluk farkıyla gerçek düşük voltajlı hata verilerini gerçekçi bir şekilde taklit ederek yapay bir model oluşturan MoRS adlı bir altyapıyı öneriyoruz. Bildiğimiz kadarıyla, bu çalışma, düşük voltaj koşulları altındaki SRAM blokları için ilk yaklaşık doğru modeli oluşturmaktadır.
  - Farklı ağırlık eşlemelerinin, nicemlemenin ve değer maskelemelerinin sınıflandırma doğruluğunu nasıl etkilediğini görmek için DSAlardaki yapay modellerimizi ve gerçek verileri değerlendiririz. Ağırlıkların hassasiyetini düşürmeye devam edersek, DSAların düşük gerilime karşı daha dirençli hale geldiğini önceki çalışmalarla paralel olarak [78, 79] buluyoruz. En düşük voltaj seviyesinde 8-bit LeNeT doğruluğu %14, 4-bit LeNeT doğruluğu %60'tır. Bu olası olmayan durumda bile, Karma Modelimiz gerçek verilere yakın doğruluk yüzdesi üretmektedir.
  - Güvenli sınırın altında agresif voltaj alt ölçeklemesi gerçekleştirerek gerçek rasgele sayıları çıkarmak için SRAMlerden yararlanan yeni bir SRAM tabanlı TRNG olan TuRaN'ı sunuyoruz. Bildiğimiz kadarıyla, bu çalışma gerçek rasgele sayılar üretmek için voltaj alt ölçekleme tekniğini kullanan ilk çalışmadır.
- Tezin ikinci çalışması: TuRaN
  - Gerçek rastgele sayı üretme potansiyelini değerlendirmek için, FPGAlara gömülü birkaç SRAM bloğu kullanarak SRAMlerin düşük voltaj seviyelerinde çalışan davranışını karakterize ediyoruz.
  - Rastgelelik için standart NIST STS'yi kullanarak TuRaN'ın yüksek kaliteli bir GRSÜ olduğunu deneysel olarak değerlendiriyoruz ve TuRaN'dan çıkarılan rastgele bit akışlarının tüm testleri geçtiğini gösteriyoruz.
  - TuRaN'ın aktarım hızının, en son teknoloji SRAM tabanlı GRSÜ çalışmalarından ortalama $2,26x$ daha iyi performans gösterdiğini, maksimum ve ortalama aktarım hızı olarak sırasıyla $1,812Gbps$ ve $1,6Gbps$'a ulaştığını gösteriyoruz. TuRaN, gerçek rastgele sayılar üretmek için güç döngüsüne bağlı olmadığından, son teknoloji SRAM tabanlı GRSÜ'den 5,39 kat daha düşük olan 278,46 μs gecikme süresine sahiptir. Sonuçlarımız, TuRaN'ın enerji tüketiminin, önceki SRAM tabanlı GRSÜlerden $5.09x$ daha düşük olan $0.11nJ/bit$ olduğunu gösteriyor.
  - İki olası sistem entegrasyonunu inceliyoruz ve TuRaN'ın CPU'nun L1 veri önbelleğinde (L2 önbelleği) $4.03Gbps$ ($10.95Gbps$) hızında gerçek rastgele sayılar üretebileceğini ve $0.00165mm^2$ ($0.0111mm^2$)'lik ihmal edilebilir bir alan ek yüküne ve aynı zamanda sistemin %4.86 (%1.92) performans düşüşüne maruz kalabileceğini gösteriyoruz.

2. TEMEL BİLGİLER

2.1 Static Random Access Memory(SRAM)

SRAM'ler, birçok bilgi işlem sisteminde bir yazmaç öbeği, önbellek, dallanma öngörücü birimi ve yonga üzerinde arabellek olarak yaygın olarak kullanılmaktadır. Farklı bit topolojilerine sahip birçok SRAM bit hücresi türü vardır. Bu bölümde, üstün sağlamlığı, paketleme yoğunluğu özellikleri ve tipik olarak ticari cihazlarda kullanılan geleneksel tasarım olduğundan, altı transistörlü SRAM (6T-SRAM) hücresine odaklanıyoruz [80].

2.1.1 SRAM Blok Organizasyonu

Bir SRAM bloğu, satır ve sütun devreleri ile birlikte bir dizi SRAM hücresinden oluşur. Şekil 2.1, bir SRAM blok yapısı ve bir 6T SRAM hücresi örneğini göstermektedir.

Bir SRAM satırı, *satır hattı* (*-ing.* wordline) adı verilen ortak bir kabloyu paylaşan bir dizi SRAM hücresidir. SRAM hücrelerinin bir sütunu, *bit hattı* (*-ing.* bitline) olarak adlandırılan aynı kabloya bağlanır. Bir satır kod çözücüsü (*-ing.* row decoder), erişilen satırın adresini çözer ve satır hatlarından birini etkinleştirir. Sütun devresi, sütun kod çözücü (*-ing.* column decoder), önyükleme devresi (*-ing.* precharge circuit) ve algılama yükseltecinden (*-ing.* sense amplifier) oluşur. Sütun kod çözücüleri, bir satırdaki hücrelerin yalnızca bir alt kümesi için okuma veya yazma işlemlerini gerçekleştirmek adına tek bir algılama yükselticinin sütunlar arasında paylaşılmasına izin verir. Bir besleme voltajında (VDD) sütunun iki bit hattının voltaj seviyesini ayarlamak için bir önyükleme devresi kullanılır. Bir algılama yükselteci, bit hatlarındaki küçük voltaj farkını yükseltir ve bir dijital çıkış olarak, mantık-0 veya mantık-1 üretir. Geleneksel bir 6T SRAM hücresi, altı transistörden oluşur. Hücre, bir döngüye bağlı iki özdeş CMOS tersleyiciden ve bit hatlarını (*bl* ve *bl_b*) ve satır hattını (WL) bağlamak için iki erişim transistöründen (AT1 ve AT2) oluşur.

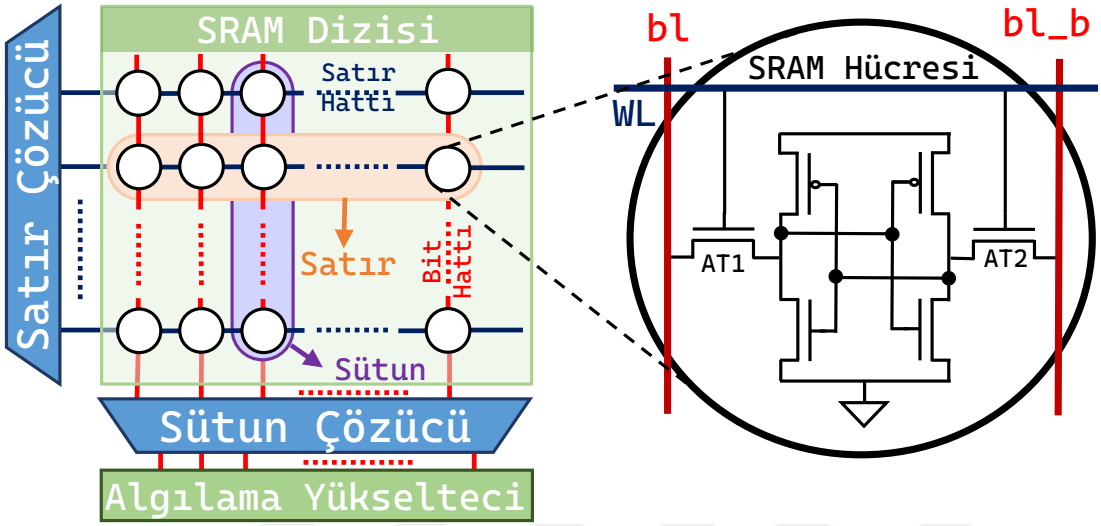
2.2 SRAM'lerde okuma işlemi

SRAM okuma işlemleri iki kısımdan oluşur: hücrede ve algılama yükseltecinde.

Hücre İşlemleri. Genelliği kaybetmeden, SRAM hücresinin (Q düğümü) mantık-0'ı depoladığını varsayalım. Dolayısıyla Q_b mantık-1'dir. Şekil 2.2 bu örnek hücredeki okuma işlemini üç adımda göstermektedir. Bir okuma işlemini başlatmadan önce, bit hatları önceden VDD olarak yüklenir. AT1 ve AT2, kapıları mantık-0'a sahip olduğu için kapalıdır (Adım ❶).

Adım ❷'de, satır hattı yükseltilir ve bir hücrenin okuma işlemi başlar. Bu adımda AT1 ve AT2, hücreyi önceden yüklenmiş bit hatlarına bağlar. P1 ve D1'in kapıları mantık-0'a sahip olduğu için ikisi de kapalıdır. Q, mantık-0'ı sakladığından, D1, toprağa giden bir yol (yeşil çizgi) oluşturur ve bu, *bl* voltajının mantık-0 olmasıyla sonuçlanır.

Adım ❸'de, algılama yükselteci iki bitlik hattın (*bl, bl_b*) küçük voltaj farkını bit hatlarının tamamen boşalmasını beklemeden yükseltir ve çıkış değeri mantık-0 olarak dışarı sürülür.



Şekil 2.1: SRAM blok ve hücre organizasyonu

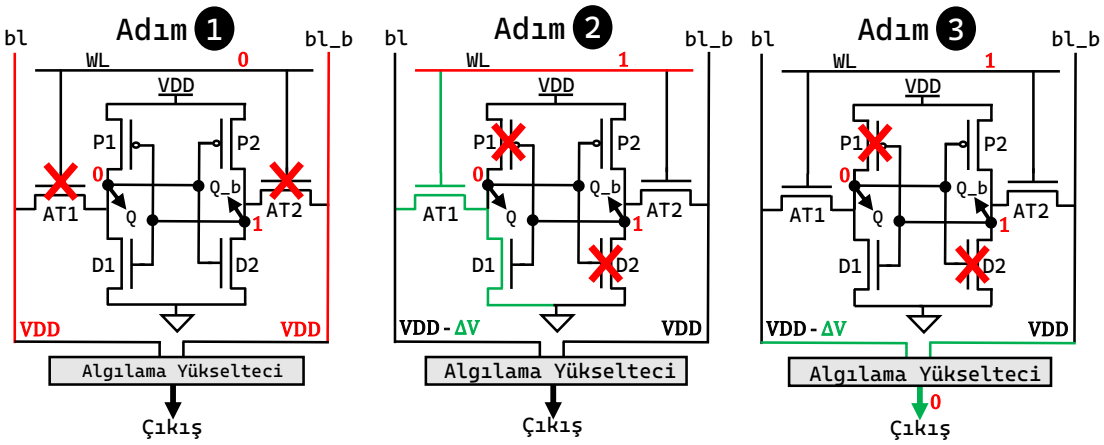
Algılama Yükselteci İşlemleri. SRAM bloklarındaki algılama yükselteçleri, bit hatlarındaki küçük analog diferansiyel voltajı algılar. Böylece bu mekanizma, bit hatlarının tamamen salınımı için beklemeyerek gecikmeyi ve enerji tüketimini azaltır. Geleneksel bir mandallı (*-ing.* flip-flop) tip algılama yükselteci, girişlerini başlangıçta önyükleme voltajı seviyesine ayarlar. Bir SRAM hücresi okunduğunda ve bit hatları boşaldığında, yeterli bir diferansiyel voltaj oluşturur. Algılama yükselteci, algılama yükselteci etkinleştirme sinyalini tetikleyerek diferansiyel voltajı bit hatlarında kilitler [81, 80]. Bundan kısa bir süre sonra, erişilen sütunlar, bit hatlarının algılama yükselteci tarafından boşaltılmasını önlemek için sütun çoklayıcı sinyalini yükselterek algılama yükseltecine bağlanır [82, 83].

2.3 Voltaj Düşürme

CMOS, mevcut bilgi işlem cihazları için baskın devre teknolojisidir. CMOS tabanlı SRAM'lerin güç tüketimi iki bölümün toplamından oluşur: birincisi aktif güç olarak da adlandırılan dinamik güç, diğeri ise statik güçtür. Daha önceki çalışmalar [84, 85, 86], SRAM'lerin güç tüketimine dinamik gücün hakim olduğunu göstermektedir.

Dinamik güç *yani* P_{dyn}, transistörlerdeki bazı düğümlerde anahtarlama etkinliği olduğunda dağıtılır. Statik güç *yani* P_{leak}, cihaz aktif değilken bile akan akımlar tarafından dağıtılır. Matematiksel olarak, bu güç denklemleri şu şekilde verilir:

$$P_{dyn} \propto f \times V_{dd}^2 \tag{2.1}$$



Şekil 2.2: Bir SRAM hücresindeki okuma işlemi

$$P_{akma} = k_{tasarim} \times n \times I_{akma} \times V_{dd} \tag{2.2}$$

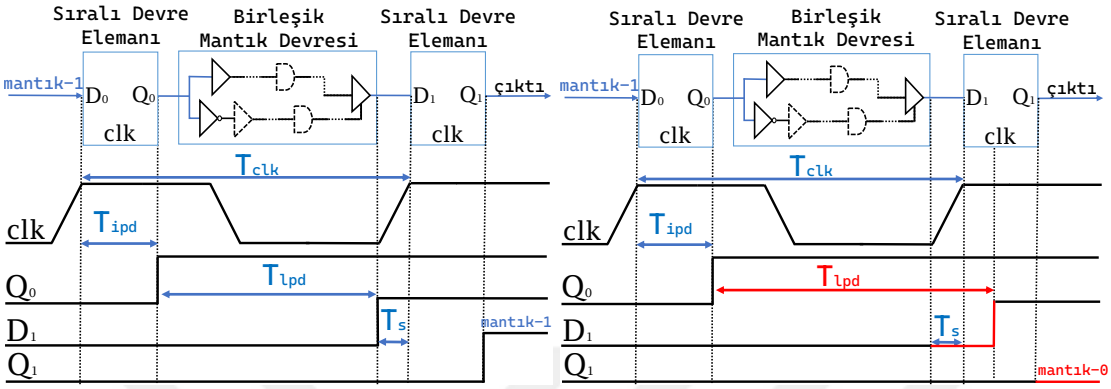
Burada V_{dd} besleme voltajını gösterir, f Denklem 2.1'deki çalışma frekansını gösterir. Ayrıca, n transistör sayısını gösterir, $k_{tasarim}$ tasarıma bağlı parametredir ve I_{akma}, Denklem 2.2'de teknolojiye bağlı bir parametre olan akma akımı gösterir. 2.1 denklemine göre, besleme gerilimi ve çalışma frekansı ayarlanarak dinamik güç tüketimi azaltılabilir. Benzer şekilde, 2.2 denkleminden, besleme voltajının ölçeğinin düşürülmesi ve toplam transistör sayısının azaltılmasıyla akma güç tüketimi azaltılabilir. Toplam güç tüketimi, besleme voltajının ölçeğinin düşürülmesiyle azaltılabilir.

Voltaj alt ölçeklendirme, enerji verimli uygulamalar için yaygın olarak kullanılan bir tekniktir. 2.1 denkleminden, dinamik güç tüketimi, besleme voltajının düşürülmesiyle ikinci dereceden azaltılır. Bu teknik, çalışma frekansının değişmemesi nedeniyle nominal voltaj seviyesi *yani* V_{nom} performansına ulaşabilir. Bununla birlikte, minimum güvenli voltaj *yani* V_{min}'ın altındaki daha da düşük voltaj, zamanlama hatalarının sonucu olarak güvenilirlik sorunlarına neden olabilir. V_{nom} ile V_{min} arasına voltaj koruma bandı denir. Bu koruma bandı, en kötü çevresel durumda bile doğru işlevsellik güvencesi sağlamak içindir. Önceki çalışmalar, güç tüketimini voltaj düşürme teknikleri uygulayarak; FPGA yonga üstü belleklerde %39 [3], GPU'larda %20 [87] ve DRAM'larda %16 [19] herhangi bir hata oluşmadan güç tüketiminin azaltıldığını göstermektedir. Ayrıca, son araştırmalar, FPGA'nın [12] voltajı düşürülen dahili bileşenlerinin yaklaşık 3 kat güç verimliliğine yol açtığını ve HBM'lerin [22] voltaj alt ölçeklendirmesiyle toplam 2,3 kat güç tasarrufu sağladığını göstermektedir. Minerva [16]'da SRAM voltajını düşürerek toplam 2,7 kat güç tasarrufu sağladığını raporlamıştır.

2.3.1 Düşük-gerilimden meydana gelen zamanlama hataları

Sıralı mantık devreleri, iki sıralama elemanı sırası (örneğin yazmaçlar ve mandallar) arasındaki çoklu kombinasyonel mantık seviyesinden oluşur. Girişi (ilk sıralama elemanının çıkışı) çıkışa (sonraki sıralama elemanının girişi) uygun şekilde yaymak için devrenin gecikme ve zamanlama kısıtlamalarını karşılaması gerekir. Şekil 2.3a,

sıralı bir devrenin D0 girişinden Q1 çıkışına veri yayılımının örneğini ve zamanlama diyagramını göstermektedir. Yol, verileri sıralama elemanının (Q0) çıkışına yaymak için birinci elemanın (D0) girişini tetikleyen devre saatinin (clk) yükselen kenarı ile başlar. Kombinasyonel mantık yoluyla, bir sonraki yükselen saat kenarı gelmeden önce Q0 bir sonraki sıralı bloğun (D1) girişine aktarılmalıdır.



(a) Nominal-Voltajdaki Zamanlama (b) Düşük-Voltajdaki Zamanlama

Şekil 2.3: Nominal-Voltajda ve Düşük-Voltajdaki veri aktarımının zamanlama diyagramı

Kombinasyonel devrenin gecikmesi çok büyükse, alıcı sıralama bloğu kurulum süresini kaçırabilir ve maksimum gecikme hatası olarak adlandırılan sorun nedeniyle verileri yanlış örnekleyebilir [82]. Sıralı bir devrenin zamanlaması üç temel zamanlama gecikmesinden oluşur [88, 82, 83]: 1) sıralama elemanlarının yayılma gecikmesi (T_{ipq}), 2) mantık yayılma gecikmesi (T_{lpd}) ve 3) toplam kurulum süresi (T_s). Devrenin doğru çalışması için saat periyodu (T_{clk}) bu üç gecikmenin toplamından büyük veya eşit olmalıdır. Bu tür devrelerde hataları oluşturmak için iki seçenek vardır: (i) saat frekansını artırarak (*yani,* hız aşırtma) T_{clk}'ı azaltmak veya (ii) yayılma gecikmesini artırmak için besleme voltajını düşürmek, T_{lpd}. Şekil 2.3b, voltaj düşürüldüğünde oluşan bir hata durumunun örneğini gösterir.

2.4 Hata Yerleştirme Teknikleri

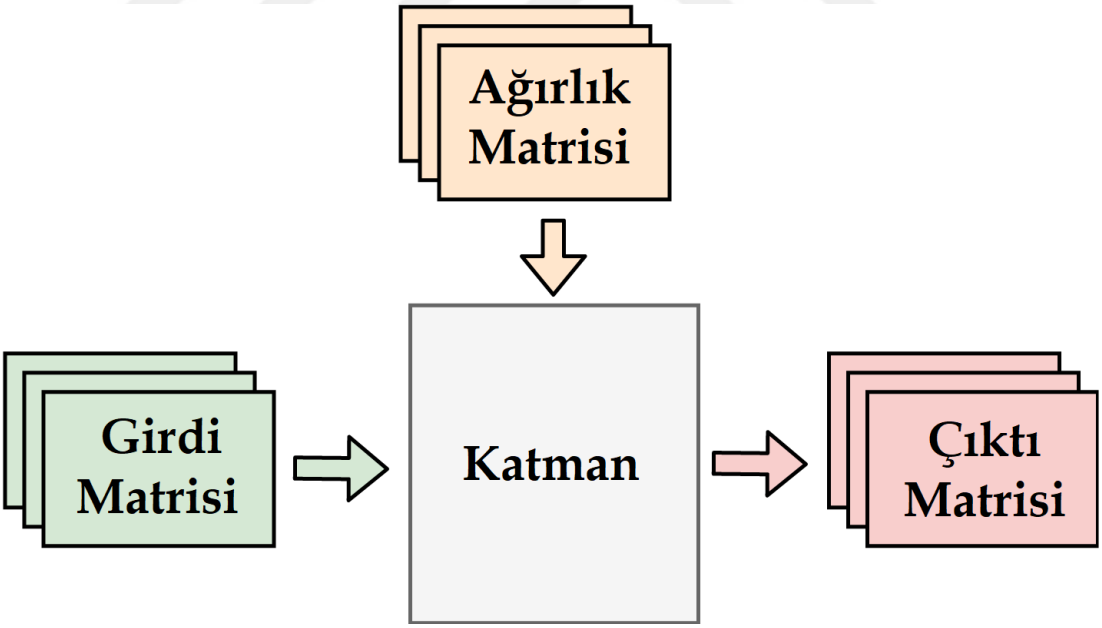
Hata yerleştirme mekanizması, farklı koşullar altında sistemlerin davranışını incelemenin bir yoludur. Düşük voltajlı SRAM'ler için hata yerleştirme çalışması üzerinde önceki çalışmalar, dallanma öngörücüsü birimlerinde [89, 31], önbelleklerde [90, 91] ve FPGA yonga üstü belleklerde [32] uygulanmıştır. Hata yerleştirmeu için mühendislik çabası ve yerleştirme doğruluğu üzerinde ödünleşme olan çeşitli yaklaşımlar vardır. i) Birincisi, gerçek deneylerden herhangi bir bilgi kullanılmadan rastgele konumlara rastgele hatalar uygulamaktır. ii) Bir diğeri, gerçek hata verileri doğrudan hata haritası olarak kullanmaktır. iii) Sonuncusu, gerçek deneylerden elde edilen gerçek verilere dayanan ve gerçek verilere yeterince yakın olan yaklaşık modellemedir. Çizelge 2.1, çaba ve gerçek verilere ne kadar yakın oldukları açısından bu üç hata yerleştirme tekniğinin karşılaştırmasının bir özetini verir. Doğruluk yüzdesi bu çalışmanın sonuçlarından alınmıştır.

Çizelge 2.1: Mühendislik çabası ve gerçek verileri temsil etme doğruluğu açısından hata yerleştirme tekniklerinin karşılaştırılması.

Hata Yerleştirme Yöntemi	Mühendislik Eforu	Doğruluk (Min)
Rastgele Hata Yerleştirme	Düşük	77%
MoRS	Yüksek	94%
Gerçek Deney	Yüksek	100%

2.5 Derin Sinir Ağları

Derin Sinir Ağları (DSA), nesne tanıma, sınıflandırma, segmentasyon ve diğer birçok alanda etkili bir çözüm olarak yaygın olarak kullanılmaktadır. MNIST [92] ve CIFAR-10 [93] veri kümeleri, DSA'lardaki en son teknolojik gelişmeleri sergilemek için yapay öğrenme topluluğu tarafından yaygın olarak kullanılmaktadır. DSA'lar iki aşamada çalışır: eğitim ve sınıflandırma (çıkarım). İlk süreç, DSA'ların ağırlıklarının veri kümelerine göre uydurulmasının eğitimidir. Eğitim aşaması esas olarak CPU'lar veya GPU'lar gibi yüksek performanslı bilgi işlem platformlarında gerçekleştirilir. İkinci aşama çıkarım aşamasıdır, ilk aşamada eğitilen DSA modelleri, daha önce görülmemiş girdi verilerini sınıflandırmak ve çıkarım için kullanılır. Genellikle, eğitim aşaması bir kez gerçekleştirirken, çıkarım aşaması tekrar tekrar gerçekleştirilir. Çıkarım aşaması temel olarak üç unsurdan oluşur: girdi, ağırlık ve çıktı. Şekil 2.4, bir



Şekil 2.4: Yüksek seviyede soyutlanmış DSA'nın çıkarım katmanı

çıkarım katmanının yüksek seviyeli bir soyutlamasını gösterir. Ağın her bir çıkarım katmanı, bir girdi, ağırlık ve çıktı matrisinden oluşur. Bu katmanların ağırlıkları, yonga üzerindeki belleklerde saklanır. Bazı çalışmalar, voltaj alt ölçeklendirme [3, 12], mimari teknikler [94, 95, 96, 97, 98, 99, 100, 101] ve donanım düzeyinde teknikler [102, 103, 104] uygulayarak enerji verimliliğini artırır ve DSA'ların toplam güç tüketimini azaltır.

Nicemleme, veri türlerinin değer hassasiyetini değiştiren bir mimari optimizasyon tekniğidir. Bu teknik, performansı ve enerji verimliliğini artırabilir. Son araştırmalar [105, 16, 106, 107, 108], nicemlemenin bir dereceye kadar DSA modellerinin doğruluğunu önemli ölçüde etkilemediğini göstermektedir. Çalışmamızda, ağırlık hassasiyetini 16-bit yarı-hassasiyetli kayan nokta, sabit nokta biçiminde 8-bit ($Q_{4.4}$), 4-bit ($Q_{2.2}$) ve 1-bit ikili değerler olarak dönüştürüp sınıflandırma doğruluğuna olan etkisini test ettik.

2.6 Gerçek Rastgele Sayı Üreteçleri

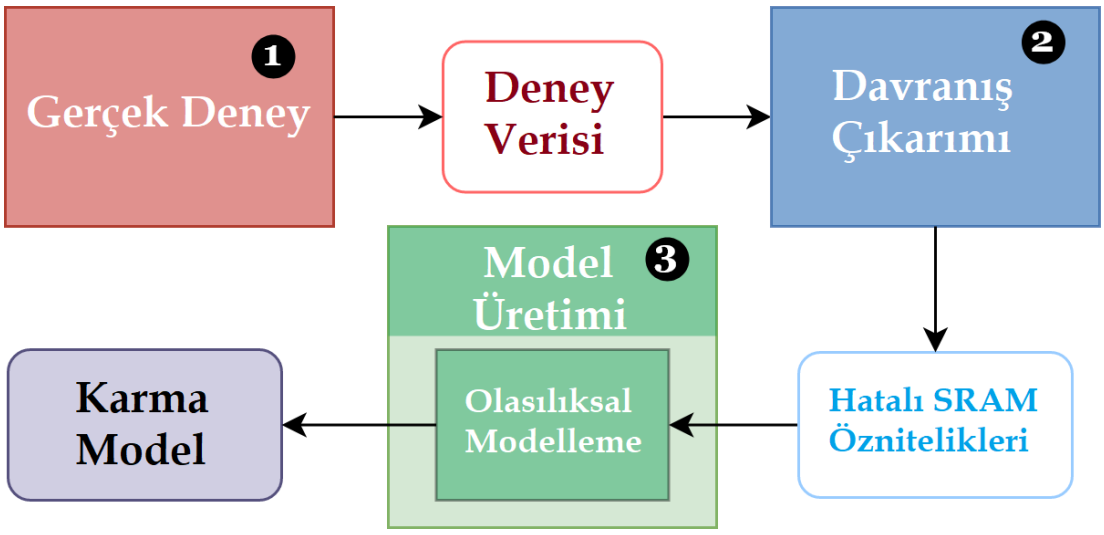
Gerçek rastgele sayı üreteçleri (GRSÜ'ler), hesaplama algoritmaları yerine özel donanımlara dayanır. Tipik tahmin edilemeyen GRSÜ kaynakları, termal gürültü [109], saatlerde titreşim [110], rastgele telgraf gürültüsü (RTN) [111] ve mandalların yarı kararlı salınımı [112] gibi deterministik olmayan fiziksel süreçlere dayanır. GRSÜ'ler, istatistiksel olarak ilişkisiz ve bağımsız bitler oluşturmak için bu rastgele fiziksel olayları örnekler. GRSÜ'ler kullanılarak oluşturulan rasgele sayı dizileri, sözde rasgele sayı üreteçlerinin (SRSÜ'ler) aksine bir *tohum* değerine bağlı değildir. GRSÜ'ler tipik olarak bir değere daha yatkın (yanlılık) entropi kaynaklarını örnek alır. Bir GRSÜ'nin çıkış bit akışları genellikle daha yüksek oranda mantık-1 veya mantık-0 değerleri içerir. Azaltılmış aktarım hızı ve artan gecikme pahasına GRSÜ bit akışlarındaki yanlılığı ortadan kaldırmak için işleme sonrası yöntemleri kullanılır. İşleme sonrası yöntemleri, basit işlevlerden (örneğin, von Neumann Düzeltici [39]) değişen oranlarda son işleme yeteneklerine sahip kriptografik karma işlevlerine (örneğin, SHA-256) kadar uzanır.

3. MoRS

3.1 MoRS Mekanizması

Yaklaşık yapay düşük voltajlı SRAM hata modelleri oluşturmak için bir altyapı olan MoRS'yi öneriyoruz. Bu mekanizma, gerçek hata haritalarına dayanan ilk altyapıdır. MoRS, tam donanım ve tam yazılım hata yerleştirme tekniklerinin ortasında yer alır. Bu altyapı, tam yazılım hata yerleştirme mekanizmasına kıyasla gerçek verilere yeterince yakın olan yaklaşık bir model oluşturur. Ayrıca, MoRS, gerçek deneylerden elde edilen gerçek verileri kullanan tamamen donanım yaklaşımına kıyasla yüksek çaba gerektiren mühendislik gerektirmez.

Şekil 3.1'de gösterildiği gibi MoRS üç adımdan oluşur: ❶ Deney, ❷ Davranış Çıkarma ve ❸ Model Oluşturma. İlk olarak, Bölüm 3.1.1'de gerçek SRAM bloklarından gerçek hata haritaları elde ettiğimiz mevcut deneyin ilk adımını açıklıyoruz. Bölüm 3.1.2'de, ilk adımın çıktısını kullanarak hata haritalarının davranışını ince taneli ve kaba taneli öznitelikler olarak yorumluyoruz. İnce taneli profil oluşturmada, SRAM bloklarının fiziksel mesafe, satır ve sütunlardaki her bit hatasının sayısı ve blok başına hatalı satır ve sütun sayısı açısından satır ve sütun davranışlarını çıkarmaktayız. İri taneli profil oluşturma, genellikle önceki hata yerleştirme çalışmalarında kullanılan sığ bir yaklaşımdır. Kaba taneli profil oluşturmada, yalnızca toplam bit hatası ve hatalı SRAM blok sayısını çıkarılmaktadır.



Şekil 3.1: MoRS'un Genel Mekanizması

Son adımda, gerçek verilere yaklaşık bir model olan Karma Model üretmekteyiz. Bu model, hem ince taneli hem de kaba taneli hata öznitelikleriyle olasılıksal modelleme algoritması kullanarak üretilir. Şekil 3.1, MoRS'un üç adımına genel bir bakış sağlar.

3.1.1 Gerçek deney

Voltaj düşürme, enerji tasarrufu için yaygın olarak kullanılan bir tekniktir. Deneydeki her SRAM bloğu 1024 satır ve 16 sütunlu 16 Kbit'tir.

Bu deneysel çalışmayı, kullanıma hazır Xilinx FPGA'larda, (VC707, ZC702'de ve KC705'in iki özdeş örneğinde (KC705-A ve KC705-B olarak)) bulunan SRAM blokları üzerinde gerçekleştiriyoruz. Çizelge 3.1, konuşlandırılmış FPGA'ların ayrıntılarını gösterir. Bu deney iki adımdan oluşmaktadır. İlk olarak düşük voltaj hatalarını görmek için SRAM bloklarına okuma ve yazma yapılır. İkinci adımda, FPGA'ların bir güç yönetim birimini kullanarak SRAM bloklarının besleme voltajı ayarlanır. FPGA'lar, kendi üzerindeki voltaj raylarını izleyen ve ayarlayan bir voltaj regülatörü olan bir güç yönetim veri yoluna sahiptir. Bu veri yolu, PMBUS standartlarını kullanır ve komutları yürütmek için bir I2C protokolüne sahiptir. FPGA'ların SRAM bloklarının besleme gerilimi V_{CCBRAM} olarak adlandırılır. V_{CCBRAM}'ı PMBUS aracılığıyla azaltarak, yalnızca SRAM bloklarının güç tüketimi azalır. Bu durum diğer mantık elemanlarını (DSP'ler, LUT'lar vb.) etkilemediği için düşük gerilimin SRAM blokları üzerindeki etkisi açıkça test edilip görülmektedir. Bu metodolojinin kurulumu Şekil 3.2'de gösterilmektedir.

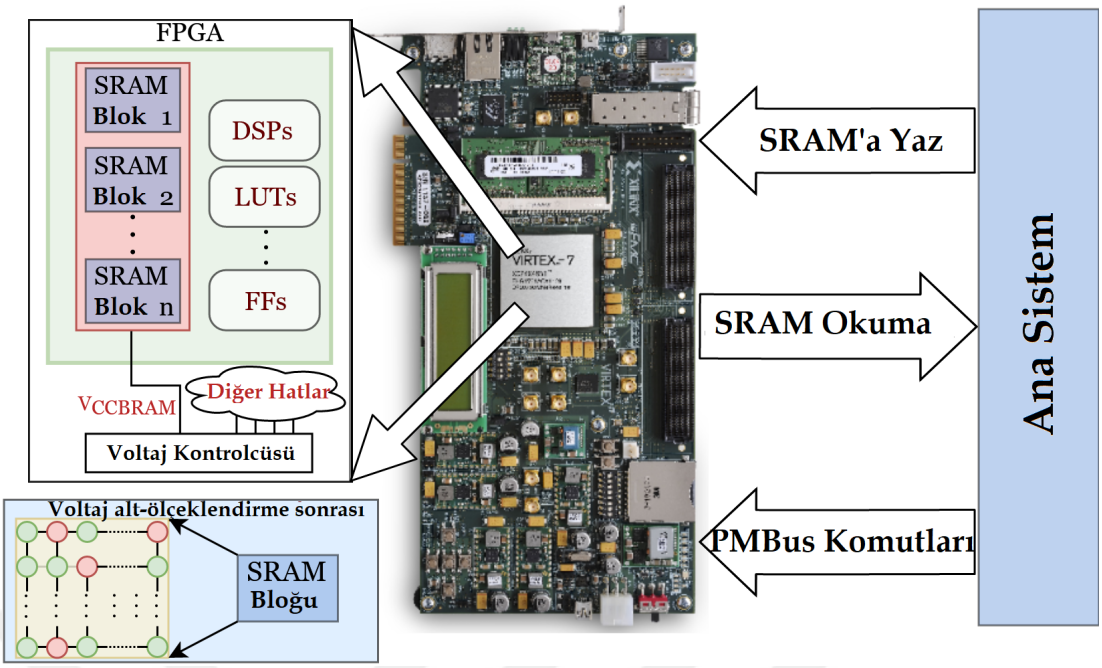
Çizelge 3.1: Deneyde kullanılan FPGA kartlarının özellikleri

Kart	**VC707**	**ZC702**	**KC705**
Teknoloji Düğümü	28nm	28nm	28nm
Nominal Gerilim	1V	1V	1V
Sıcaklık	$50°C$	$50°C$	$50°C$
En Düşük Voltaj Seviyesi	0.54V	0.53V	0.54V
SRAM Blok Sayısı	2060	280	890

Bu deneyin yöntemi, ilk adımda SRAM'lere veri yazan, ikinci adımda hataları oran ve konum açısından analiz eden, üçüncü adımda besleme gerilimini $10mV$ azaltan ve son olarak bunları tekrarlayan bir algoritmayı takip eder. FPGA çalışmayı durdurana kadar bu adımlar tekrarlanır.

Güvenli voltaj seviyesinin, V_{min}, altına düşük voltaj uygulandığında, hata oranı katlanarak artar. Voltaj, FPGA'nın çalışmayı durdurduğu voltaj seviyesine (V_{crash}) kadar SRAM'ların besleme voltajı düşürülebilir. V_{min} ile V_{crash} arasında zamanlama hataları meydana gelirken, güç tüketimi önemli ölçüde azalır. SRAM tabanlı gerçek hata haritaları, aynı platformun farklı FPGA'ları için bile süreç varyasyonunun bir sonucu olarak önemli ölçüde değişmektedir. Ayrıca, çalışma, hata örüntüsünün çoğunlukla kalıcı olduğunu göstermektedir. En önemlisi, düşük voltaj tabanlı hatalar, farklı SRAM'lar üzerinde eşit olarak dağılmadığı gözlemlenmiştir.

Yapay hata haritaları oluşturmak için ilk aşamanın (Deneysel Veriler) çıktısı olarak önceki çalışmanın [3] genel kullanıma açık verilerini [113] kullanıyoruz. Davranışı anlamak için 2000 SRAM bloğu ve altyapımızı test etmek için 950 SRAM bloğu, (VC707'den 2060 SRAM bloğu ve KC705'ten 890 SRAM bloğu) kullanıyoruz. Yapay modelimizi oluşturmak ve test etmek için farklı veri setleri kullanarak MoRS



Şekil 3.2: Gerçek Deneyin Yöntembilimi

metodolojisini güvenilir hale getiriyoruz.

V_{crash}'da, VC707 ve KC705-B için 1 Mbit başına sırasıyla %0,06 ve %0,005'e kadar hata oranı bulunmaktadır. Hataların VC707 için V_{min} = 0,6V, V_{crash} = 0,54V ve KC705-B için V_{min} = 0,59V, V_{crash} = 0,53V arasında görüldüğü gözlemlenmiştir. Önceki çalışmada [3] belirtildiği gibi, VC707 diğer üç kart arasında en fazla bit hatasına sahiptir. En düşük voltaj seviyesinde, VC707'de 23706 bit arızalı %10,2 hatalı SRAM bulunur. Bu önceki çalışma hakkında daha ayrıntılı bilgi için bu referansı [3] önermekteyiz.

Çizelge 3.2: Öznitelikler ve hangi modellerde kullanıldıkları

Profilleme Biçimi	**Öznitelik**		**Model**	
	İsim	**Terim (Yüzde olarak)**	**Karma Model**	**Rastgele Model**
Kaba taneli	Toplam bit hatası	P_F	✓	✓
	Toplam hatalı SRAM bloğu sayısı	P_S	✓	✓
İnce taneli	SRAM bloğu başına toplam hatalı satır sayısı	$P_{SR_{0..1024}}$	✓	✗
	SRAM bloğu başına toplam hatalı sütun sayısı	$P_{SC_{0..16}}$	✓	✗
	Satırlardaki bit hatası sayısı	$P_{FR_{0..15}}$	✓	✗
	Sütunlardaki bit hatası sayısı	$P_{FC_{0..1023}}$	✓	✗
	Bir satırda ardı ardına bulunan hatalar arasındaki uzaklık	$P_{FDR_{1..15}}$	✓	✗
	Bir sütunda ardı ardına bulunan hatalar arasındaki uzaklık	$P_{FDC_{1..1023}}$	✓	✗

3.1.2 Davranış çıkarımı

Bölüm 3.1'de bahsettiğimiz gibi, voltaj düşük ölçekleme hatalarının örüntüsü vardır ve bu hata örüntüleri çoğunlukla benzerdir. Bu bit hatalarının olasılığını etkileyen öznitelikler bulunmaktadır. Bu adımda, bu tür önemli öznitelikleri çıkarmak için düşük voltajla ilgili bit hatalarının davranışının profilini çıkarıyoruz. Profillemeyi iki adımda gerçekleştiriyoruz: kaba taneli profilleme ve ince taneli profilleme. Tüm özelliklerin, profil oluşturma türlerinin ve bunların hangi modelde kullanıldığının bir özeti Çizelge 3.2 içinde bulunabilir.

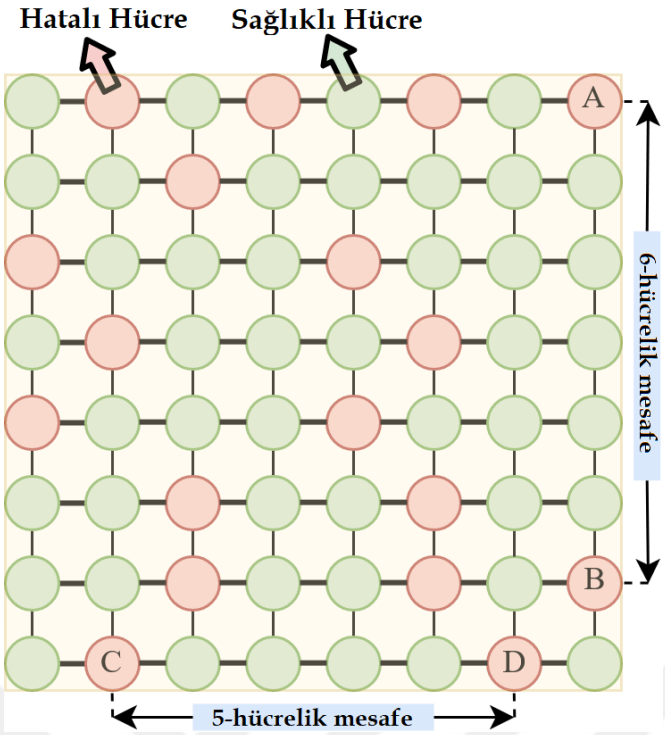
Kaba taneli profil oluşturma iki özellikten oluşur. İlki, P_F, tüm SRAM'lerdeki bit hatalarının yüzdesidir. İkincisi, P_S , tüm SRAM'lardaki hatalı SRAM bloklarının yüzdesidir. VC707 için, V_{crash}'da toplam bit hataları %0.07 ve hatalı SRAM bloklarının yüzdesi %10,2'dir.

İnce taneli profil oluşturma iki bölümden oluşur: satır tabanlı, sütun tabanlı. Hem satır tabanlı hem de sütun tabanlı üç özellik vardır. Birincisi, hatalı SRAM bloklarındaki hatalı satırların ($P_{SR_{0..1023}}$) ve hatalı sütunların ($P_{SC_{0..15}}$) yüzdesidir. İkincisi, satır başına ($P_{FR_{0..15}}$) veya sütun başına ($P_{FC_{0..1023}}$) oluşan bit hatalarının yüzdesidir. Sonuncusu, aynı satırdaki ($P_{FDR_{1..15}}$) veya sütun ($P_{FDC_{1..1023}}$) içindeki ardışık olarak hatalı hücreler arasındaki fiziksel mesafenin yüzdesidir. Bu adım sonucundaki testlerden satır ve sütun için bu iki özelliğin rastgele veya düzgün dağılmadığını keşfettik.

V_{crash}'da,

- Her bit hatasının yüzdesi
  - Satır açısından 2 bitlik hatalara (P_{FR_2}) ve hata olmama (P_{FR_0}) ağır basmaktadır.
  - Sütun açısından, 10 bitten fazla hata ($P_{FC_{0..10}}$) neredeyse olmamaktadır.
- Ardışık olarak hatalı bit hücreleri arasındaki bit hücre mesafelerinin yüzdesi
  - Satır açısından 8 bitlik mesafede yoğunlaşır (P_{FDR_8}).
  - Satır açısından 8-bit uzaklığından fazla uzaklık bulunmamaktadır. ($P_{FDR_{9..15}}$).
  - Sütunlar için, $P_{FDC_{0..2..1024}}$ azalan düzende çift sayılarda yoğunlaşmıştır, 2 bit hücre mesafesi (P_{FDC_2}) en yüksek yüzdeye sahiptir 1022 bit hücre mesafesi ($P_{FDC_{1022}}$) çift sayılarda en düşük mesafeye sahiptir.
  - Ayrıca, sütunlar için, tek sayılardaki ($P_{FDC_{1..3..1023}}$) hücre mesafeleri 0 ile 4 arasında yoğunlaşmıştır.

Örnek olarak, Şekil 3.3'de bir $8 * 8$ SRAM bloğunun hata davranışını gösteriyoruz. Bu örnekte, A hücresi ile B hücresi arasındaki sütun tabanlı uzaklık 6 bit hücredir. C hücresi ve D hücresi için satır tabanlı mesafe 5 bit hücredir. Şekil 3.3'deki bu örneği hem satır tabanlı hem de sütun tabanlı bit hatası açısından incelediğimizde, C hücresini içeren bir sütun 2 bitlik hatalara sahiptir. Ayrıca, A hücresinin satırında 4 bit hata bulunmaktadır. Kaba taneli profil özelliklerinden biri, toplam bit hatalarının yüzdesidir



Şekil 3.3: Örnek bir hatalı SRAM bloğu ve fiziksel özelliklerinin gösterimi

(P_F). Toplam hatalı hücre sayısının tüm hücrelere bölünmesiyle hesaplanabilir. Bu nedenle, Şekil 3.3'in SRAM bloğu P_F için %28.125'dir.

Şekil 3.3'deki bu örneğin davranışını çıkardığımızda, bu işlemi deneysel veriler için gerçekleştirdik. Profil oluşturma adımı tamamlandığında, bu iri taneli ve ince taneli profilleri olasılıksal modelleme ve düzgün rastgele dağılım ile kullanarak modeller oluşturmaya başlıyoruz.

3.1.3 Model üretimi

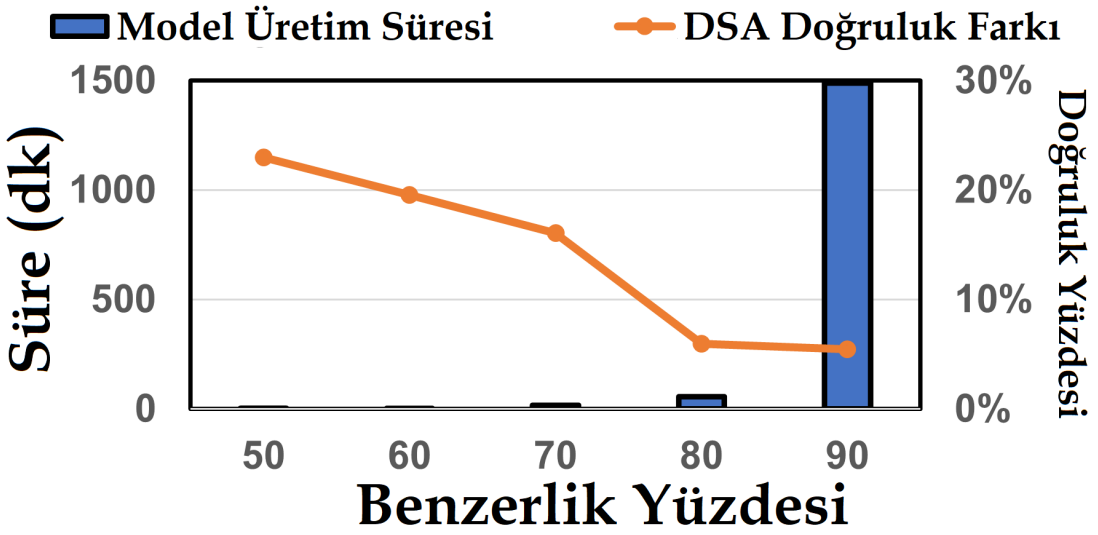
Olasılıksal modelleme [33] ve tek biçimli rastgele dağılım [33, 17, 114, 89], hata haritaları oluşturmak ve hataları yerleştirmek için birçok modelleme çalışmasında yaygın olarak kullanılmaktadır. Yaklaşık bir model, MoRS'un çıktısı olan, Karma Model için, özel bir olasılıksal modelleme işleviyle hem ince taneli hem de kaba taneli öznitelikler kullanıyoruz. Bunlara ek olarak üreteceğimiz SRAM bloklarının sayısı da algoritmamıza giriş olarak verilmektedir.

Algoritma 1: Karma Model Oluşturma Algoritması

Require: $n \leftarrow \#SRAMBlokSayisi$
  $bithatalari \leftarrow n \times 1024 \times 16 \times P_F$
  $hataliS \leftarrow n \times P_S$
  while $hataliS > 0$ *and* $bithatalari > 0$ **do**
    $block \leftarrow rastgele$
    $hataliBlok \leftarrow SRAMbloklari[blok]$
    $hataliSatirlar \leftarrow (F_{SR_{0..16}})'dansec$
    while $hataliSatirlar > 0$ **do**
      $satir \leftarrow rastgele$
      $sutun \leftarrow rastgele$
      $satirdakiHataSayisi \leftarrow (F_{FR_{0..16}})$
      while $satirdakiHataSayisi > 0$ **do**
        $uzaklik \leftarrow (F_{FDR_{1..15}})'densec$
        $sutun \leftarrow sutun + uzaklik$
        $hataliBlok[satir][sutun] \leftarrow hata$
        $satirdakiHataSayisi \leftarrow satirdakiHataSayisi - 1$
      end while
      $hataliSatirlar \leftarrow hataliSatirlar - 1$
    end while
    $YapaySutunOZnitelikleri \leftarrow SutunOznitelikleri(hataliBlok)$
    $GercekSutunOznitelikleri \leftarrow SutunOznitelikleri(GercekVeri)$
    if $Benzerlik(YapaySutunOZnitelikleri, GercekSutunOznitelikleri) > 80\%$ **then**
      $SRAMbloklari[blok] \leftarrow hataliBlok$
      $hataliS \leftarrow hataliS - 1$
      $bithatalari \leftarrow bithatalari - \#hata(hataliBlok)$
    end if
  end while

Bir Karma Model oluşturmak için Algoritma 1 içinde gösterilen yöntemi takip ediyoruz. Algoritmanın girişi SRAM bloklarının sayısıdır ve çıkış, hata haritası olarak da adlandırılan hatalı SRAM verileridir. İlk olarak Algoritma 1'de hatalı SRAM blokları belirlenir. Daha sonra P_F değeri kullanılarak hatalı hücre sayısı hesaplanır. İlk olarak, tüm SRAM bloklarında hatalı SRAM'leri rastgele seçiyoruz. Ardından, hatalı SRAM'lerde hücrelere karşılık gelen hataları yerleştiriyoruz. Bu yerleştirme algoritması, ince taneli öznitelikler kullanmaktadır. Bu işlem iki aşamada gerçekleştirilir: satır tabanlı hata yerleştirme ve sütun tabanlı kontrol mekanizması. Satır tabanlı hata yerleştirme, satır tabanlı özellikleri kullanır. $P_{SR_{0..1024}}$'ten türetilen SRAM bloğu *yani,* $F_{SR_{0..1024}}$ başına hatalı satır sayısının olasılığını kullanarak kaç satırın hatalı olacağını belirlenir. Ardından, $P_{FR_{0..16}}$'dan türetilen *yani,* $F_{FR_{0..16}}$ satırlarındaki her bit hatasının sayısının olasılığına göre bit hatası yerleştirilir. Bir satırda birden fazla hata yerleştirmek için, $P_{FDR_{1..15}}$'dan türetilen $F_{FDR_{1..15}}$ satırlarındaki ardışık hatalı bit hücreleri arasındaki fiziksel mesafe olasılığını kullanırız. Bu algoritmanın ikinci adımında, ilk olarak, her bir yapay hata haritasının sütun tabanlı öznitelikleri çıkartılır.

Çıkartma işleminden sonra, bu öznitelikleri ikinci adımdan çıkartılan ince taneli sütun tabanlı öznitelikler ile karşılaştırırız. Yapay hatalı bir SRAM'ın deneysel gerçek



Şekil 3.4: Farklı benzerlik eşik seviyeleri için gerçek veriler ile oluşturulan yapay model arasındaki çalışma süresi ek yükü ve doğruluk farkı

verilerle benzerliği %80'den düşükse, benzerlik %80 veya daha yüksek olana kadar bu adımları tekrar uygularız. Çalışma zamanı (hata modellerini oluşturmak için) ve doğruluk (oluşturulan hata modellerinin) arasında iyi bir denge olarak %80 benzerlik eşiğini seçiyoruz. Şekil 3.4'da gösterildiği gibi, eşik seviyesini arttırırsak, MoRS'un çalışma süresi büyük ölçüde artar. Bununla birlikte, eşiği artırmak, optimal eşik ile karşılaştırıldığında önemli bir doğruluk iyileştirmesi sağlamaz. Bu eşiğin altında, yaklaşık model, kabul edilebilir doğruluğa sahip olmayan Rastgele Modele yakınsar.

3.2 Deneysel Yöntembilim

MoRS, düşük gerilimli SRAM blokları için yaklaşık hata haritaları oluşturan genel bir altyapıdır. Bu çalışmada, MoRS'u son teknoloji DSA'lar üzerinde test ediyoruz. Deneylerimiz, hataları eğitilmiş DSA'ların ağırlıklarına yerleştirmeye dayanmaktadır. MoRS'un ne kadar hassas olduğunu değerlendirmek için Caffe [115] aracını kullanıyoruz. Ayrıca, deneylerimizi çeşitlendirmek için farklı nicemlemeler (kesinlik), bit eşlemeleri ve değer maskeleme gerçekleştirmekteyiz. Bu seçeneklerin özeti Çizelge 3.3 içinde bulunmaktadır.

Deney, V_{min} ve V_{crash} arasındaki her voltaj seviyesi için hassasiyet, haritalama ve maskeleme seçeneği ile gerçekleştirilir. Yapay modelleri gerçek verilerle karşılaştırmak için bu metodolojiyi gerçek veriler için de işliyoruz. Yapay Modeller, Rastgele model ve Karma Modeldir. Rastgele Model, hataları yerleştirmek için önceki çalışmalarda kullanılan rastgele hata yerleştirme yapılan bir temeldir. Karma Model, MoRS çıktısı, yaklaşık bir modeldir. Gerçek Veri, önceki deneysel çalışmadan [3] çıkarılan gerçek verilerdir. Gerçek verileri değerlendirmek için, gerçek verilerden rastgele gerekli miktarda SRAM bloğu seçiyoruz.

Metodoloji, Şekil 3.5'de gösterildiği gibi dört bölümden oluşur: ❶ MoRS & Deney, ❷ Bit-Hassasiyeti & Bit-Haritalama Birimi, ❸ Hata Yerleştirme Birimi ve ❹ DSA Sınıflandırma Uygulaması. İlk adımda, Bölüm 3.1'de açıklanan MoRS'dan yapay

Çizelge 3.3: Değerlendirme için Yöntemler

Yöntem	İsim
Hassasiyet	32-bit tek-hassaiyetli kayan virgül
	16-bit yarı-hassaiyetli kayan virgül
	8-bit sabit sayı (Q4.4)
	4-bit sabit sayı (Q2.2)
	1-bit (Binary)
Bit Haritalama	En anlamlı bitten başlama
	En anlamsız bitten başlama
	İlk yarısı en anlamlı, diğer yarısı en anlamsız
	İlk yarısı en anlamsız, diğer yarısı en anlamlı
Değer Maskeleme	Sonsuz veya NaN değerini 1 yapma
	Sonsuz veya NaN değerini 0 yapma

modeller elde ediyoruz. Üçüncü adımda, birinci ve ikinci adımın çıktıları ile hatalı ağırlıklar üretiyoruz. Son adımda, nihai doğruluk yüzdesini elde ediyoruz.

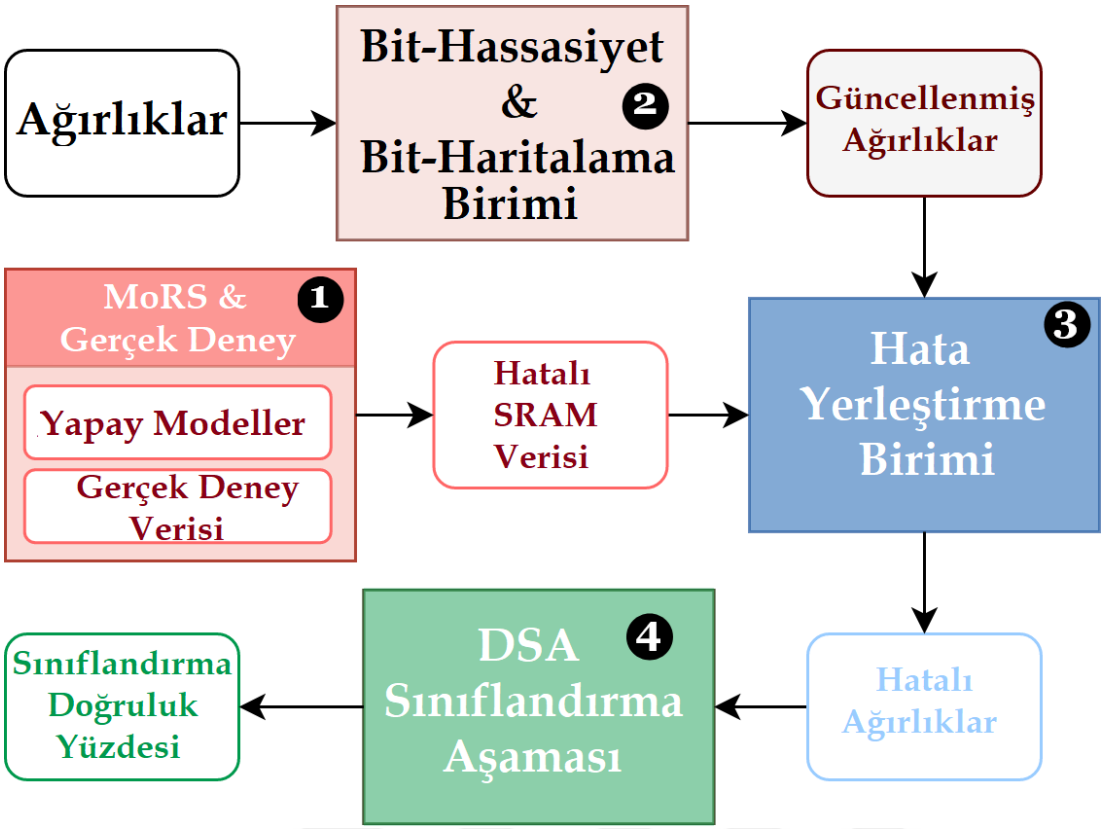
1. MoRS & Deney. Bu adımda, Hata Yerleştirme Birimine hangi SRAM verilerinin gönderileceğini seçiyoruz. MoRS'u değerlendirmek için bir temel model, *yani,* Rastgele Model oluşturuyoruz. Rastgele Model, kaba taneli özelliklerle tek tip rastgele dağılımla üretilmektedir. Bölüm 3.1'de açıklanan MoRS'daki her adımı işliyoruz. Ancak, kaba taneli ve ince taneli özelliklere özel olasılıklı modelleme ve algoritma uygulamak yerine, yalnızca kaba taneli özelliklere doğrudan rastgele bir dağıtım işlevi uyguluyoruz.

Rastgele Modelde öncelikle belirli sayıda SRAM bloğu ile P_S değerini kullanarak hangi bloğun ve kaç bloğun hatalı olduğunu belirliyoruz. Ardından P_F değerini kullanarak kaç hücrenin hatalı olacağını hesaplıyoruz. Hesaplama sonucu, rastgele seçilen hatalı SRAM bloklarının hücrelerine rastgele hatalar yerleştiriyoruz. Düzgün rastgele dağılım nedeniyle, hatalı bloklardaki her hücre aynı olasılığa sahip olmaktadır. Karma Model ve Rastgele Model arasındaki farklar Çizelge 3.2 içinde özetlenmiştir.

Şekil 3.5'de Karma Model ve Rastgele Modeli yapay modeller olarak adlandırmaktayız. Yaklaşık modelimizin ne kadar doğru olduğunu anlamak için Gerçek Veri olarak da adlandırılan deneysel verileri değerlendiriyoruz. Değerlendirilen DSA maksimum 850 SRAM bloğu kullandığından, gerekli miktarda SRAM bloğunu rastgele seçiyoruz.

2. Bit-Hassasiyeti & Bit-Haritalama Birimi. Nicemleme ve düşük voltaj, DSA'ların enerji verimliliğini artırmak için etkili tekniklerdir. Ancak, agresif parametre değişimi ile doğruluk kaybına yol açabilirler. MoRS, optimal bir çalışma noktası bulmak için bu iki parametrenin ilişkini keşfetmemizi sağlar.

Bu adımda hassasiyet ve haritalama seçeneklerine göre ağırlıkların değerini değiştiriyoruz. Caffe aracının nominal ağırlık hassasiyeti 32 bitlik tek-duyarlıklı kayan noktalardır. Ağırlıkların hassasiyetini azalttığımızda bit uzunlukları da azalmaktadır.



Şekil 3.5: Değerlendirme Yöntembilimi

SRAM bloğunun her satırı 16 bit olduğundan, hassasiyet azalmadığında (32-bit) bu ağırlıkları iki satırda saklarız.

Dört farklı hassasiyet seviyesi kullanıyoruz: 16-bit yarı hassasiyetli kayan nokta, 8-bit (Q4.4), 4-bit (Q2.2) ve 1-bit. Sabit hassasiyetli seçenekleri (Q4.4, Q2.2 ve 1-bit) etkinleştirmek için, önceki bir çalışma [116] orijinal Caffe'nin uyarlanmış bir sürümünü ve sınırlı sayısal ağırlık hassasiyetini kullanıyoruz. 16 bitlik yarı hassasiyetli kayan nokta için, NVIDIA tarafından sağlanan Caffe olan NVCaffe [117] kullanıyoruz. Hassasiyet $X - bit$'e düşürüldüğünde, $16/X$ ağırlıklarını art arda bir satırda saklarız. Bu nedenle, hassasiyeti azaltarak, SRAM bloklarının kullanımı ve güç tüketimi azalırken, doğruluktan daha fazla kaybedilir.

Hassas seçeneklere ek olarak, ağırlıkların SRAM bloklarına haritalanmasını değiştiriyoruz. Dört haritalama seçeneği vardır: en anlamlı bitler en solda (MSB), en anlamsız bitler en solda (LSB), sayının ilk yarısı MSB ve sayının ikinci yarısı LSB, sayının ilk yarısı LSB ve diğer yarısı MSB olacak şekilde. MSB, bir satırın ilk hücresine en önemli ağırlık eşlemeleri anlamına gelirken, LSB, bir satırın ilk hücresine en az anlamlı ağırlık eşlemeleri anlamına gelir.

3. Hata Yerleştirme Birimi. Hassasiyet ve haritalama seçeneklerinden sonra güncellenmiş ağırlıkları elde ederiz. Bu adımda, güncellenmiş ağırlıklardaki hataları yerleştirmek için yapay modeller kullanıyoruz. Yapay modelin her bir hücresi ya hatalı ya da doğru değeri içerir. Hatalı hücrede bir miktar ağırlık eşlenirse, değerin ilgili bitleri değişir. Aksi takdirde, değer değişmez. Bu bit çevirme işlemi gerçekleştiğinde, bazen ağırlıkların değeri sonsuz veya NaN olabilir. Bu durumu önlemek için bu

Çizelge 3.4: Değerlendirme için kullanılan DSA'ların özellikleri

DSA Modeli	LeNeT-5 [34]	cuda-convnet [35]
Veri Kümesi	MNIST [92]	CIFAR-10 [93]
Ağırlık Sayısı	430500	89440
Kullanılan SRAM Blok Sayısı	850	180
Sınıflandırma Doğruluk Yüzdesi (%)	99.05%	79.59%

değerleri bir veya sıfır olarak maskeliyoruz. Maskeleme işlemi yalnızca 32 bit tek hassasiyetli ve 16 bit yarı hassasiyetli kayan nokta için gerçekleştirilir. Çünkü sabit nokta, NaN veya sonsuz değeri yakınsayacak herhangi bir mantis veya üs parçasına sahip değildir. Çalışmamızda en büyük sabit nokta gösterimi Q4.4 ve maksimum değeri 15'tir.

4. DSA Sınıflandırma Aşaması. Ağırlıklara hataları yerleştirdikten sonra, sinir ağlarının doğruluğunu ölçmek için Caffe aracını kullanıyoruz. En doğru temel olarak gerçek deney verilerini kabul ediyoruz. Deneylerimizi çeşitlendirmek için her voltaj seviyesi için dört farklı bit eşleme, üç farklı hassasiyet ve iki farklı maskeleme yöntemlerini kullanmaktayız.

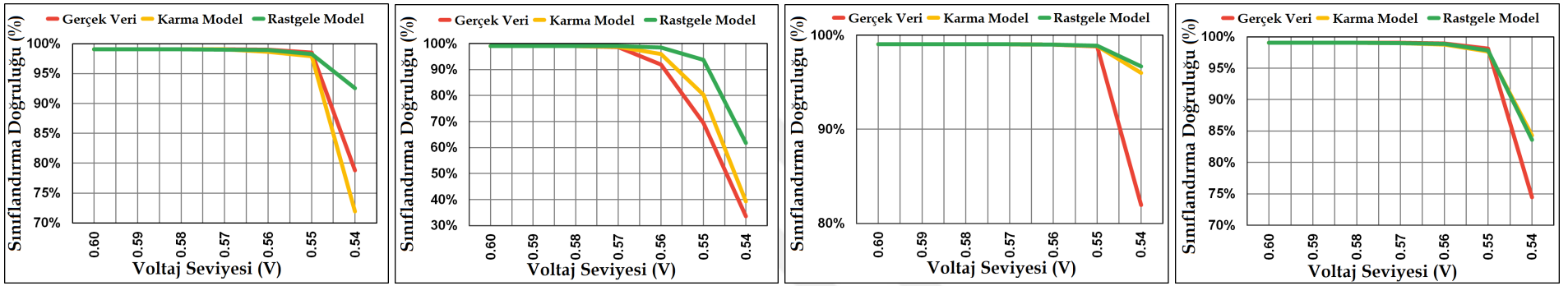
3.3 Sonuçlar

Önceki bölümde bahsettiğimiz gibi, bir derin öğrenme aracı olan Caffe [115] kullanarak iki farklı sinir ağı modelinde MoRS ve rastgele hata yerleştirme modelinin çıktısını test ediyoruz: MNIST veri kümesiyle [92] LeNeT-5 [34] ve CIFAR-10 veri kümesiyle [93] cuda-convnet [35]. Her sinir ağı mimarisi için farklı bit eşlemeleri ve değer maskelemeleri yapıyoruz. Bu testlere ek olarak LeNeT-5 üzerinde azaltılmış hassasiyet testleri de gerçekleştiriyoruz. Değerlendirilen her bir kıyaslamanın ayrıntıları Çizelge 3.4'da özetlenmiştir.

3.3.1 Genel dayanıklılık

Şekil 3.6, farklı bit eşleme ve değer maskeleme seçenekleriyle MNIST veri kümesi [92] üzerindeki LeNeT-5 [34]'in doğruluğunu gösterir. Tüm seçeneklerde Karma Model'in Rastgele Model'den daha hassas ve gerçek verilere daha yakın olduğunu gözlemliyoruz. Ayrıca, uygulama daha az dayanıklı hale gelirse, Karma Model gerçek verilere daha da yakınlaşıp Rastgele Model ile Karma Model arasındaki fark, gerçek verilere ne kadar yakın oldukları açısından artmaktadır.

Sonsuzluğu ve NaN değerini 0'a maskelersek, LeNeT-5 ağının 1'e maskelemekten daha dayanıklı olduğunu gözlemliyoruz. Görüntülerin arka planı 0 ile temsil edildiğinden, sınıflandırma söz konusu olduğunda 1'in etkisi 0'dan daha fazladır.

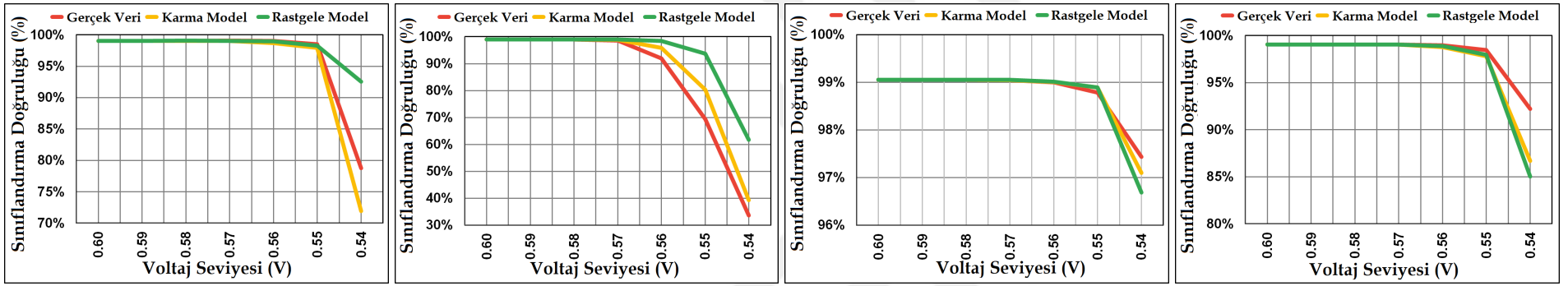


(a) *Sonsuz – NaN değer ← 0*
Bit – Haritalama ← MSB

(b) *Sonsuz – NaN değer ← 1*
Bit – Haritalama ← MSB

(c) *Sonsuz – NaN değer ← 0*
Bit – Haritalama ← LSB

(d) *Sonsuz – NaN değer ← 1*
Bit – Haritalama ← LSB



(e) *Sonsuz – NaN değer ← 0*
Bit – Haritalama ← MSB | LSB

(f) *Sonsuz – NaN değer ← 1*
Bit – Haritalama ← MSB | LSB

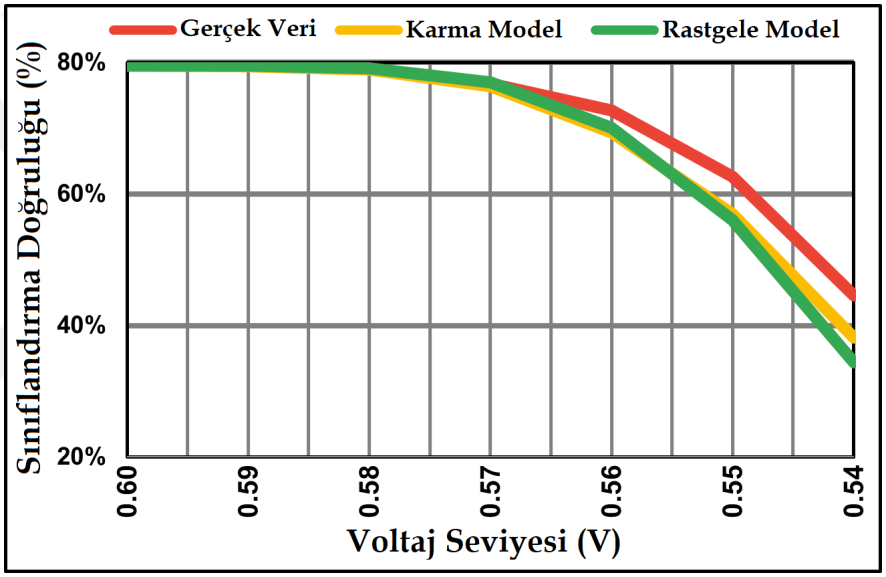
(g) *Sonsuz – NaN değer ← 0*
Bit – Haritalama ← LSB | MSB

(h) *Sonsuz – NaN değer ← 1*
Bit – Haritalama ← LSB | MSB

Şekil 3.6: Her bit haritalama ve maskeleme seçeneği için LeNeT-5 DSA'sı üzerindeki yapay modellerin ve deneysel (gerçek) verilerin bit hassasiyeti azaltılmadığında (32 bit tek hassasiyetli kayan virgül) voltaj seviyelerine göre sınıflandırma doğruluk yüzdesi

MSB ve *MSB* | *LSB* eşlemesinin daha fazla hataya neden olduğunu ve LSB ve *LSB* | *MSB* eşlemesinden daha az dayanıklı olduğunu görüyoruz. Bunun nedeninin, genellikle ilk hücrelerde meydana gelen düşük voltaj tabanlı hatalar olduğunu gözlemledik. MSB, en anlamlı bit anlamına geldiğinden, bit hatası gerçekleştiğinde, karşılık gelen değeri diğer bit basamaklarından daha fazla etkiler. Şekil 3.6b ve Şekil 3.6f daha karakteristik davranış gösterir ve her ikisinin de MSB eşlemeleri ve 1 seçeneğe maskelemesi olduğundan diğerlerinden daha az dayanıklıdır.

Şekil 3.6'de her seçenek için 300 yineleme gerçekleştiriyoruz (değer maskeleme, bit eşleme). Ardından, LeNeT-5 ve cuda-convnet [35] ağ mimarileri için bu seçeneklerin ortalamasını alıyoruz. Bunlara ek olarak LeNeT-5 için farklı hassasiyet seviyelerinde bu deneyleri yapıyoruz ve her bir hassasiyet seviyesi için tüm seçeneklerin ortalamasını alıyoruz.



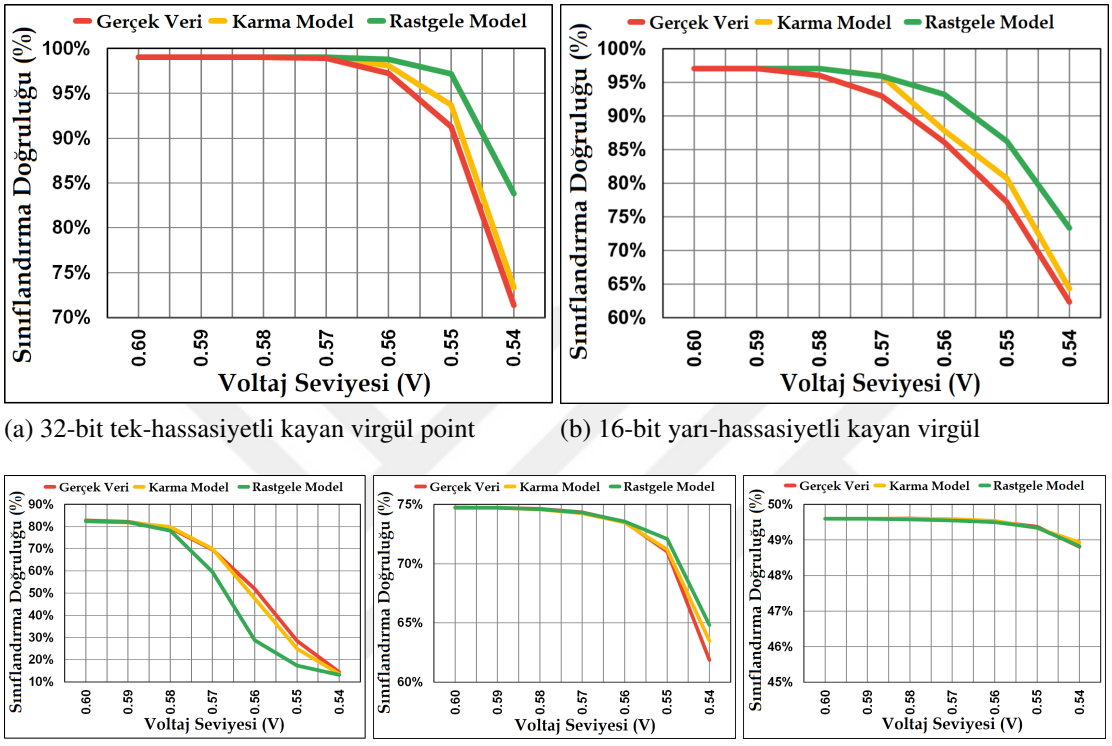
Şekil 3.7: cuda-convnet ağı için yapay modellerin ve deneysel (gerçek) verilerin voltaj ve dayanıklılık davranışına ilişkin ağırlık hassasiyeti düşürmeden tüm parametrelerle testlerinin ortalama sonucu.

Şekil 3.7, CIFAR-10 veri kümesi [93] üzerindeki cuda-convnet'in hassasiyetini azaltmadan doğruluğunu gösterir, *yani* ağırlıkların hassasiyeti nominal 32-bit kayan noktalardır. LeNeT-5 [34] ağını Şekil 3.8a'de değerlendirirken, cuda-convnet [35] ağı için her bit eşleme ve değer maskeleme seçeneğinin ortalamasını alıyoruz.

LeNeT-5'in daha iyi çalışmasının nedeninin, CIFAR-10'daki cuda-convnet'in ağırlık sayısı ve SRAM blok kullanımı açısından oldukça küçük olmasıdır. Ancak bu ağda bile Karma Model, Gerçek Veri ile ortalama doğruluk farkı açısından Rastgele Modelden 1,47 kat daha yakındır. SRAM bloklarının besleme gerilimi azalırken, gerçek verilerden daha karakteristik ve ayırt edici davranış gözlemliyoruz. V_{crash} seviyesinde, Karma Model ile Gerçek Veri arasındaki doğruluk farkı sadece %6 iken Rastgele Model ile Gerçek Veri arasındaki doğruluk farkı %10'dur.

3.3.2 Nicemleme

Şekil 3.8, LeNeT-5'in MNIST veri kümesindeki doğruluğunu farklı hassasiyet seviyelerinde gösterir. Her bir hassasiyet seviyesinde, Karma Model'imizin, Rastgele Model'in gerçek verilere sahip olduğundan daha fazla benzer trende sahip olduğunu görüyoruz. Şekil 3.8a, Şekil 3.6'de gösterilen tüm seçeneklerin ortalamasıdır.



(a) 32-bit tek-hassasiyetli kayan virgül point

(b) 16-bit yarı-hassasiyetli kayan virgül

(c) 8-bit sabit nokta (Q4.4)

(d) 4-bit sabit nokta (Q2.2)

(e) 1-bit (Binary)

Şekil 3.8: LeNeT-5 ağındaki deneysel (gerçek) verilerin ve yapay modellerin her voltaj seviyesindeki tüm seçeneklerinin (maskeleme, bit-haritalama) farklı hassasiyetlerle ortalama olarak sınıflandırma doğruluğuna etkisi

Şekil 3.8a, hem yapay modellerin hem de gerçek verilerin nominal hassasiyet seviyesi (32-bit tek kesinlikli kayan nokta) altında doğruluk açısından dayanıklılığını gösterir. Karma Model, Rastgele Model'den ortalama olarak gerçek verilere 3.74 kat daha yakındır. V_{crash}'da Karma Model ile Gerçek Veri arasındaki doğruluk farkı %2,46, Rastgele Model ile Gerçek Veri arasındaki fark ise %12,47'dir.

Şekil 3.8b, 16-bit yarı hassasiyet kayan noktaya indirgenmiş hassasiyetle ağırlıklar için LeNeT'nin doğruluğunu gösterir. Tüm modellerin davranışı 32-bit mimariye benzer çünkü 16-bit hassasiyet, doğruluk üzerinde önemli bir etkisi olmaksızın çoğunlukla 32-bit [118, 119] içindeki tüm değerleri kapsar. Gerçek veriler ile Karma Model arasındaki en yüksek doğruluk farkı 0,55V'de %4,47, gerçek veriler ile Rastgele Model arasındaki en yüksek doğruluk farkı ise 0,57V'de %9,05'tir. Gerçek veriler ile Karma Model arasındaki tüm voltaj seviyelerinin ortalama doğruluk farkı %2,25, gerçek veriler ile Rastgele Model arasındaki ortalama doğruluk farkı %4,84'tür ve bu, gerçek verilere sınıflandırma doğruluğu farkı açısından Karma Model'den 2,15 kat daha kötüdür.

Şekil 3.8c, ağırlıklar için 8 bit hassasiyetle LeNeT'nin doğruluğunu gösterir. 32-bit ile karşılaştırıldığında, ağın düşük voltajlı hatalara karşı daha az dayanıklı olduğunu görüyoruz. Ortalama olarak, Karma Model, gerçek verilere yaklaşma açısından Rastgele Modelden 7 kat daha iyidir. 560mV'da Karma Model ile Gerçek Veri arasındaki doğruluk farkı %4 iken Rastgele Model ile Gerçek Veri arasındaki fark %23'tür.

Figure 3.8d, ağırlıkların hassasiyeti 4 bit'e düştüğünde hem yapay modellerin hem de gerçek verilerin doğruluğunu gösterir. Ağırlıkların hassasiyetini 8 bit ila 4 bit arasında azaltarak ağların voltaj alt ölçeklendirmesine dayalı hatalara karşı daha fazla hataya dayanıklı hale geldiğini görüyoruz. V_{nom}'da 8-bit LeNeT, 4-bit LeNeT'den daha yüksek doğruluk yüzdesine sahip olmasına rağmen, V_{crash}'da 8-bit doğruluk yüzdesi %14 iken 4-bit doğruluk yüzdesi %62'dir. Bu beklenmedik durumda bile, MoRS altyapısının yapay modellemesi gerçek verilere benzer davranışa sahiptir. Ortalama olarak, Karma Model, Rastgele Model'den gerçek verilere 2 kat daha yakındır. 550mV'da, Karma Model ile gerçek veriler arasındaki doğruluk farkı %1.5 civarındayken, Rastgele Model ile gerçek veriler arasındaki doğruluk farkı %3'dir.

Şekil 3.8e, hassasiyet 1 bit'e düştüğünde LeNeT'nin doğruluğunu gösterir. 1 bitlik testlerde ağırlıkları üç farklı değer kümesine eşliyoruz. Birincisi {-1,1}, ikincisi {-1,0} ve sonuncusu {0,1}. Ancak, değer kümeleri arasında çok fazla fark gözlemlenmemiştir. 4-bit'te gördüğümüz gibi, 1-bit LeNeT ağı hatalara karşı daha dirençli hale gelir. V_{nom} doğruluğu ile V_{crash} doğruluğu arasındaki fark %0,79'dur. Her iki yapay model de aynı davranışa sahiptir ve doğruluk açısından önemli bir farklılığa sahip değildir. Karma Model, doğrulukta %0.03 farka sahipken, Rastgele Model %0.02'dir. Bu ihmal edilebilir istatistikler nedeniyle, Şekil 3.9'ya 1 bitlik LeNeT eklemiyoruz.

Ağırlıklarının hassasiyeti azaltılmış ağların dayanıklılığına dikkat çekmek için, V_{nom} ve V_{crash} arasındaki doğruluk düşüşünü incelediğimizde 8-bit, 4-bit ve 1-bit LeNet için düşüş sırasıyla %68,4, %12.9 ve %0,79 olduğunu gözlemliyoruz.

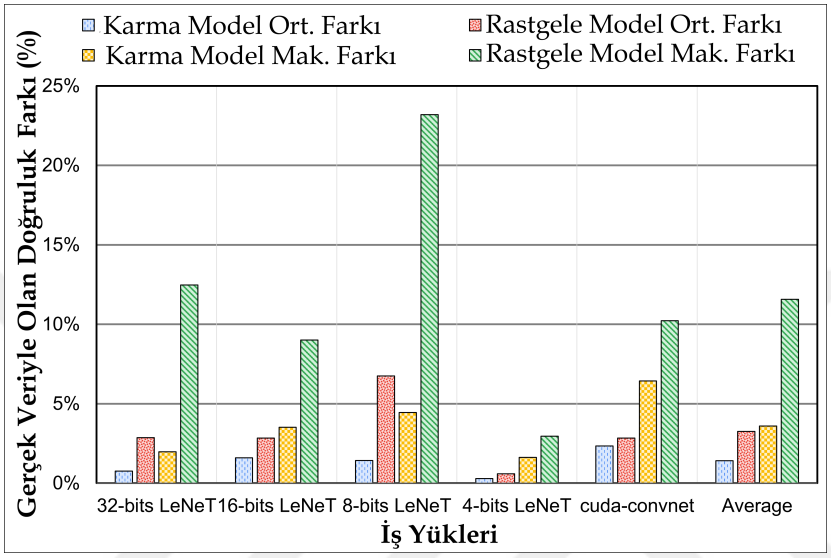
Çizelge 3.5: Farklı Hassasiyet faktörlerinde LeNeT-5 için SRAM Block Kullanımı ve Sınıflandırma Doğruluk Yüzdesi (Voltaj V_{nom} iken)

LeNeT-5 Hassasiyeti	**SRAM Kullanımı**	**Sınıflandırma Doğruluk Yüzdesi**
32-bit kayan virgül	850	%99,05
16-bit Yarı kayan virgül	425	%97,03
8-bit (Q4.4)	213	%82,79
4-bit (Q2.2)	107	%74.75
1-bit (Binary)	27	%49,59

Çizelge 3.5, çeşitli hassasiyete seviyelerindeki ağırlıklara sahip DSA'ların SRAM kullanımını ve doğruluğunu gösterir. Sınıflandırma doğruluğu, nominal voltaj seviyesindeki (V_{nom}) doğruluğu temsil eder. 32-bit kayan virgül, 32-bit tek duyarlıklı kayan noktayı ve 16-bit Half kayan virgül, 16-bit yarı-duyarlı kayan noktaları ifade eder.

3.4 Yapay Modellerin Karşılaştırılması

Şekil 3.9, Gerçek Veri ile iki yapay model arasındaki sınıflandırma doğruluk yüzdesindeki farkı gösterir. Tüm iş yüklerinin ortalamasında Karma Model'in Rastgele Model'e göre gerçek deneye ortalama 3,21x daha yakın olduğunu görüyoruz. Kıyaslamaların çoğu için, Karma Model ve Gerçek deney arasındaki maksimum doğruluk farkı %5'in altındadır. Ancak Rastgele Model için doğruluktaki maksimum fark %23.2'dir.



Şekil 3.9: Tüm iş yükleri için gerçek veri ve her yapay model arasındaki doğruluk farkı.

Önerilen modelin düşük gerilim etkilerine karşı gerçek verilerle aynı davranışa sahip olmasının yanı sıra doğruluk açısından tolere edilebilir bir farkla gerçek verileri taklit edebileceği sonucuna varılmıştır. En önemlisi, sistem hatalara karşı dayanıklı değilse, rastgele hata yerleştirme mekanizmaları gerçek verilerin davranışını göstermez. Düşük voltajın sistemleri nasıl etkilediği konusunda değerlendirme yapmak için, gerçek verilerin ince taneli olarak profilinin çıkarılması gerekir. Düşük gerilim gerçekleştirildiğinde gerçek SRAM davranışını modellemek için kaba taneli özellikler (Rastgele Model) yetersizdir.

3.5 İlgili Çalışmalar

Bildiğimiz kadarıyla, bu çalışma ilk yaklaşık hata modelleme altyapısını ve düşük gerilimli SRAM'lerde gerçeğe yakın hata yerleştirilmesini sağlar. Bu bölümde, düşük voltajlı sistemlerde hata yerleştirme ve modelleme ve DSA'ların dayanıklılığı ile ilgili çalışmaları tartışıyoruz.

DSA'ların Dayanıklılığı. DSA'lar, doğası gereği hatalara karşı bir noktaya kadar dayanıklıdır. Ancak zorlu ortamlarda, işlem farklılıkları ve düşük voltaj ölçekleme önemli doğruluk kayıplarına neden olabilir. Reagen ve diğerleri [16], Minerva'yı DNN hızlandırıcılarında düşük voltajlı SRAM hatalarının etkilerini azaltmak için bir hata azaltma mekanizması olarak önermektedir. Salami ve diğerleri [3], SRAM

tabanlı yonga üzerindeki bellekleri düşük voltajla çalıştıran ve DSA sınıflandırma katmanındaki düşük voltaj hatalarını azaltmak için akıllıca sınırlandırılmış BRAM yerleşimi sunan bir çalışma gerçekleştirmiştir. Torres-Huitzil ve diğerleri [29], sinir ağlarında hata ve hata toleransı hakkında kapsamlı bir inceleme sunar ve hata azaltma ve yerleştirme tekniklerini tartışır. Ayrıca, hataların sinir ağları üzerindeki etkilerini çözülmemiş bir sorun olduğunu derinlemesine anlamak için daha gerçekçi/yeni hata modellerinin geliştirilmesi gerektiğini belirtmişlerdir. Deng ve diğerleri [30], sinir ağlarının daha dayanıklı hale gelmesi için yeniden eğitime dayalı bir hata azaltma tekniği sunar.

Hata Yerleştirme. Hata yerleştirme, dayanıklılık çalışmalarında yaygın olarak kullanılan bir tekniktir. Ayrıca, hata yerleştirme, bit değerlerini tersine çevirme saldırıları [120, 121, 122] ve hata yerleştirme saldırıları [123, 124] olarak da adlandırılan bit çevirme tabanlı rakip ağırlıklı saldırılar olarak kullanılır. Pek çok çalışma, yumuşak hatalar, gürültü [125, 126, 114, 127] ve voltaj düşük ölçeklendirme konularında sistemlerin güvenilirliğine ve esnekliğine hataları yerleştirme çalışması yaparak ederek odaklanmaktadır. Koppula ve diğerleri [33], karakterize edilmiş DRAM hataları üzerine zamanlama ihlalleri ve voltaj alt ölçeklendirmesi ile hata azaltma stratejileri ve ağırlık eşlemeleri açısından eğitimi birleştirmeyi öneren EDEN adlı bir altyapı önermektedir. EDEN, tek tip rastgele dağılım tüm DRAM'leri kapsamadığından dört farklı hata modeli sunar. Chatzidimitrou ve diğerleri [89, 31], düşük voltaj ölçeklemenin etkilerini incelemek için hataları dallanma öngörücü birimlerine rastgele enjekte eder. Chandramoorthy ve diğerleri [17], SRAM'lere rastgele hatalar yerleştirerek ederek farklı ağ katmanlarındaki düşük voltajdan oluşan hataları inceler ve bit hatalarının kalıplarını veya uzamsal dağılımını dikkate almaz. Stutz ve diğerleri [128], varsayılan voltaj düşük ölçekli SRAM arızaları rastgele dağıldığını savunur ve daha dayanıklı bir DSA sınıflandırması için rastgele bit hatalarını geri dönüşümlü kullanarak DSA eğitimi önermektedir. Salami ve diğerleri [32], DSA hızlandırıcılarının dayanıklılığını , hata karakterizasyonu ve hatanın nasıl engelleneceğini inceler. Yang ve diğerleri [23], SRAM'lerde voltaj ölçeklendirmesi gerçekleştirerek enerji açısından verimli DSA'ların çalışmasını ve doğruluğa etkisini inceler. Bit hatalarının etkisini incelemek için SRAM'deki hataların yüzeysel bir şekilde eşit olarak dağıldığını varsaymışlardır. Eşiğe yakın Gerilim FinFET SRAM'leri üzerine önceki bir çalışma [129], tek tip rastgele dağılıma dayalı olarak SRAM'ler için bir hata modeli sunar. Givaki ve diğerleri [130], DSA'ların eğitim aşamasını incelemek için doğrudan deneysel verileri kullanarak düşük voltajlı SRAM tabanlı FPGA yonga üstü bellekler altında DSA'ların direncini inceler.

Tüm bu düşük voltajlı yonga üzerinde hata yerleştirme çalışmaları, yerleştirmeyi rastgele gerçekleştirir ve hücreler arasındaki uzamsal mesafeler, satır tabanlı ve sütun tabanlı yaklaşımlar gibi ince taneli öznitelikleri hesaba katmaz. Rastgele yerleştirilen hatalar, sistemin düşük voltaj alanı altında nasıl çalıştığını anlamak için yanıltıcı olabilir. Bölüm 3.3'te bahsettiğimiz gibi, sistem voltaj düşüklüğüne karşı çok fazla dayanıklılığa sahip değilse, MoRS'un Karma Modeli, gerçek verilere, rastgele yerleştirilen modelden 7 kat daha yakındır.

4. TuRaN

4.1 Motivasyon

Gerçek rastgele sayı üreteçleri (GRSÜ'ler), çeşitli modern güvenlik açısından kritik sistemlerin, özellikle güvenli ve özel iletişimleri başlatmak için oturum ve geçici anahtar oluşturma gibi kriptografik uygulamaların, güvenli sunucuların, VPN erişiminin ve kimlik doğrulama tabanlı uygulamaların vazgeçilmez parçalarıdır [131, 132, 37, 133]. Bu uygulamalar güvenliklerini rastgele sayıların kararlılığı ve tahmin edilemezliğine dayandırır. Dışarıdan yapılan bir saldırı nedeniyle cihazların RSÜ kısmında bir hata, tüm sistemin güvenliği büyük ölçüde azaltılabilir. Önceki çalışmalar [134, 135], düşük kaliteli RSÜ'lere sahip sistemlerin RSÜ saldırılarından önemli ölçüde etkilenebileceğini göstermektedir. Bu nedenle, sistemlerin güvenliğini sağlamak için donanım saldırılarına karşı bir önlem olarak yüksek kaliteli RSÜ'ler gereklidir.

Yüksek aktarım hızına ve düşük gecikme süresine sahip GRSÜ'ler, modern ticari cihazlar, özellikle güvenli veri odaklı sistemler için bir zorunluluk haline gelmektedir [136, 137, 138]. Bu sistemler, performanslarını düşürmeden güvenli işlemlerini sürdürebilmek için genellikle özel GRSÜ donanımıyla donatılmıştır. Daha önceki birçok çalışma, bu tür sistemler için halka osilatör tabanlı [112, 139], kaos tabanlı [140, 141, 142] ve gecikme zinciri tabanlı [143, 144] dahil olmak üzere birçok farklı donanım tabanlı GRSÜ'ler önermektedir. Bununla birlikte, bu donanım tabanlı GRSÜ'lerin iki ana kısıtlamaları vardır: (i) ek ve yüksek karmaşıklıkta donanıma ihtiyaç duydukları için ticari sistemler için uygun değildirler veya (ii) düşük gecikme süresinde yüksek hızda rasgele sayılar sağlayamazlar. Bu sorunları gidermek için birkaç bellek tabanlı (örneğin DRAM'ler, SRAM'ler, NVM'ler) GRSÜ'ler önerilmiştir [145, 146, 147, 60, 55, 63].

SRAM'ın diğer bellek aygıtlarına göre iki büyük avantajı vardır: (i) üretilen rastgele bitleri CPU'ya göndermek için yonga dışı bir aktarım gerektirmediğinden daha güvenlidir ve (ii) SRAM, her CMOS tabanlı sistemde kullanılır ve birçok cihazda gerçek rastgele sayıları etkinleştirebilir (örneğin, EMV kartları [148], büyük ölçekli sistemler [149]). Bu nedenle, SRAM tabanlı GRSÜ'ler, modern yaygın sistemler için gerçek rastgele sayı uygulamalarını etkinleştirmek için bir çözüm sunar.

Önceki SRAM tabanlı GRSÜ'ler [56, 57, 65, 66, 67, 58, 68, 55, 69, 70, 59] yalnızca SRAM hücrelerindeki başlangıç değerlerini kullanır. Açılış durumunda, bir SRAM hücresinin başlangıç değeri, süreç varyasyonu (*-ing.* process variation) nedeniyle farklılık gösterebilir. Önceki çalışma [71], tüm SRAM hücrelerinin %5-15'inin kısmen başlangıç değerlerinin değişken olduğunu ve bunların %5'inden azının yüksek rastgelelik sergilediğini göstermektedir. Rastgele davranan bu hücreler, gerçek rastgele sayı üretiminde entropi kaynağı olarak kullanılır.

Bu çalışmalar, uygulanabilir GRSÜ mekanizmaları önermelerine rağmen pahalı aç-kapa döngülerine bağlı oldukları ve SRAM hücrelerinde düşük entropiye sahip oldukları için, gerçek sistem entegrasyonu için pratik olmayan dört ana zayıflıktan muzdariptirler: mevcut SRAM tabanlı GRSÜ'ler (i) sürekli bir şekilde gerçek rasgele sayılar üretemez, (ii) açılış döngüsü süresi nedeniyle yüksek gecikmeye maruz kalır (ör. $\sim$250 ms [57]), (iii) yüksek aktarım hızı ile gerçek rastgele sayılar üretemez [71] ve (iv) düşük güçlü enerji verimli cihazlar için yüksek enerji tüketir.

Önceki çalışmaların analizine dayanarak, SRAM tabanlı bir GRSÜ'nün aşağıdaki özellikleri karşılaması gerektiğini çıkarıyoruz:

- Yüksek performanslı sistemler için düşük gecikme süresinde yüksek aktarım hızında sürekli ve tutarlı bir şekilde gerçek rasgele sayılar üretmelidir.
- Enerji açısından verimli cihazlarda gerçek rastgele sayılar üretmek için düşük enerji tüketmesi gerekir.
- Düşük güçlü uç cihazlardan yüksek performanslı büyük ölçekli sistemlere kadar ticari cihazlarda uygulamak pratik olmalıdır.

Bu çalışmadaki amacımız, yaygın olarak bulunan modern cihazlarda gerçekten rastgele sayılar üretmek için yukarıdaki tüm özellikleri karşılayan SRAM tabanlı bir GRSÜ tasarlamaktır.

4.2 Düşük-Gerilimli SRAM'lerin Rastgelelik Karakterizasyonu

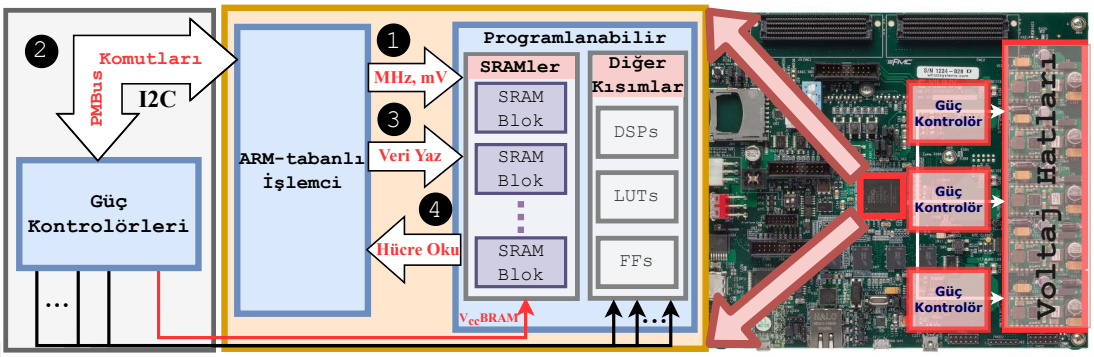
SRAM hücrelerinin rastgelelik davranışını farklı çalışma voltajı, frekans, sıcaklık seviyelerinde ve veri örüntüsü üzerinde deneysel olarak inceliyoruz. FPGA kartlarında SRAM tabanlı yonga üstü bellekler üzerinde deneyler yapıyoruz. Bu platform, *(i)* SRAM bloklarının besleme voltajını ayrı ayrı ayarlamak da dahil olmak üzere voltaj hatlarını (düşük voltaj için) manipüle etmemizi, *(ii)* farklı frekans seviyelerinde çalışma esnekliğine sahip olmamızı ve *(iii)* çok sayıda SRAM bloğu üzerinde deneyler yapmamızı sağlar. Süreç varyasyonunun etkisini incelemek ve kapsamlı bir karakterizasyon çalışması yürütmek için deneylerimizi iki özdeş FPGA kartı üzerinde gerçekleştiriyoruz.

4.2.1 Karakterizasyon yöntembilimi

Deneylerimizi, 28nm teknoloji düğümünde üretilen Xilinx Zynq ZC702 FPGA kartlarının (XC7Z020-CLG484-1) [72] iki özdeş numunesi üzerinde gerçekleştiriyoruz. Bu kartlar, ayrı voltaj hatları aracılığıyla SRAM bloklarının bağımsız voltaj ölçeklendirmesini sağlar. Her FPGA kartında 560 SRAM bloğu bulunur ve her SRAM bloğu toplam 16Kbit'lik 1024 satır ve 16 sütundan oluşur. Üretici [150] tarafından ayarlanan SRAM bloklarının nominal besleme voltajı $1V$'dır.

Gerilim alt ölçeklendirme gerçekleştirebilmek adına, gerilim hatlarını işlemek için Güç Yönetim Veri Yolu standartını (PMBus) [151] kullanıyoruz. Bu hatlar PMBus

kullanılarak tamamen manipüle edilebilir ve gözlemlenebilir. I^2C arabirimi aracılığıyla PMBus'u yapılandırmak için bir işletim sistemi kullanıyoruz. Çalışma sıcaklığını, ilgili voltaj rayının akımını ve güç tüketimini izlemek için aynı arayüzü kullanıyoruz. Bu çalışmada, SRAM tabanlı yonga üstü belleklerin besleme voltajı olan V_{CCBRAM}'a odaklanıyoruz [152]. V_{CCBRAM}'ı nominal voltajdan minimum çalışma voltajına, $535mV$'a (deneysel olarak belirlenen) kadar düşürüyoruz.



Şekil 4.1: SRAM'lerde Düşük Voltajı temel alan Genel Rastgelelik Karakterizasyon Metodolojisi

Şekil 4.1, SRAM bloklarındaki rastgeleliği karakterize etmek için kullandığımız genel voltaj alt ölçeklendirme metodolojisini gösterir. Bu metodoloji sadece bizim platformumuzla sınırlı kalmayıp aynı zamanda ZC702 kartındaki gibi aynı bağımsız voltaj rayına sahip olması koşuluyla, diğer FPGA tabanlı platformlara da kolayca genişletilebilir. Önceki çalışmaların önerdiği gibi, yalnızca test ettiğimiz SRAM cihazlarının değil, tüm SRAM cihazlarının düşük voltaj verildiğinde rastgele davranışı göstermesini bekliyoruz [153, 154, 155]. SRAM'lerde düşük gerilime dayalı hataların

Algoritma 2: SRAM'larda Voltaj Alt Ölçeklendirme Tekniğiyle Rastgelelik Testi

Require: $voltaj, frekans, veri\_oruntusu$
1: $ayarla\_frekans(frekans)$
2: $ayarla\_voltaj(voltaj)$
3: *her satira uygula* $satira\_yaz(veri\_oruntusu)$
4: **for all** $satirlar \in tumSRAMBloklari$ **do**
5: **while** $tekrarla < 1000$ **do**
6: $deger_{satir} \leftarrow oku\_satir(satir)$
7: $kaydet(deger_{satir})$
8: **end while**
9: **end for**

rastgelelik davranışını incelemek için Algoritma 2'da açıklanan genel karakterizasyon metodolojisini takip ediyoruz. Algoritma 2 ve Şekil 4.1, ❶'de gösterildiği gibi, önce çalışma frekansını (Satır 1) ayarlıyoruz ve ❷ azaltıyoruz SRAM'lerin besleme gerilimi (Satır 2). Gerilimi azaltmak için, işletim sisteminden gerilim denetleyicisine karşılık gelen PMBus komutlarını gönderiyoruz. SRAM bloklarının çalışma frekansını ve besleme voltajını ayarladıktan sonra, ❸ ilgili veri örüntüsünü her satıra yazıyoruz (Satır 3). Verileri yazdıktan sonra, ❹ her satırı 1000 kez okuyup (Satır 6) tüm 1000

bitlik bit akışlarını her satır için yonda dışı belleğe *(yani, DDR)* kaydediyoruz (Satır 7). Ardından, FPGA'nın işletim sistemi tarafında her satırın entropisini ölçüyoruz.

Düşük voltajlı SRAM satırlarında rastgeleliği değerlendirmek için Shannon Entropi [156] mekanizmasını kullanıyoruz.

$$H(x) = -\sum_{i=0}^{1} P(x_i) \log_2 P(x_i) \tag{4.1}$$

Shannon Entropisi, Denklem 4.1 ile hesaplanır. $H(x)$, ilgili hücrenin Shannon Entropisini, $P(x_0)$, mantık-0 değerinin olasılığını ve $P(x_1)$, mantık-1 değerinin olasılığını belirtir. Her hücrenin entropisini ölçüyoruz ve her satır için toplamlarını hesaplıyoruz. Bu metodoloji her iki FPGA kartı için de gerçekleştirilir.

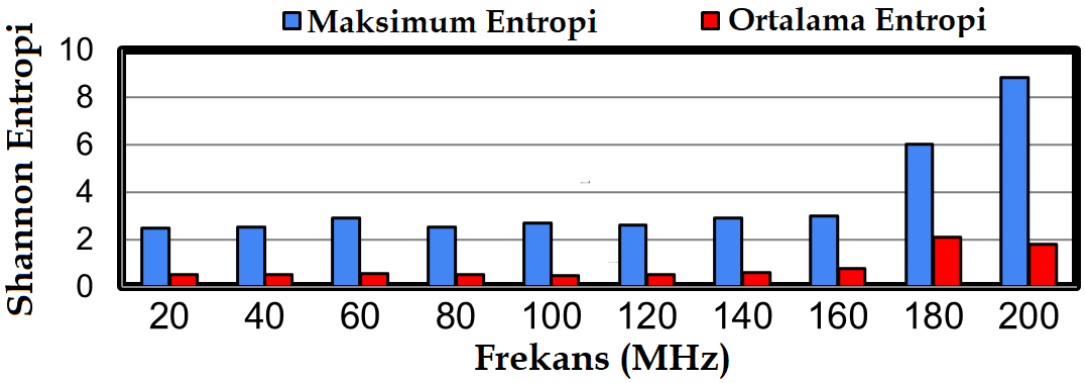
SRAM satırlarının entropisini dört farklı parametre altında inceliyoruz. İlk olarak, SRAM yongalarının çalışma frekansını inceliyoruz. İkinci olarak, SRAM yongalarının besleme voltajının entropi üzerindeki etkilerini inceliyoruz. Üçüncüsü, SRAM satırlarının entropisi üzerindeki veri örüntüsü etkilerini inceliyoruz. Bu üç testi nominal çalışma sıcaklığında gerçekleştiriyoruz. Ham verileri topladıktan sonra, besleme gerilimi ile çalışma frekansı arasındaki ilişkiyi entropi açısından analiz etmekteyiz. Dördüncü olarak ise, SRAM'in rastgeleliğinin sıcaklıkla olan değişimini araştırıyoruz. Her parametrede her 32 bitlik blok için ortalama ve maksimum entropi sunuyoruz. Maksimum entropi, bir FPGA'daki SRAM tabanlı yonga üstü belleğin (SRAM) 32 bitlik bir bloğun en yüksek entropisidir. Bir FPGA'daki tüm SRAM dizilerindeki tüm 32 bitlik blokların ortalama entropisi olarak ortalama entropi terimini kullanırız.

4.2.2 Frekans

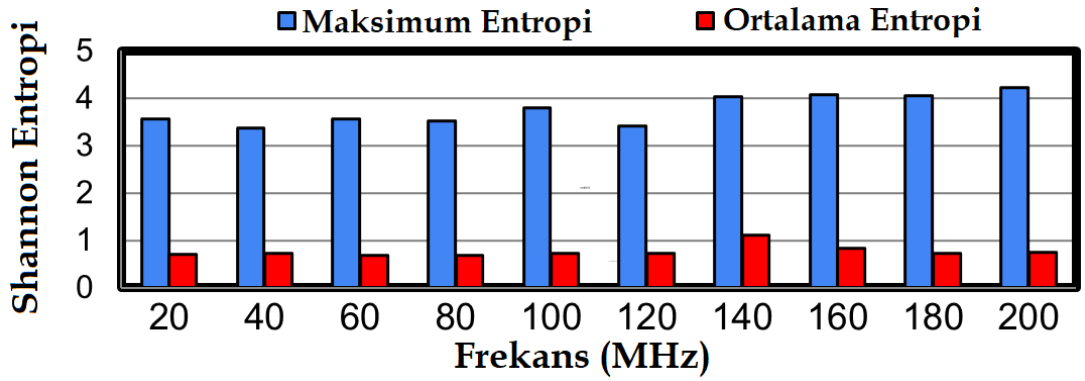
Hangi frekans seviyelerinin daha yüksek entropi ürettiğini ve ortalama ve maksimum entropiyi etkilediğini analiz etmek için, frekansı 20MHz'den 200MHz'e değiştiriyoruz, burada SRAM bloklarının besleme gerilimi minimum çalışma gerilimine, yani $535mV$'a düşürülmüştür. Önceki çalışma [3], 0xFFFF (hepsi mantık-1) veri örüntüsünün en yüksek hata oranına sahip olduğunu bildirmektedir. Bu nedenle, frekansın etkilerini incelemek için bu örüntüyü kullanıyoruz. Daha sonra 4.2.4 Bölümünde farklı veri örüntülerini test ediyoruz ve önceki çalışmayla benzer sonuçları gözlemliyoruz.

Şekil 4.2, farklı frekans seviyelerindeki SRAM'lerdeki her 32 bitlik blok için ortalama ve maksimum entropiyi gösterir. Kart-A ve Kart-B arasında, süreç varyasyonundan kaynaklanabilecek bazı farklılıklar vardır. Ancak, her iki kartın en yüksek maksimum ve ortalama entropisi aynı frekans seviyesinde elde edilir.

Farklı frekans seviyelerinde çalışmanın hem maksimum hem de ortalama entropiyi etkilediğini bulduk. Her iki kartta da 200MHz'de en yüksek maksimum entropiyi elde ediyoruz. 200MHz, Kart-A'da 20MHz'den $3.56x$ daha yüksek maksimum entropiye sahiptir (Şekil 4.2a). Kart-B'de (Şekil 4.2b), 200MHz'deki maksimum entropi, 20MHz'dekinden $1,2x$ kat daha yüksektir. Bu deney sonucunda 1) çalışma frekansını artırmanın daha yüksek maksimum entropi sağladığını ve 2) test edilen her



(a) Kart-A



(b) Kart-B

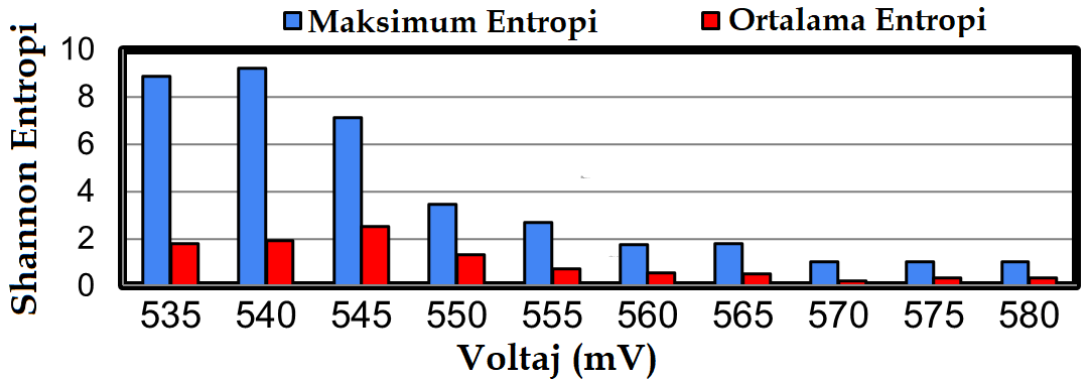
Şekil 4.2: Her SRAM yongasında farklı frekans seviyeleri için maksimum ve ortalama 32 bit blok entropisi.

frekans seviyesinde, düşük voltajlı SRAM'lerin rastgelelik gösterdiğini (yani entropi ürettiğini) gözlemliyoruz.

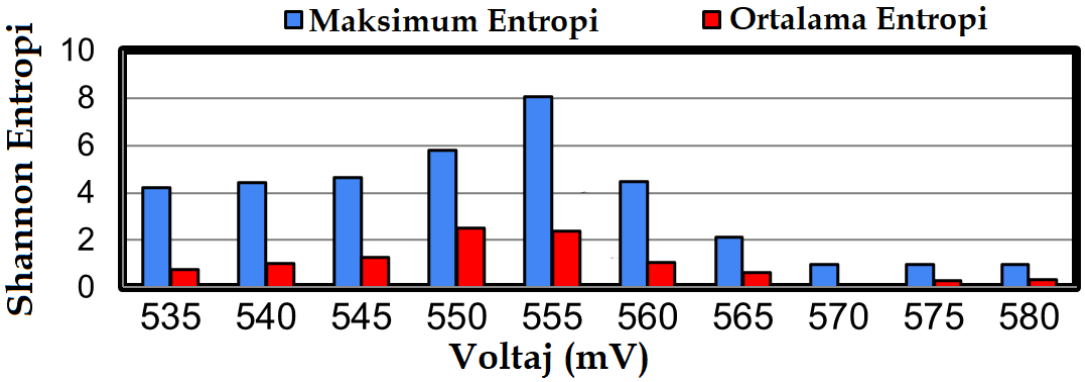
4.2.3 Gerilim seviyesi

Frekanstan sonra, SRAM bloklarının besleme voltajının rastgeleliği nasıl etkilediğini ve hangi voltaj seviyesinin daha yüksek entropi ürettiğini inceliyoruz. Önce minimum çalışma voltajı seviyesinden, $535mV$'dan başlıyoruz, ardından maksimum entropi 1'den küçük olana kadar her adımda voltaj seviyesini $5mV$ artırıyoruz. Frekans testlerindeki aynı veri örüntüsünü kullanıyoruz, 0xFFFF. Her iki kartta da en yüksek maksimum entropiyi 200MHz'de gözlemlediğimiz için çalışma frekansı olarak onu seçiyoruz.

Şekil 4.3, farklı voltaj seviyeleri için ortalama ve maksimum 32-bit blok entropisini gösterir. Şekil 4.3a'da, Kart-A $540mV$'da, minimum çalışma voltajından $5mV$ daha yüksekte, hem en yüksek maksimum hem de ortalama entropiye sırasıyla 9,2 ve 2,52 sahip olduğunu gözlemliyoruz. Bu entropi, minimum çalışma voltajındaki maksimum entropiden %5 daha fazladır. Şekil 4.3b, Kart-B'nin 200MHz'deki voltaj davranışını gösterir. Kart-A'ya benzer şekilde Kart-B, $555mV$'da, minimum çalışma voltajının üzerinde, en yüksek maksimum entropiye, 8.04'e ulaşır.



(a) Kart-A



(b) Kart-B

Şekil 4.3: Her SRAM yongasındaki farklı voltaj seviyeleri için 32 bitlik bloğun maksimum ve ortalama entropisi.

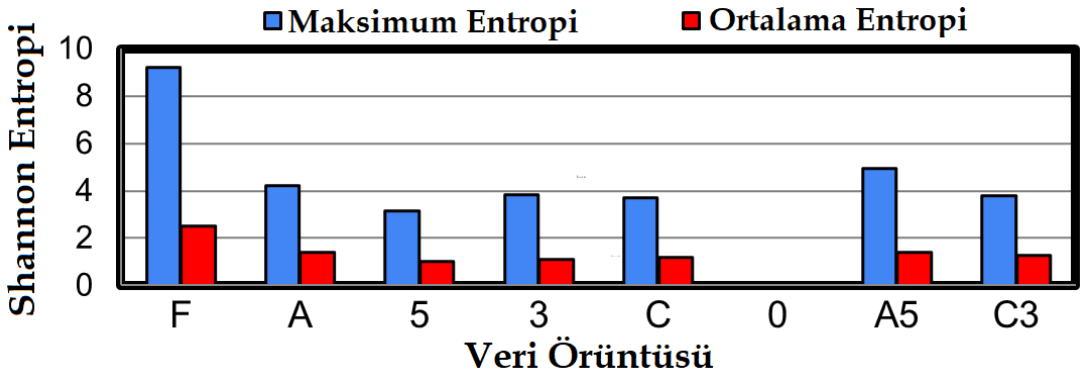
Maksimum entropi, her iki kart için minimum çalışma voltajı seviyesinin üzerinde en yüksek değerine ulaşır. Besleme gerilimi mümkün olduğunca düşük olarak ayarlandığında (örneğin 535 mV), düşük gerilime dayalı hataları her zaman gözlemlediğimiz için deterministik hale geldiğini varsayıyoruz (%100 olasılık). Buna göre, biraz daha yüksek voltaj seviyelerinin ayarlanması %100 olasılıkla başarısız olan hücrelerin sayısını azaltır ve %50 okuma hatası oranı sergileyen SRAM hücrelerinin sayısını artırır, bu da daha yüksek entropi ile sonuçlanır. Minimum voltaj seviyesinin üzerinde, daha yüksek maksimum entropi gözlemlediğimiz için farklı frekans seviyeleri altında voltaj testlerini tekrarlıyoruz ve maksimum ve ortalama entropi gözlemlerini daha sonra Bölüm 4.2.5'de gösteriyoruz.

4.2.4 Veri örüntüsü

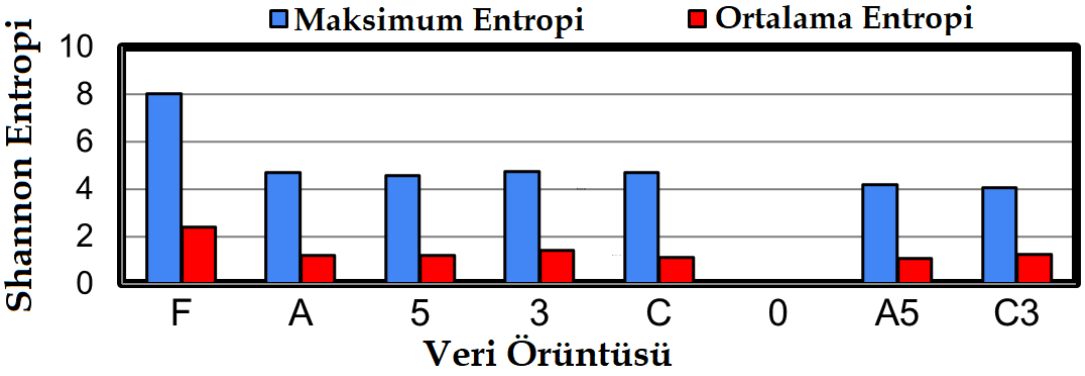
Veri örüntülerinin entropi üzerindeki etkilerini incelemek için sekiz farklı veri örüntüsünü test ediyoruz. Voltaj ve frekansı her iki kart için de maksimum entropinin en yüksek olduğu seviyeye ayarlıyoruz. En yüksek maksimum entropiyi elde eden voltaj seviyesi her kart için farklı olduğundan, frekansı her iki kart için 200MHz olarak belirlendi ve besleme voltajı Kart-A için $540mV$ ve Kart-B için $555mV$'a düşürülmüştür.

Veri örüntülerinin entropi üzerindeki etkisini iki yaklaşımla analiz ediyoruz. İlk

olarak, geleneksel yöntemde, değerler aynı veri örüntüsü ile her satıra yazılır. Bu ilk yöntem için altı farklı veri modeli kullanıyoruz; 0xFFFF, 0xAAAA, 0x5555, 0x0000, 0x3333 ve 0xCCCC. İkinci yaklaşım, ardışık iki satırda farklı değerler yazmaya dayanmaktadır. Aynı sütunda, ardışık iki satırın birbirini etkileyip etkilemediğini anlamak için birine bit-0, diğerine bit-1 yazılır. Bu yaklaşımı değerlendirmek için 0x5555 ile 0xAAAA ve 0x3333 ile 0xCCCC kullanıyoruz. Şekil 4.4'de, ilk yaklaşımın değerlerini ilk 4 bitlik değerleriyle ifade ediyoruz, örneğin F; 0xFFFF, A; 0xAAAA anlamına gelir. Ayrıca, ikinci yaklaşım için, 0x5555 ile 0xAAAA için A5 ve 0x3333 ile 0xCCCC için C3 kullanıyoruz. Şekil 4.4, veri örüntülerinin ortalama ve maksimum entropisini gösterir.



(a) Kart-A



(b) Kart-B

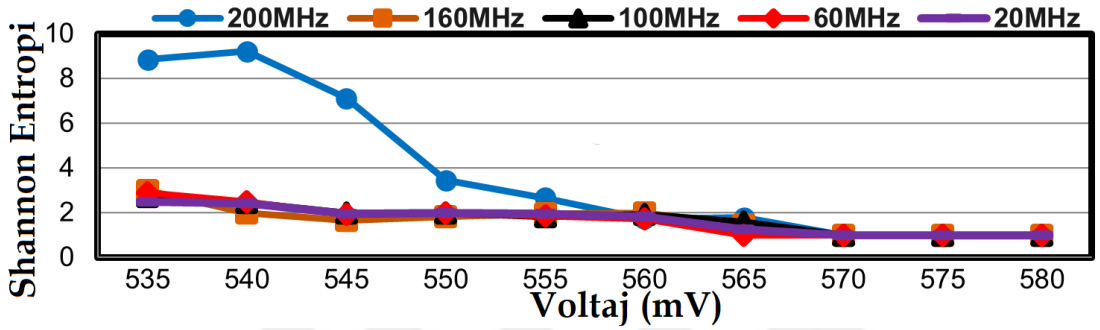
Şekil 4.4: Her SRAM yongasındaki farklı veri örüntüleri için maksimum ve ortalama 32 bit blok entropisi.

Şekil 4.4'den üç temel gözlem yapıyoruz. İlk olarak, bu kartların veri örüntüleri davranışı önceki iki deneyin aksine aynıdır. 0x0000 veri modeli, maksimum ve ortalama entropi için en düşük değere sahiptir. Ayrıca en yüksek maksimum entropi 0xFFFF değeri ile gözlenmektedir. Bunun nedeni, hataların çoğunlukla mantık-1 değerine sahip hücrelerde gerçekleşmesidir. Bu, önceki çalışmaların sonuçlarıyla da paraleldir [3]. İkincisi, birinci yaklaşımın diğer değerleri, verilerinde aynı sayıda mantık-1 içerdiğinden maksimum ve ortalama entropi açısından hemen hemen aynı değerlere sahiptirler. Üçüncüsü, ikinci yaklaşımın veri örüntüleri diğerlerine benzer değerlere sahiptir, çünkü örüntüler de aynı miktarda mantık-1'e sahiptir. Bu, ardışık satırların birbirini etkilemediğini gösterir.

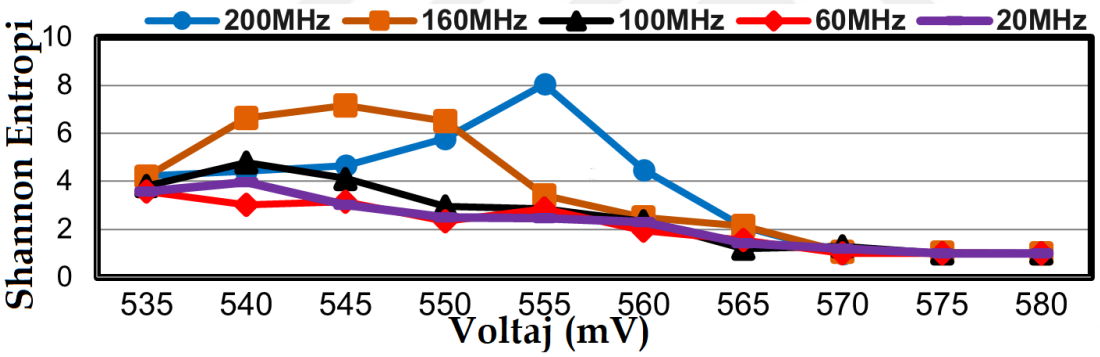
4.2.5 Frekans ve gerilim ilişkisi

Bölüm 4.2.3 kısmında, farklı voltaj seviyelerinde sadece 200MHz çalışma frekansı davranışını analiz ediyoruz ve Şekil 4.3'de en yüksek maksimum entropinin minimum çalışma voltajının üzerinde olduğunu gösteriyoruz. Bu nedenle, bu bölümde farklı voltaj seviyelerinde farklı frekansların rastgelelik davranışını inceliyoruz.

Analiz etmek için frekans seviyeleri olarak 20MHz, 60MHz, 100MHz, 160MHz ve 200Mhz ve veri örüntüsü olarak 0xFFFF seçiyoruz. Ayrıca, voltaj seviyesi aralığı, Şekil 4.3'te de görülebileceği gibi, maksimum entropinin doyduğu ve daha sonra değişmediği minimum çalışma voltajından (yani 535mV) 580mV'a kadardır.



(a) Kart-A



(b) Kart-B

Şekil 4.5: Her bir SRAM yongasında farklı çalışma voltajı ve frekansı setlerinde maksimum 32 bit blok entropisi.

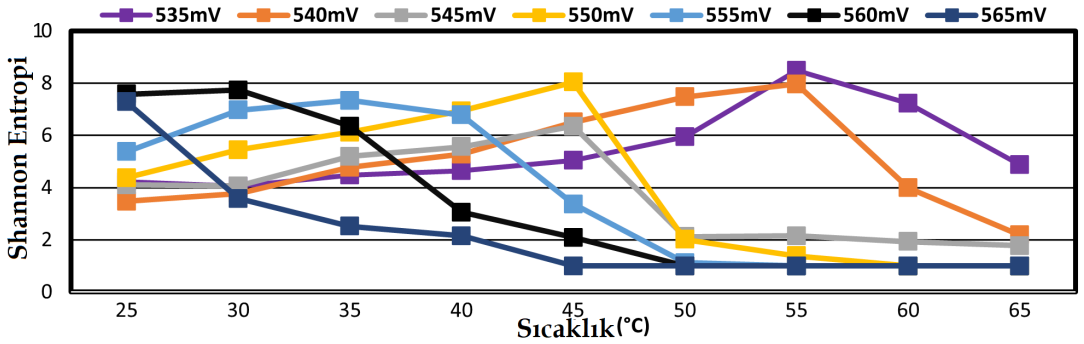
Şekil 4.5, her iki SRAM yongası için farklı voltaj ve frekans parametre setlerinin ortalama ve maksimum entropisini gösterir. Buradan yapılan çıkarımlar ise *(i)* Şekil 4.3 ve Şekil 4.2'e benzer şekilde, her iki kart için de en yüksek maksimum entropiye *farklı* çalışma voltajı ve frekans seviyelerinde elde edildiğini, örneğin Şekil 4.5b, *(ii)* maksimum entropinin tepe değerine ulaşan voltaj seviyesi her frekans için aynı olmadığını ve *(iii)* çeşitli farklı bir dizi voltaj ve frekans parametresinde 4.5a ve 4.5b'dan 200MHz, Kart-A ve Kart-B için sırasıyla 9.21 ve 8.04 olmak üzere en yüksek maksimum entropiye sahip olduğunu gözlemliyoruz.

Mümkün olduğunca düşük voltaj (örneğin 535 mV) ve yüksek frekans (200 MHz) olarak çalışma parametreleri ayarlandığında, SRAM hücrelerinin büyük çoğunluğu için okuma hatası olasılığı %100'e ulaşır. %50'lik bir okuma hatası oranı sergileyen SRAM hücrelerinin sayısının 555 mV'de maksimize edildiğini gözlemliyoruz. Voltajı

555 mV'nin üzerine çıkardığımızda, %50 olasılıkla başarısız olan SRAM hücrelerinin sayısı azaldıkça %0 olasılıkla başarısız olan SRAM hücrelerinin sayısı artar. Bu nedenle, Şekil 4.5b'da görülen monoton olmayan bir davranış gözlemlenmiştir.

4.2.6 Sıcaklık

Sıcaklığın entropi üzerindeki etkisini inceliyoruz. Bu deneyi gerçekleştirmek için çevresel sıcaklık kontrol cihazını kullanıyoruz. PMBus arayüzünü kullanarak bir FPGA kartının yerleşik canlı sıcaklığını izliyoruz. Her iki kart için de frekansı 200MHz olarak ayarlandı ve veri örüntüsü olarak 0xFFFF kullanıdı. Sırasıyla $25°C$ ila $65°C$ ve 535mV ila 565mV arasında değişen farklı sıcaklık ve voltaj seviyeleri çiftleri altında entropiyi analiz ediyoruz. Şekil 4.6, Kart-B için farklı çalışma voltajı ve sıcaklık çiftleri arasındaki en yüksek 32-bit blok entropisini gösterir. En yüksek voltaj seviyesi olarak 565mV, en düşük voltaj seviyesi olarak 535mV ve nominal sıcaklıkta ($45°C$) en yüksek entropiye ulaşan 550mV olmak üzere üç voltaj seviyesini vurguluyoruz. Ayırca, Kart-A için de benzer bir davranış gözlemliyoruz.



Şekil 4.6: Kart-B için farklı çalışma voltajı ve sıcaklık setlerinde maksimum 32 bit blok entropisi

Şekil 4.6'dan üç temel gözlem yapıyoruz, 1) çoğu sıcaklık seviyesi için farklı voltaj seviyelerinde en yüksek maksimum entropiye ulaşılır, 2) daha düşük sıcaklıklarda daha yüksek voltaj seviyeleri en yüksek entropiye ulaşır (örneğin $25°C$'da en yüksek entropi 565mV'den üretilir) ve daha yüksek sıcaklıklarda daha düşük voltaj seviyeleri en yüksek entropiye ulaşır (örneğin $> 55°C$'da en yüksek entropi 535mV'den üretilir) ve 3) her sıcaklık seviyesinde düşük voltajlı SRAM rastgelelik sergiler ve en az > 7 entropi üretebilen bir voltaj seviyesine sahiptir.

4.2.7 Tartışma

Karakterizasyon, bir entropi kaynağı olarak kullanılabilecek düşük voltajlı SRAM hücrelerini tanımlayan tek seferlik ve düşük maliyetli bir işlemdir. Bu bölümde, önce zamna bağımlılığın ve süreç varyasyonunun SRAM hücrelerinin entropisi üzerindeki etkisini tartışıyoruz. Son olarak, SRAM bloklarının besleme voltajını düşük ölçtüğümüzde neden rastgelelik gözlemlediğimizi tartışıyor ve varsayım geliştiriyoruz.

Zamana Bağlılık: SRAM yaşlanmasının düşük voltajlı SRAM hücrelerinin entropisini olumsuz etkilememesini sağlamak için, karakterizasyon sonuçlarımızın

yaşlanmasını ampirik olarak değerlendirir ve dört ay sonra tüm deneyleri tekrarlarız. Aynı sonuçları başarıyla yeniden üretiyoruz. Bu nedenle, rastgelelik karakterizasyon sonuçlarının en az dört ay geçerli olmasını bekliyoruz.

Süreç Varyasyonu: Önceki çalışma [3], voltaj koruma bandının ve minimum çalışma voltajı seviyesinin farklı SRAM yongaları arasında değişiklik gösterdiğini göstermektedir (benzer gözlemler, HBM'ler [157] ve DRAM'ler [19]) gibi diğer bellek teknolojileri için de geçerlidir. süreç varyasyonu nedeniyle. İki FPGA kartı üzerinde rastgelelik karakterizasyonu gerçekleştiriyoruz ve aynı çalışma koşulları altında maksimum ve ortalama entropide farklı davranışları gözlemliyoruz. Rastgelelik davranışının farklı SRAM cihazlarında değişebileceği sonucuna vardık. Bu nedenle, rastgelelik karakterizasyonu her SRAM yongası için bir kez gerçekleştirilmelidir.

Varsayımsal Hata Mekanizması: SRAM zamanlama parametreleri SRAM üreticisi tarafından belirlenir ve başarılı cihaz çalışması için bu parametrelere uyulması gerekmektedir [158]. Modern SRAM'ler kendi kendine zamanlamalı, yani tüm dahili zamanlama parametreleri bir SRAM dizisinin [83] zamanlama devresi tarafından üretilip işlenmektedir ve dışarıdan herhangi bir müdahale veya değişiklik mümkün değildir.

Bir zamanlama mantık devresinin, algılama yükseltecinin diferansiyel voltajı ne zaman algıladığını belirlemesi gerekir. Bir hücrede okuma işlemi gerçekleştirildiğinde, bit hatlarının voltaj farkı, algılama yükselteci devresine gönderilir. Bir hücreden bir algılama yükseltecine diferansiyel voltaj yayılma gecikmesi (T_{pss}), bit hatlarının farkını yükseltmek için çok kısaysa, algılama yükselteci yanlış sonuçlar üretebilir [82]. T_{pss}, minimum bit hattı diferansiyel voltajı gecikmesi tarafından kısmen belirlenir. Daha büyük bir diferansiyel voltaj zamanlama gecikmesi, algılama yükselteçleri bit hatlarını boşaltmak ve önşarj etmek için daha fazla zaman harcadıklarından, ekstra okuma erişimi gecikmesi ve enerji tüketimi maliyetiyle daha güvenilir bir algılama işlemi sağlar [80]. Süreç varyasyonu, gerekli bit hattı salınımının miktarını değiştiren algılama yükselteçlerinde zamanlama gecikmesi açısından bir farklılığa yol açar [83].

T_{pss}, uzamsal farklılıklar ve süreç varyasyonu nedeniyle tüm SRAM çipinde değişir [80]. Süreç varyasyonu nedeniyle, T_{pss}, tüm SRAM çipinde en yüksek T_{pss}'in değeri olarak belirler ve en kötü çevre koşullarında bile doğru işlevsellik güvencesi sağlamak için ekstra band eklenir. Bölüm 2.3.1'da tartıştığımız gibi, düşük voltaj devre yayılma gecikmesini arttırır. Bu nedenle, hücreden algılama yükselteci arasındaki T_{pss} artar ve zamanlama kısıtlamalarını ihlal eder (örneğin, Bölüm 2.2'da açıklanan sütun çoklayıcı sinyalini etkinleştirme zamanlaması).

Hücrelerdeki bit hatasının, algı yükseltecinin önceki çalışmalarda gözlemlere benzer şekilde iki bitlik hattın voltaj farkını yanlış bir şekilde yükseltmesine neden olan bu ihlalden kaynaklandığını varsayıyoruz [82, 159, 160, 161]. Bu hipotezi desteklemek için, önce voltajı düşürüp ölçtüğümüz ve ardından rastgele hücreleri (yani, rastgele değerler üreten SRAM hücreleri) tanımladığımız başka bir deney yapıyoruz. Gerilimi nominal seviyeye yükselttiğimizde, bu hücreler deneyden önce başlattığımız doğru değeri tutarlı bir şekilde sağlamaktadır. Bu gözlem, voltajı düşürdüğümüzde, SRAM hücresinin içinde bit hatalarının meydana gelmediğini, ancak bozulmamış değeri yanlış örnekleyen algılama yükseltecinde meydana geldiğini göstermektedir.

Rastgeleliğin Nedeni: Düşük voltaj uyguladığımızda, tüm hatalı SRAM hücrelerinin her seferinde deterministik düşük voltaj hatasına sahip olmadığını gözlemliyoruz. Bunun, T_{pss} ihlallerinin algılama yükseltecinin güvenilir algılama marjının altında bir diferansiyel voltajı yükseltmesine neden olduğuna inanıyoruz [162]. Bu nedenle, algılama yükselteci, diferansiyel voltajı belirsiz bir şekilde örneklemektedir (%50'lik olasılık VDD veya GND'ye). Sonuç olarak, bu hücrelerin okuma işlemi, düşük voltajlı SRAM'lerde rastgelelik (yani entropi) üretir.

4.3 TuRaN: SRAM tabanlı GRSÜ

SRAM hücrelerindeki düşük voltaj arızalarına ilişkin rastgelelik analizimize dayanarak, SRAM bloklarında voltaj düşük ölçekleme gerçekleştiren ve ortaya çıkan hataları bir kriptografik karma işlevi olan SHA-256 kullanarak işleyen yeni bir SRAM tabanlı TRNG olan TuRaN'ı öneriyoruz. TuRaN, voltaj güvenli voltaj marjının altına düştüğünde, SRAM hücrelerinin algılama işleminde belirsiz bir şekilde başarısız olduğu ve bu deterministik olmayan arızaların bir entropi kaynağı olarak kullanılabileceği gözleminden yararlanır. TuRaN üç adımdan oluşur: 1) karakterizasyon sonrası belirlenmiş çalışma parametrelerinin ayarlanması: *(i)* SRAM'lerin frekansı ve besleme voltajı, *(ii)* SRAM'ın her hücresine mantık-1 yazma, 2) karakterizasyon sonrası belirlenmiş en yüksek entropili satırları okuma 3) yüksek kaliteli, 256-bit gerçek rasgele sayı elde etmek için SHA-256 karma işlevini gerçekleştirerek her bloğu sonradan işleme.

4.3.1 TuRaN'ın Değerlendirmesi

TuRaN'ı FPGA kartlarına gömülü, kullanıma hazır SRAM yongaları üzerinde değerlendiriyoruz. Değerlendirmemizi iki özdeş Xilinx ZC702 FPGA kartı örneği üzerinde yapıyoruz [72]. Bu platformu TuRaN'ı değerlendirmek için kullanıyoruz çünkü 1) SRAM yongalarının hem frekansını hem de besleme voltajını kolayca manipüle etmemize, 2) sonradan işleme yöntemini donanım olarak bu platforma uygulanabileceğinden hızlı deneysel deneyler yapmamıza ve 3) enerji tüketimini bir voltaj regülatörü/güç kontrolörü aracılığıyla izlememize olanak sağlıyor. TuRaN'ı dört kategoride değerlendiriyoruz. İlk olarak, standart NIST STS rastgelelik testleri [73] kullanarak oluşturulan rastgele sayıların kalitesini değerlendiriyoruz. İkinci olarak, farklı frekans seviyeleri için TuRaN'ın aktarım hızını analiz ediyoruz. Üçüncüsü, TuRaN'ın enerji tüketimini değerlendiriyoruz. Dördüncüsü, TuRaN'ın gerçek rastgele sayı üretme gecikmesini değerlendiriyoruz. En yüksek entropinin gözlemlendiği voltaj seviyesine bağlı olarak her bir frekans seviyesi için besleme voltajını seçiyoruz (örneğin, Kart-A için 200MHz, 540mV). TuRaN'ın yüksek aktarım hızı, yüksek enerji verimliliği ve düşük gecikme ile yüksek kaliteli gerçek rastgele sayılar ürettiğini gösteriyoruz.

4.3.1.1 Kalite

TuRaN tarafından üretilen rastgele sayıların kalitesini değerlendirmek için her iki FPGA'dan rastgele bit akışları çıkarıyoruz. 1 Gbit rastgele bit akışı oluşturuyoruz ve onu her biri 1 Mb (2^{20} bit) uzunluğunda olacak şekilde 1024 diziye bölüyoruz.

Bu 1024 diziyi NIST STS [73] testlerini kullanarak test ediyoruz. NIST STS, birkaç istatistiksel test formüle ederek rastgeleliği değerlendirmek için kullanılır. Her testin, testin boş hipotezinin durumunu gösteren bir p değeri vardır. Eğer p değeri, α önem seviyesinden büyükse, test edilen diziler gerçekten rastgeledir.

Çizelge 4.1: TuRaN'ın NIST STS Rastgelelik Testi Sonuçları

NIST Test İsmi	**p-değeri** ($\alpha = 0.01$)	**Test Sonucu**
Frequency	0.42649	BAŞARILI
Block Frequency	0.24730	BAŞARILI
Cumulative Sums	0.38451	BAŞARILI
Runs	0.63712	BAŞARILI
Longest Run	0.09818	BAŞARILI
Rank	0.55003	BAŞARILI
DFT	0.07785	BAŞARILI
Non-Overlapping Template	0.51272	BAŞARILI
Overlapping Template	0.67787	BAŞARILI
Universal	0.84941	BAŞARILI
Approximate Entropy	0.28524	BAŞARILI
Random-Excursions	0.67243	BAŞARILI
Random-Excursions Variant	0.52986	BAŞARILI
Serial	0.58120	BAŞARILI
Linear Complexity	0.01383	BAŞARILI

Çizelge 4.1 $\alpha = 0.01$ olan rastgelelik için 15 testin tamamında p-değeri cinsinden 1024 1Mbit dizinin ortalama sonuçlarını gösterir. Her test için p değerleri, önem düzeyi değerinden (α) daha büyüktür. Bu nedenle, her boş hipotez, her testi başarıyla geçer.

Sonuçlarımız, 1Mbit dizilerin (toplamda 1024) 99.02%'ının her NIST testini geçtiğini gösteriyor. Bu yüzde, STS testleri için NIST tarafından belirlenen kabul edilebilir aralıktadır ($> 98.84\%$) [73] $(1-\alpha) \pm 3\sqrt{\frac{\alpha(1-\alpha)}{k}}$ burada α önem düzeyidir ve k test edilen dizilerin sayısıdır. Bu, TuRaN'ın yüksek kaliteli gerçekten rastgele sayılar ürettiğini gösterir.

Yüksek kaliteli gerçek rastgele sayılar üretmek için yeterli rastgelelik toplamak için gereken okuma sayısı (SHA-256 için 256-bit entropi elde etmek), çeşitli çalışma frekansı seviyeleri için farklılık gösterir. Bunun nedeni, Şekil 4.5'de gözlemlediğimiz gibi, düşük frekans seviyelerinin daha düşük entropiye sahip olma eğiliminde

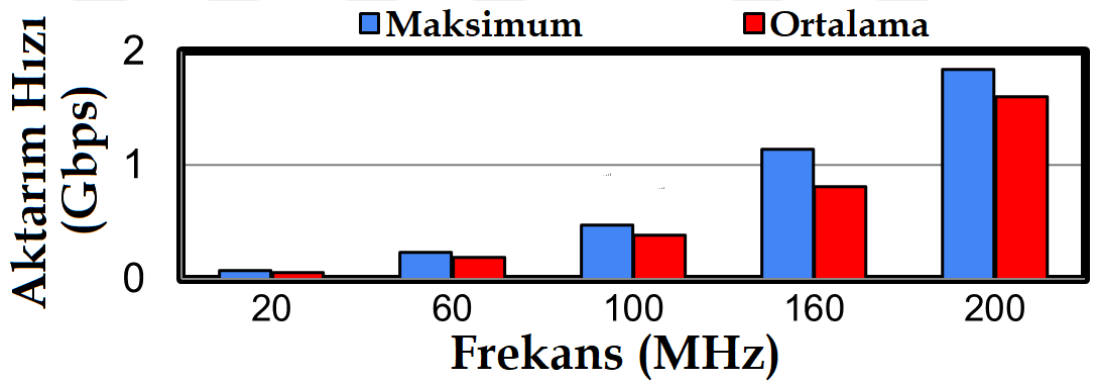
olmasıdır. Buna göre kapsamlı bir analiz yapabilmek için değerlendirmemizi beş farklı frekans seviyesinde (yani 20MHz, 60MHz, 100MHz, 160MHz ve 200MHz) yapıyoruz. Bu değerlendirmeye dayanarak, artan sıklık ile gerekli okuma sayısının azaldığını gözlemliyoruz. Okuma sayısının sırasıyla (85, 79, 66, 50 ve 32) yukarıda belirtilen frekans seviyeleriyle ilişkili olduğunu bulduk.

4.3.1.2 Aktarım hızı

TuRaN'ın aktarım hızını değerlendirmek için, önce 256 bit entropi biriktirmek için en yüksek entropiye sahip SRAM satırına gerekli okuma işlemi sayısını belirleriz. Ardından, sonradan işleme (SHA-256) adımının gecikme ve aktarım hızı üzerindeki etkisini hesaplıyoruz.

$$Throughput = \frac{256bit}{N_{read} \times \left(1/f_{req}\right)} \tag{4.2}$$

Denklem 4.2, f_{req} SRAM'ın çalışma frekansını gösterir, N_{read} 256 bit entropi elde etmek için gerekli okuma sayısını gösterir. Bu denklemden, TuRaN kullanarak gerçek SRAM bloklarından gerçekten rasgele sayıları için maksimum $1.812Gbps$ ve ortalama $1.6Gbps$ aktarım hızıyla üretiyoruz.



Şekil 4.7: TuRaN'ın farklı çalışma frekansı seviyelerinde maksimum ve ortalama aktarım hızı

Şekil 4.7, iki set SRAM bloğu için Bölüm 4.2'da tanımlanan beş farklı frekans seviyesinden elde edilen ortalama ve maksimum verimi gösterir.

Frekansı artırmanın hem maksimum hem de ortalama aktarım hızını *üstel olarak* artırdığını gözlemliyoruz. Örneğin, 200MHz'de maksimum hız, 20MHz'de gözlemlenen hızdan $25.8x$ daha yüksektir. Bunun nedeni, 4.2.2 Bölümünde belirtildiği gibi, daha yüksek frekans seviyelerinin daha yüksek entropi elde etmesidir, bu da 256-bit entropi biriktirmek için gereken okuma işlemi *yani N_read* miktarını azaltır. Ayrıca, Denklem 4.2 ile frekans ve aktarım hızı doğru orantılıyken *N_read* ve aktarım hızı ters orantılıdır.

4.3.1.3 Enerji

TuRaN'ın enerji tüketimini iki adımda değerlendiriyoruz: 1) Voltaj ve akım değerlerini elde etmek için PMBus kullanarak okuma işleminin enerji tüketimini izliyoruz, 2) 256 bit gerçek rastgele üretmek için SHA-256 hash fonksiyonunun enerjisini hesaplıyoruz.

İlk adımda, okuma işlemlerinin enerji tüketimini elde etmek için SRAM bloklarının akım ve gerilim hattının değerlerini kaydediyoruz. Her frekans seviyesi için en yüksek maksimum entropiye ulaşan SRAM'ın besleme gerilimini seçiyoruz.

İkinci olarak, değerlendirmelerinde aynı FPGA kartını, ZC702'yi, ve SHA-256 gerçekleştirmek için herhangi bir SRAM bloğu kullanmayan bir tasarım önerdikleri için SHA-256'nın güç ve aktarım hızı sonuçlarını geçmiş bir çalışmadan elde ediyoruz [163]. Bu çalışma, bir SHA-256 donanımının $0.1W$ tüketirken $917Mbps$' aktarım hızına ulaştığını bildirmektedir. Aktarım hızı üzerinde tam performans elde etmek için uygulamamız gereken SHA-256 hızlandırıcılarının sayısını değişken olarak belirliyoruz çünkü SHA-256 donanımının hızı ilk adımdan çok daha düşükse sistemin aktarım hızı önemli ölçüde azalabilir. $1.812Gbps$ maksimum aktarım hızını elde ettiğimizden, bunu korumak için iki SHA-256 bloğu kullanıyoruz.

$$E_{Read} = N_{read} \times T_{read} \times P_{dd} \tag{4.3a}$$

$$E_{SHA} = P_{SHA} \times T_{SHA} \tag{4.3b}$$

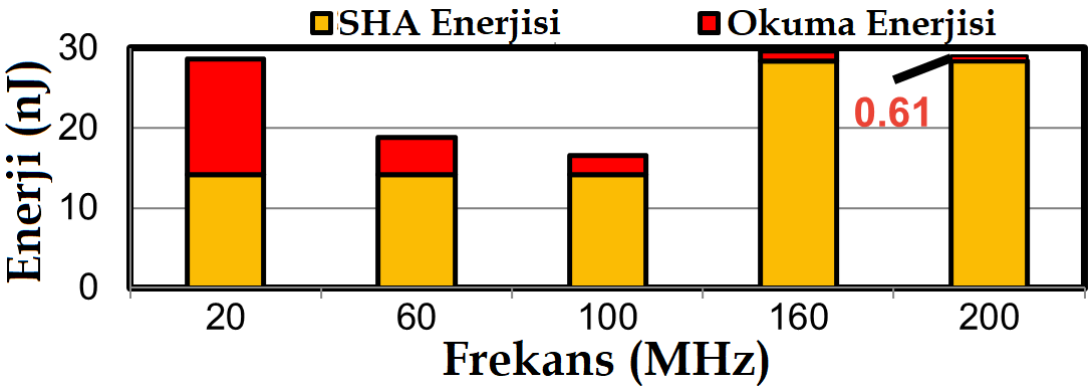
Denklem 4.3a, okunan enerji tüketimini gösterir, E_{Read}. N_{read} 256 bit entropiye sahip bir dizi elde etmek için gerçekleştirilecek okuma sayısını belirtir. P_{dd}, okuma işlemlerini gerçekleştirirken SRAM bloklarının güç tüketimini gösterir. T_{read}, SRAM'ın saniye cinsinden okunma süresini belirtir.

Denklem 4.3b, SHA-256 hızlandırıcısının enerji tüketimini gösterir, E_{SHA}. P_{SHA}, önceki çalışmaya göre $0.1W$ olan bir SHA-256 karma işleminin güç tüketimini belirtir [163]. İki SHA-256 hızlandırıcı kullandığımız için, bizim durumumuzda P_SHA, $0.2W$'a eşittir. T_{SHA}, 256 bit çıktı elde etmek için SHA-256'nın çalışma süresini saniye cinsinden gösterir.

Enerji tüketimimizi 4.3a ve 4.3b Denklemini kullanarak hesaplıyoruz. Sonuçlarımız TuRaN'ın bir bitlik gerçek rastgele sayı üretmek için $0.11nJ$ tükettiğini gösteriyor.

Şekil 4.8, 256-bit gerçek rasgele sayılar üretmek için her iki SRAM bloğu grubu için daha önce Bölüm 4.2.5'de tanımlanan beş farklı frekans seviyesinde elde edilen enerji tüketimini gösterir.

Şekil 4.8'den üç temel gözlem yapıyoruz: 1) SHA-256 hızlandırıcı, 20MHz hariç her frekans seviyesinde toplam enerji tüketimini domine eder. 2) Düşük frekanslarda okuma işlemlerinin güç tüketimi azalsa da, toplam okuma işlemlerinin enerji tüketimi düşük frekans seviyelerinde daha yüksektir. Çünkü SRAM daha düşük bir frekansta çalıştığında okuma işlemi sayısı ve erişim gecikmesi artar. Örneğin, 200MHz'deki enerji tüketimi, 20MHz'deki enerji tüketiminden $23,57x$ daha düşüktür. 3) 100MHz'den sonra, SHA-256 hızlandırıcısının enerji tüketimi, 160MHz



Şekil 4.8: TuRaN'ın farklı çalışma frekansı seviyelerinde iki adımının ortalama enerji tüketimi

(1.144*Gbps*) ve 200MHz (1.812*Gbps*) maksimum aktarım hızını desteklemek için SHA-256 hızlandırıcılarının sayısı iki katına çıkarıldığından artar.

4.3.1.4 Gecikme

TuRaN gecikmesi, 1) gerilim hatlarını kullanabilmek için PMBus kurulum süresi, 2) gerilim alt ölçeklendirme komutunun yürütme süresi, 3) SRAM erişim gecikmesi (yazma ve okuma işlemleri dahil) ve 4) sonradan işleme donanımının kurulum süresi ile doğrudan ilişkilidir.

ZC702'nin ARM tabanlı işletim sistemini kullanarak 1., 2. ve 3. işlemlerin gecikmesini ölçüyoruz. Dördüncü işlem için, önceki çalışmanın gözlemini kullanıyoruz [163]. PMBus'un kurulum zamanında, sistem ilgili konfigürasyon parametreleri ayarlanır ve SRAM'lerin besleme voltajının altında ölçeklenmek için işlemlerde bulunur. Sistemi başlattıktan sonra gerilimi düşürmek için düşük gerilim komutunu göndermekteyiz. Üçüncü işlemde, SHA-256'nın giriş bit akışlarını elde etmek için düşük voltajlı SRAM'deki bir satıra erişiriz. Rastgele sayı üretmenin son adımında 256 bitlik gerçek rastgele sayılar üretmek için SHA-256 işlemini gerçekleştiriyoruz.

İlk işlem PMBus sistemini etkinleştirmek için 228,3 μs alır, ikinci adımda SRAM'ın besleme voltajını azaltmak için 49,7 μs, üçüncü adım için 200MHz erişim gecikmesinde bir kez yazma için ve gerekli miktarda okuma işlemi 160*ns* alır. Ancak 200MHz'de iki SHA-256 bloğu kullandığımız için bu, 320*ns* sonucuyla gecikmeyi iki katına çıkarır. Son olarak, dördüncü adım 142.2*ns* alır. 200MHz'de 278,46μs gecikme elde ederiz. Üçüncü adım dışındaki tüm adımlar, herhangi bir çalışma koşulundan (ör. frekans, voltaj) bağımsız olduğundan, farklı frekanslar arasındaki toplam gecikmedeki fark, okuma erişim süresi (3. adım) tarafından belirlenir. En düşük gecikmeyi 200MHz'de, 278.46μs'da ve en yüksek gecikmeyi 20MHz'de 282.39μs'da gözlemliyoruz.

4.4 Son teknoloji SRAM tabanlı GRSÜ'lerle Karşılaştırma

Bildiğimiz kadarıyla, bu, bir entropi kaynağı olarak SRAM'lerde düşük voltaj tabanlı hataları kullanan ilk çalışmadır. TuRaN'ı sürekli çalışma, en yüksek aktarım hızı, bit başına enerji tüketimi ve 256 bit gecikme süresi açısından son teknoloji SRAM tabanlı TRNG'lerle karşılaştırıyoruz. Çizelge 4.2, TuRaN ve önceki son teknoloji SRAM tabanlı TRNG'lerin bir özetini ve karşılaştırmasını gösterir [59, 57]. TuRaN'ın SRAM tabanlı TRNG'lerin sahip olması gereken tüm özellikleri (Bölüm 4.1) karşıladığını ve tüm karşılaştırma noktalarında önceki çalışmalardan sırasıyla $2.26x$ aktarım hızında, $5.39x$ enerji verimliliğinde, $5.09x$ gecikmede üstün olduğunu göstermekteyiz.

Ne yazık ki, son teknoloji SRAM tabanlı GRSÜ çalışmaları [57, 59] toplam gecikmelerini bildirmemiştir. Önceki çalışma [57], aç-kapa döngüsü süresinin gecikmesini deneysel olarak değerlendirdiğinden, aç-kapa döngüsü gecikmesinden bahsetmeyen işler için aç-kapa döngüsü gecikmesi olarak $250ms$ değerini kullanırız. Ayrıca SRAM'ın çalışma frekansı ve besleme voltajının ne olduğuna değinmemişlerdir. Kalan parametreleri (gecikme ve enerji tüketimi) değerlendirmek için, önceki her çalışma için SRAM'ın 200MHz'de çalıştığını ve besleme voltajının değerlendirmemizin nominal voltaj seviyesi olan $1V$ olduğunu iyimser bir şekilde varsayıyoruz. 256 bit entropi elde etmek için okuma işlemlerinin sayısını hesaplayarak değerlendirme platformumuzdaki PMBus kullanarak önceki çalışmaların enerji tüketimini değerlendiriyoruz [1].

Zhang+ [57]: Zhang ve diğerleri, SRAM üzerinde iyonizasyon ışınlaması kullanarak GRSÜ performansını artıran SRAM tabanlı bir GRSÜ önermektedir. Diğer önceki çalışmaların aksine, bu çalışma aç-kapa döngüsü süresini 250 ms'den 1,5 ms'ye düşerek en düşük gecikmeye sahiptir. Yazarlar, 64 çevrim gecikme süresi ile SRAM'den elde edilen bit akışlarını sonradan işlemek için ZC702 FPGA kartında (TuRaN ile aynı) 200 MHz'de çalışan SHA-256 donanımını uygulamaktadır. Ayrıca, Zhang ve diğerleri, aktarım hızları $178Mbps$ olarak bildirmiştir. Ancak GRSÜ'lerinin herhangi bir enerji tüketiminden bahsetmemişlerdir. Bu iyimser parametrelere dayanarak, Zhang+'ın GRSÜ'sü gerçek rastgele bit başına $0,56nJ$ tüketir. Önerilen TRNG'nin gecikmesini değerlendirmek için, geliştirilmiş aç-kapa döngüsünün gecikmesini, SRAM okuma erişimi gecikmesini ve işlem sonrası işlevinin gecikmesini dikkate alıyoruz. Zhang+'ın SRAM tabanlı GRSÜ'sinin gecikme süresi, aç-kapa döngüsüsünün domine ettiği için $1.501ms$'dır.

PUFKEY [59]: PUFKEY, fiziksel olarak klonlanamayan iki farklı işlev (PUF) kullanarak gerçek rastgele sayılar üretir. İlk adımda, SRAM hücrelerinin başlangıç değeri, koşullu bir algoritma (u-Quark) kullanılarak gerçek rastgele tohumlar elde etmek için bir entropi kaynağı olarak kullanılır. İkinci olarak, ilk adımın çıktısını (gerçek rastgele tohumlar) içeren bir bit akışı, gerçek rastgele sayılar üretmek üzere bir donanım RSÜ'sü (DRSÜ) için bir girdi olarak kullanılır. Yazarlar, PUFKEY'in $803Mbps$ aktarım hızı elde ettiğini bildirmektedir. Gözlemlerine göre, u-Quark'ın 400 baytı işlemek için 2550 döngüye (0.0255 saniye) ihtiyacı vardır. $803Mbps$ elde etmek için, yazarların $5.1s$ gecikme süresine sahip 52.4 u-Quark blokları uyguladığını varsayıyoruz. İkinci adımın, DRSÜ'nün enerji tüketimi, rapor edilmemiştir. Bu özel

[1] Önceki çalışmaların her bir SRAM hücresinin 1-bit entropisini ideal durum olan (tamamen rastgele) 1 olduğunu varsayıyoruz, bu varsayım ayrıca TuRaN'dan $3.47x$ daha yüksektir.

bir donanım olduğu için DRSÜ'nün enerji tüketimini hesaplayamıyoruz. DRSÜ'nün gecikme süresi 159.22ns olarak bildirilmiştir. Bu hesaplamalara ve gözlemlere dayanarak PUFKEY'in toplam gecikme süresi $5,35$'dır.

Çizelge 4.2: Önceki SRAM-tabanlı GRSÜ'lerle TuRaN'ın Karşılaştırılması

Çalışma	Devamlı Üretim	En Yüksek Aktarım Hızı	Enerji Tüketimi	256-bit Üretim Gecikmesi
Zhang+[57]	✗	$178Mbps$	$0.56nJ/bit$	$1.501ms$
PUFKEY [59]	✗	$803Mbps$	N/A	$5.35s$
TuRaN	✓	$1.812Gbps$	$0.11nJ/bit$	$278.46\mu s$

4.5 Sistem Entegrasyonu

Bu bölümde, TuRaN'ın modern sistemlere nasıl entegre edilebileceğini tartışıyoruz. Bir işlemcinin önbellek hiyerarşisinde TuRaN'ı simüle ederek iki sistem entegrasyon senaryosu ve bunların ödünleşmelerini gösteriyoruz. Bölüm 4.3'da tartışıldığı gibi, TuRaN gerçek rastgele sayılar üretmek için iki adımdan oluşur: 1) 256-bit entropili bir bit akışı elde edene kadar SRAM'den satırları okuma, 2) ilk adımın çıktısını SHA-256'ya karma işlevine gönderme. Deneyimizde, ilk adımı etkinleştirmek için entropi kaynağı olarak L1 veya L2 önbelleğini kullanıyoruz [2]. Önbellek satırlarının voltajını düşürmek için önbellekleri, Drowsy Cache [75]'nin değiştirilmiş bir versiyonu olarak simüle ediyoruz. İkinci adım için, SHA-256 şifreleme karma işlevini gerçekleştirmek için CPU'yu kullanıyoruz.

4.5.1 Mekanizma

Temel Kurulum: Drowsy Cache [75], önbellek satırlarının voltajını kontrol etmek için bir önbellek satırına uykulu bir bit ve bir satır geçit devresi ekler. Önbellek denetleyicisi uykulu biti izler. Önbellek hattına erişildiğinde, önbellek denetleyicisi, ilgili hattın besleme voltajının durumunu belirlemek için uykulu biti okur. Önbellek hattının besleme gerilimi, uykulu bitin durumuna bağlı olarak önbellek hattının gerilimini nominal ve düşük (uykulu) besleme gerilimleri arasında değiştiren gerilim denetleyicisi tarafından ölçeklenir. Uykulu önbellek hattına erişildiğinde, uykulu bit mantık-0'a çekilir ve önbellek hattının besleme gerilimi nominal gerilime çevrilir. Tek performans kaybı, ilgili veriler yanlış okunabileceğinden, önbellek satırı uykulu moddayken ortaya çıkar. Önbellek denetleyicisi, önbellek satırını uykulu moda geçirmek için uykulu biti periyodik olarak ayarlar. Uykulu bir önbelleğin ek yükü, bir önbellek satırı için $< 3\%$'dır [75].

TuRaN'ın Sistem Entegrasyon Kurulumu: TuRaN'ın entegrasyonu için, önbellek satırlarında düşük gerilime dayalı zamanlama hatalarını etkinleştirerek rasgele sayılar üretmek için Drowsy Cache'te değişiklikler öneriyoruz. Bozulmuş verileri bir entropi

[2] TuRaN'ın entegrasyonu L1/L2 önbellekleriyle sınırlı değildir ve herhangi bir SRAM yapısına kolayca genişletilebilir (örn. önbellek, yazmaç öbeği, dallanma öngörücü birimleri vb.).

kaynağı olarak kullanmak istediğimiz için ilk olarak satır geçit devresini ortadan kaldırıyoruz. İkincisi, Drowsy Cache uygulamasının aksine, önbellek satırındaki geçerli verileri bozmamak için, gerçek rasgele sayılar üretmek için yeterli boş döngü olduğunda, önbellek satırlarını periyodik olarak uykulu moda sokmuyoruz.

Rastgele Bit Akışları Oluşturma: Eşzamanlı olarak çalışan görevleri etkilemeden rastgele sayılar oluşturmak için önbelleklerin boş döngülerini kullanıyoruz. Karakterizasyon sonrası belirlenmiş yüksek entropili önbellek satırlarını boşaltıyoruz ve değerleri daha derin bellek seviyelerine aktarıldığında onları uyku moduna alıyoruz. Uykulu bir önbellek, önbellek satırı modunu (uykulu veya normal) değiştirmek için bir saat döngüsü gecikmesine sahiptir. Bir önbellek satırından rastgelelik oluşturmak için 4 saat döngüsüne ihtiyacımız var, ikisi modu değiştirmek için (normalden uykuluya ve uykuludan normal moda), biri veri örüntüsü yazmak için (tüm mantık-1'ler) ve biri önbellek satırını okumak için. Daha sonra 4 saat döngüsünde 64 bayt uzunluğunda bir bit akışı elde ediyoruz. Bu veriyi depolamak için yalnızca $r_{rastgele}$ adlı tek bir genel yazmaca ihtiyacımız var. 256-bit entropiye sahip bit akışı $r_{rastgele}$'ye depolandığında, rastgele sayılar üretmek için boş döngüler olsa bile rastgele sayılar üretmeyi durdurmak için bir denetim sinyalini sıfırlıyoruz. Ardından, çekirdek gerçek bir rasgele sayı isterse, yalnızca $r_{rastgele}$'yi okur ve sonraki rasgele sayı isteği için 256 bit entropili bir bit akışı elde etmek için denetim sinyalini mantık-1'e çeker. Gerçek rasgele sayılar üretmek için CPU, işlev için bir girdi olarak $r_{rastgele}$ değerini kullanarak SHA-256 gerçekleştirir.

SHA-256'nın CPU'da İşlemi: Elde edilen bit akışını ($r_{rastgele}$) sonradan işlemek için SHA-256 şifreleme karma işlevini gerçekleştiririz. CPU'yu SHA-256'yı gerçekleştirmek için kullanıyoruz çünkü (i) çağdaş CPU'lar özel SHA-1 ve SHA-256 buyruklarıyla donatılmıştır [164], bu, 1MB'den fazla blok girişi [165] için 27.984 Gbps hızla SHA-256 gerçekleştirmemizi sağlar ve (ii) bizi özel SHA-256 donanımı eklemekten kurtarır, böylece ek bir alan ve enerji yüküne neden olmaz.

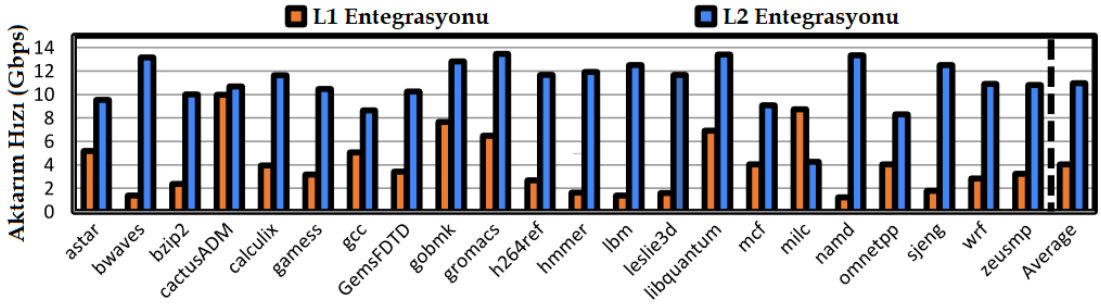
4.5.1.1 Değerlendirme

Modern sistemlerde TuRaN'ın rastgele sayı üretme faydalarını tahmin etmek için gem5 [76] kullanarak simülasyonlar yapıyoruz. SPEC2006 iş yüklerini, modern bir üst düzey CPU [74] özelliklerine sahip simüle edilmiş sistemde çalıştırıyoruz. L1 veri önbelleğinin ve L2 önbelleğin boşta kalma döngülerini analiz ediyoruz ve ardından bu boş döngülere TuRaN mekanizması komutlarını yerleştiriyoruz. Her önbellek satırı için önbelleğin sadece bir yolunu kullanıyoruz. Her çalıştırmada, seçilen bir yoldan farklı bir önbellek satırı seçiyoruz ve rastgele bir sayı üretmek için her zaman bu satırı çıkarıyoruz.

4.5.1.2 Sonuçlar

Şekil 4.9, SPEC2006 iş yükleri [77] için TuRaN'ın ortalama aktarım hızını gösterir. 3.6 GHz'de, en kötü durum senaryosu olan ve iki okuma işlemi gerektiren en yüksek entropiye sahip önceden tanımlanmış her 32 bitlik önbellek satır bloğu için

200 MHz'in entropi sonuçlarını kullanıyoruz. Çünkü, 4.2.2 Bölümünde bahsedildiği gibi, çalışma frekansını arttırmak aktarma hızını katlanarak arttırır. Önbelleklerin boş döngülerinde rasgele sayılar üretmek için gereken süreyi bularak Denklem 4.2'e benzer bir şekilde aktarım hızını hesaplıyoruz.



Şekil 4.9: SPEC2006 iş yükleri için modern sistemlerde iki TuRaN entegrasyon senaryosunun ortalama aktarım hızı.

TuRaN, L1 veri önbelleğinde (L2 önbellek) ortalama 4.03$Gbps$ (10.95$Gbps$) ve maksimum 9.96$Gbps$ (13.46$Gbps$) hızla gerçek rastgele sayılar üretir. TuRaN, gerçek bir rastgele sayı ürettiğinde karakterizasyon sonrası tanımlanan yükske entropili önbellek hattını tahliye ettiğinden, TuRaN sistem performansını ortalama 4.86% (1.92%) ile düşürür. 27.984$Gbps$ aktarım hızı ile SHA-256 fonksiyonunu gerçekleştirmek için CPU kullandığımızdan, karma işlevi TuRaN'ın performansını azaltmaz.

Sistem entegrasyonumuzun ihmal edilebilir bir ek alan yükü vardır. TuRaN'ın alan yükünü CACTI [166] kullanarak değerlendiriyoruz. Önceki çalışma [75] için bildirilen her bir önbellek satırı için %3'den düşük olan L1 ve L2 Önbellekteki Drowsy Cache uygulamasının ek yükü, sırasıyla $0,00135mm^2$, $0,0108mm^2$'dır. 1024 bitlik yazmaç, $r\_rastgele$, için $0.0003mm^22$'dır. TuRaN, L1 veri önbellek entegrasyonu için ek $0,00165mm^22$ ve L2 önbellek entegrasyonu için ek $0,0111mm^22$ alan gerektirir.

4.5.1.3 Tartışma

Bu çalışmada, her ölçekteki bilgi işlem cihazlarında kullanılabilecek gelecek vaat eden bir GRSÜ olmak için yaygın olarak kullanılan SRAM cihazlarından nasıl yararlanılacağına odaklanıyoruz. TuRaN, ticari CPU'larda daha ince taneli bir voltaj denetleyicisi eklenerek etkinleştirilebilir (örn. Drowsy Cache).

Bu bölümde, TuRaN'ın performans ek yükünü nasıl en aza indirebileceğimizi ve bir sistem tasarımcısının TuRaN'ı yazılımda etkinleştirmek için bir arabirimi nasıl ortaya çıkarabileceğini tartışacağız. Ayrıca, Bölüm 4.2.6 entropinin farklı çevresel koşullar altında azalabileceğini gösterdiğinden TuRaN'ın sağlamlığını ve güvenilirliğini tartışıyoruz.

Boşta Döngüleri Tahmin Etme: Boşta kalma döngüleri, bellek adresleri ve yükle-sakla kuyrukları(-*ing.* load-store queue) kullanılarak tahmin edilebilir. Son çalışma [167], DRAM tabanlı GRSÜ'ler için son erişilen bellek adreslerini ve bellek istek kuyruklarını kullanarak DRAM'deki boşta kalma döngülerini tahmin eden

uçtan uca bir sistem tasarımı geliştirmiştir. Benzer bir yaklaşımı mimari düzeyindeki tasarımımız için de kullanabiliriz, burada önbelleğin boşta kalma döngülerini, yükle-sakla kuyruklarını ve son isabet/ıska(-*ing*. cache hit-miss) bellek adreslerini kullanarak tahmin edebiliriz. Önbellekte boşta döngü yoksa, önbellek denetleyicisi ya bellek isteklerini durdurabilir ve rasgele sayı arabelleği dolana kadar rasgele sayılar oluşturabilir ya da aynı anda çalışan iş yüklerinin adaletsizliğini ve performans düşüşünü en aza indirmek için daha karmaşık bir politika uygulayabilir. Ancak önbelleği en çok kullanan iş yüklerinde bile rastgele sayı üretimi için boşta çevrimler olduğunu gözlemliyoruz. Buna ek bir önlem olarak, daha fazla fırsat yaratmak için rasgele sayılar üretmek için tüm önbellek seviyeleri aynı anda kullanılabilir.

Donanım/Yazılım Arayüzü: TuRaN'ı modern sistemlerde etkinleştirmek için giriş-çıkış veri yolları ve buyruk kümesi mimarisi uzantıları dahil ancak bunlarla sınırlı olmamak üzere çeşitli Donanım/Yazılım arayüzleri kullanılabilir. Entropi yazmacını okumak için basit bir arayüz sağlamak üzere bellek eşlemeli alan tabanlı giriş-çıkış (-*ing*. memory-mapped I/O) veri yollarını kullanmak, işlemciye rasgele sayılar almak için modern sistemlerde hali hazırda kullanılan $r_{rastgele}$'ye benzer yazmaçlardan giriş-çıkış okuması yapmak (örn., AMD'de TRNG_OUT [168] veya ARM [169] içinde APB tabanlı bağımlı arabirim) uygun bir yaklaşım olabilir. Başka bir yaklaşım olarak, $r_{rastgele}$'ı okumak için rasgele sayıyı modern sistemlerde de kullanılan işlemciye göndererek yeni bir ISA talimatı eklemek olabilir (örneğin, Intel RDRAND talimatı[170]).

Sağlamlık ve Güvenilirlik: TuRaN, GRSÜ entropisinin tahmin edilebilirliğini arttıran süreç, voltaj ve sıcaklık değişiminden yararlanan saldırılara karşı dayanıklıdır. Bu tür saldırıları önlemek için modern işlemciler halihazırda GRSÜ sağlamlığı ve kendi kendini doğrulama testleri gerçekleştiren donanımla donatılmıştır (örn. Intel'in Çevrimiçi Sağlık Testleri ve Yerleşik Kendi Kendine Testler [170]). Bu testler, işlemcilerin bir GRSÜ çıkışındaki entropiyi izlemesini sağlar. GRSÜ çıkışında yeterli entropi yoksa işlemci GRSÜ çıkışını kullanmaz. Böylece saldırganlar, çevresel parametreleri doğrudan veya dolaylı olarak kontrol ederek bir GRSÜ'nün çıktısını manipüle edemezler.

4.6 İlgili Çalışmalar

Bildiğimiz kadarıyla, gerçek rastgele sayılar üretmek için SRAM'lerde düşük voltaj tabanlı zamanlama hatalarından yararlanan ilk çalışmadır. 4.4 Bölümünde, iki son teknoloji SRAM tabanlı GRSÜ'yi kapsamlı bir şekilde açıklıyor ve TuRaN ile karşılaştırıyoruz. Bu bölümde, önceki diğer SRAM tabanlı GRSÜ'leri ve diğer bellek tabanlı (SRAM tabanlı olmayan) GRSÜ'leri kısaca açıklıyoruz ve karşılaştırıyoruz.

4.6.1 SRAM tabanlı GRSÜ'ler

SRAM tabanlı GRSÜ'ler ilk olarak gerçek rastgele sayılar üretmek için SRAM başlangıç değerlerinden yararlanılarak [54]'da önerilmiştir. Ancak, SRAM hücrelerinin çok küçük bir kısmı rastgele davranış gösterir ve 0.1 minimum entropiyi geçmez. Bu nedenle, önceki birçok çalışma [56, 57, 65, 66, 67, 58, 68, 55, 69,

70, 59] daha yüksek minimum entropi ve daha yüksek performanslı SRAM-tabanlı GRSÜ önermektedir. [56], iki aşamalı son işleme işlevleri, SHA-256 ve deterministik rastgele bit oluşturucu (DRSÜ) kullanarak SRAM tabanlı RSÜ oluşturmak için verimli bir algoritma önerir. Ortaya çıkan bit akışı, sözde rasgele sayılar üretmek için bir tohum olarak kullanılır. Minimum entropiyi artırmak için önceki çalışma [65] transistörlerin yaşlanma etkisinden yararlanır. Transistör yaşlanmasına benzer şekilde, [66], minimum entropiyi ve rastgele sayıların kalitesini artırmak için gürültüye duyarlı gömülü SRAM(NS-SRAM) tabanlı bir GRSÜ önerir. Ancak, [66], aktarım hızını ve kullanılan ek alanı değerlendirmek için işlem sonrası işlevlerini hesaba katmaz.

Önceki her SRAM tabanlı GRSÜ, başlatma döngüsüne bağlı olmaları nedeniyle *(i)* sürekli rastgele sayı üretimini sürdüremezler, *(ii)* yüksek gecikme süresinde düşük aktarım hızına sahiptir, *(iii)* nominal çalışma parametrelerinde çalıştıkları için enerji verimliliği sağlamazlar ve *(iv)*, *(i)* ve *(ii)* nedeniyle ticari cihazlara kolayca entegre edilemezler.

4.6.2 Diğer Bellek Tabanlı GRSÜ'ler

DRAM. DRAM tabanlı GRSÜ ile ilgili önceki çalışmalar, gerçek rastgele sayılar oluşturmak için DRAM zamanlama parametrelerini kasıtlı olarak ihlal etmek [61, 62, 60, 171, 172] ve başlangıç değerleri kullanmak gibi farklı yaklaşımlar kullanır [173, 174]. Bu çalışmalar ya (i) enerji tüketimini dikkate almaz ya da (ii) enerji tüketimi düşük değildir, örneğin TuRaN, en yüksek enerji verimliliğine sahip DRAM tabanlı GRSÜ [60]'den $37,6x$ kat daha az enerji tüketmektedir veya (iii) düşük aktarım hızına sahiptir .

FLASH. Önceki Flash tabanlı GRSÜ çalışmaları [175, 63, 64] gerçek rastgele sayılar oluşturmak için bellekteki termal gürültüden ve rastgele telgraf gürültülerinden yararlanır. Ancak, bu çalışmalar arasındaki en yüksek aktarım hızına sahip FLASH tabanlı GRSÜ, TuRaN'dan 1812 kat daha düşük olarak 1 Mbps'dir.

5. SONUÇ

Tezin ilk çalışmasında, ilk yaklaşık hata yerleştirme modeli *yani* yapay hata haritalarını oluşturan bir altyapı olan MoRS'yi öneriyoruz. DSA gibi programlar yonga üstündeki belleğe daha bağımlı olduğundan SRAM'ler daha fazla güç tüketir. Düşük voltaj, enerji tasarrufu için besleme voltajını nominal voltaj seviyesinin altına indiren bir tekniktir. Bununla birlikte, koruma bandının altındaki voltajın daha da düşürülmesi, zamanlama ile ilgili bit-hatalarına neden olur. Hata yerleştirme teknikleri, düşük voltajdan kaynaklanan hataların sistemleri nasıl etkilediğini anlamak için yaygın olarak kullanılmaktadır. Düşük voltaj hata yerleştirme teknikleri, hataları rastgele yerlere rastgele şekilde yerleştirerek gerçekleştirilir. Önerilen altyapının avantajı, heterojen bilgi işlem cihazları dahil olmak üzere çeşitli sistemlere hata yerleştirerek düşük voltaja dayanıklılığı hakkında gerçeğe yeterince yakın bir şekilde sistemin doğruluğunu ve güvenirliğini ölçmektir. En gelişmiş DSA uygulamaları için önerilen altyapının doğruluğunu değerlendirdik. Önerilen modelimizi değerlendirmek için yapay hata modelleri ile gerçek veriler arasındaki doğruluk farkını ölçtük. Rastgele modele dayalı hata yerleştirme mekanizması ile karşılaştırıldığında, önerilen modelin gerçek verilere göre ortalama olarak 3.21x daha yakın doğruluk sağlayabildiğini gösterdik.

Bu tezin ikinci çalışmasında ise, modern sistemlerde düşük maliyetle uygulanabilen, düşük gecikmeli yüksek aktarım hızına sahip ve yüksek enerji verimli SRAM tabanlı bir TRNG olan TuRaN'ı tanıtıyoruz. TuRaN, gerçek rasgele sayılar üretmek için SRAM'lerde besleme gerilimi düşük ölçeklendirmesinden yararlanır ve ortaya çıkan zamanlama hatalarını SHA-256 karma işleviyle işler. TuRaN'ı iki özdeş FPGA kartında karakterize ediyor ve değerlendiriyoruz. Frekansın, voltaj seviyesinin veri örüntüsünün ve sıcaklığın entropiyi nasıl etkilediğini gösteriyoruz. TuRaN tarafından üretilen rastgele sayıları kalite, aktarım hızı, enerji ve gecikme süresi açısından değerlendiriyoruz. TuRaN'ın ortalama $1.6Gbps$ aktarım hızıyla, gerçek rastgele bit başına $0.11nJ$ enerji tüketimiyle ve $278.46\mu s$ gecikme ile NIST STS testini geçen rasgele sayılar ürettiğini gösteriyoruz. TuRaN, aktarım hızında $2,26x$, enerji verimliliğinde $5,09x$ ve gecikme süresinde $5.39x$ son teknoloji SRAM tabanlı TRNG'lerden daha iyi performans gösteriyor. TuRaN'ın son teknoloji ürünü bir CPU L1 veri önbelleğinde (ve L2 önbelleklerinde) iki potansiyel entegrasyonunu gösteriyoruz ve ihmal edilebilir bir $0,00165mm^2$ ($0.0111mm^2$) ek alan yükü ile ortalama $4.03Gbps$ ($10.95Gbps$) aktarım hızı elde ediyoruz.

KAYNAKLAR

[1] **Jeon**, **H.**, **Ravi**, **G. S.**, **Kim**, **N. S.**, and **Annavaram**, **M.** (2015). GPU Register File Virtualization. In: *Proceedings of the 48th International Symposium on Microarchitecture*. MICRO-48. Waikiki, Hawaii: Association for Computing Machinery, 420–432.

[2] **Haj-Yihia**, **J.**, **Yasin**, **A.**, **Asher**, **Y. B.**, and **Mendelson**, **A.** (Dec. 2016). Fine-Grain Power Breakdown of Modern Out-of-Order Cores and Its Implications on Skylake-Based Systems. In: *ACM Trans. Archit. Code Optim.* 13.4.

[3] **Salami**, **B.**, **Unsal**, **O. S.**, and **Kestelman**, **A. C.** (2018). Comprehensive evaluation of supply voltage underscaling in fpga on-chip memories. In: *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, pp. 724–736.

[4] **Papadimitriou**, **G.**, **Chatzidimitriou**, **A.**, **Gizopoulos**, **D.**, **Reddi**, **V. J.**, **Leng**, **J.**, **Salami**, **B.**, **Unsal**, **O. S.**, and **Kestelman**, **A. C.** (2020). Exceeding conservative limits: A consolidated analysis on modern hardware margins. In: *IEEE Transactions on Device and Materials Reliability* 20.2, pp. 341–350.

[5] **Gizopoulos**, **D.**, **Papadimitriou**, **G.**, **Chatzidimitriou**, **A.**, **Reddi**, **V. J.**, **Salami**, **B.**, **Unsal**, **O. S.**, **Kestelman**, **A. C.**, and **Leng**, **J.** (2019). Modern hardware margins: CPUs, GPUs, FPGAs recent system-level studies. In: *2019 IEEE 25th International Symposium on On-Line Testing and Robust System Design (IOLTS)*. IEEE, pp. 129–134.

[6] **Bacha**, **A.** and **Teodorescu**, **R.** (2014). Using ECC Feedback to Guide Voltage Speculation in Low-Voltage Processors. In: *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 306–318.

[7] **Papadimitriou**, **G.**, **Chatzidimitriou**, **A.**, and **Gizopoulos**, **D.** (2019). Adaptive Voltage/Frequency Scaling and Core Allocation for Balanced Energy and Performance on Multicore CPUs. In: *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, pp. 133–146.

[8] **Parasyris**, **K.**, **Koutsovasilis**, **P.**, **Vassiliadis**, **V.**, **Antonopoulos**, **C. D.**, **Bellas**, **N.**, and **Lalis**, **S.** (2018). A Framework for Evaluating Software on Reduced Margins Hardware. In: *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 330–337.

[9] **Papadimitriou**, **G.**, **Kaliorakis**, **M.**, **Chatzidimitriou**, **A.**, **Gizopoulos**, **D.**, **Lawthers**, **P.**, and **Das**, **S.** (2017). Harnessing voltage margins for energy efficiency in multicore CPUs. In: *Proceedings*

of the 50th Annual IEEE/ACM International Symposium on Microarchitecture, pp. 503–516.

[10] **Bacha**, **A.** and **Teodorescu**, **R.** (June 2013). Dynamic Reduction of Voltage Margins by Leveraging On-Chip ECC in Itanium II Processors. In: *SIGARCH Comput. Archit. News* 41.3, 297–307.

[11] **Zou**, **A.**, **Leng**, **J.**, **He**, **X.**, **Zu**, **Y.**, **Gill**, **C. D.**, **Reddi**, **V. J.**, and **Zhang**, **X.** (2018). Voltage-stacked gpus: A control theory driven cross-layer solution for practical voltage stacking in gpus. In: *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, pp. 390–402.

[12] **Salami**, **B.**, **Onural**, **E. B.**, **Yuksel**, **I. E.**, **Koc**, **F.**, **Ergin**, **O.**, **Kestelman**, **A. C.**, **Unsal**, **O.**, **Sarbazi-Azad**, **H.**, and **Mutlu**, **O.** (2020). An experimental study of reduced-voltage operation in modern FPGAs for neural network acceleration. In: *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, pp. 138–149.

[13] **Salami**, **B.**, **Unsal**, **O. S.**, and **Kestelman**, **A. C.** (2019). Evaluating built-in ECC of FPGA on-chip memories for the mitigation of undervolting faults. In: *2019 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*. IEEE, pp. 242–246.

[14] **Salami**, **B.**, **Unsal**, **O.**, and **Cristal**, **A.** (2018). Fault characterization through FPGA undervolting. In: *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, pp. 85–853.

[15] **Salami**, **B.** (2018). Aggressive undervolting of FPGAs: power & reliability trade-offs. In.

[16] **Reagen**, **B.**, **Whatmough**, **P.**, **Adolf**, **R.**, **Rama**, **S.**, **Lee**, **H.**, **Lee**, **S. K.**, **Hernández-Lobato**, **J. M.**, **Wei**, **G.**, and **Brooks**, **D.** (2016). Minerva: Enabling Low-Power, Highly-Accurate Deep Neural Network Accelerators. In: *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pp. 267–278.

[17] **Chandramoorthy**, **N.**, **Swaminathan**, **K.**, **Cochet**, **M.**, **Paidimarri**, **A.**, **Eldridge**, **S.**, **Joshi**, **R. V.**, **Ziegler**, **M. M.**, **Buyuktosunoglu**, **A.**, and **Bose**, **P.** (2019). Resilient Low Voltage Accelerators for High Energy Efficiency. In: *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 147–158.

[18] **Zhang**, **J.**, **Rangineni**, **K.**, **Ghodsi**, **Z.**, and **Garg**, **S.** (2018). Thundervolt: enabling aggressive voltage underscaling and timing error resilience for energy efficient deep learning accelerators. In: *Proceedings of the 55th Annual Design Automation Conference*, pp. 1–6.

[19] **Chang**, **K. K.**, **Yağlıkçı**, **A. G.**, **Ghose**, **S.**, **Agrawal**, **A.**, **Chatterjee**, **N.**, **Kashyap**, **A.**, **Lee**, **D.**, **O'Connor**, **M.**, **Hassan**, **H.**, and **Mutlu**, **O.** (2017). Understanding reduced-voltage operation in modern DRAM devices: Experimental characterization, analysis, and

mechanisms. In: *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 1.1, pp. 1–42.

[20] **David**, **H.**, **Fallin**, **C.**, **Gorbatov**, **E.**, **Hanebutte**, **U. R.**, and **Mutlu**, **O.** (2011). Memory power management via dynamic voltage/frequency scaling. In: *Proceedings of the 8th ACM international conference on Autonomic computing*, pp. 31–40.

[21] **Deng**, **Q.**, **Meisner**, **D.**, **Ramos**, **L.**, **Wenisch**, **T. F.**, and **Bianchini**, **R.** (2011). MemScale: Active Low-Power Modes for Main Memory. In: *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS XVI. Newport Beach, California, USA: Association for Computing Machinery, 225–238.

[22] **Larimi**, **S. S. N.**, **Salami**, **B.**, **Unsal**, **O. S.**, **Kestelman**, **A. C.**, **Sarbazi-Azad**, **H.**, and **Mutlu**, **O.** (2020). Understanding Power Consumption and Reliability of High-Bandwidth Memory with Voltage Underscaling. arXiv: `2101.00969 [cs.AR]`.

[23] **Yang**, **L.** and **Murmann**, **B.** (2017b). SRAM voltage scaling for energy-efficient convolutional neural networks. In: *2017 18th International Symposium on Quality Electronic Design (ISQED)*. IEEE, pp. 7–12.

[24] **Yang**, **L.** and **Murmann**, **B.** (2017a). Approximate SRAM for Energy-Efficient, Privacy-Preserving Convolutional Neural Networks. In: *2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pp. 689–694.

[25] **Cai**, **Y.**, **Yalcin**, **G.**, **Mutlu**, **O.**, **Haratsch**, **E. F.**, **Crista**, **A.**, **Unsal**, **O. S.**, and **Mai**, **K.** (2013). ERROR ANALYSIS AND RETENTION-AWARE ERROR MANAGEMENT FOR NAND FLASH MEMORY. In: *Intel Technology Journal* 17.1.

[26] **Cai**, **Y.**, **Ghose**, **S.**, **Haratsch**, **E. F.**, **Luo**, **Y.**, and **Mutlu**, **O.** (2017). Error Characterization, Mitigation, and Recovery in Flash-Memory-Based Solid-State Drives. In: *Proceedings of the IEEE* 105.9, pp. 1666–1704.

[27] **Cai**, **Y.**, **Haratsch**, **E. F.**, **Mutlu**, **O.**, and **Mai**, **K.** (2012). Error patterns in MLC NAND flash memory: Measurement, characterization, and analysis. In: *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, pp. 521–526.

[28] **Cai**, **Y.**, **Luo**, **Y.**, **Ghose**, **S.**, and **Mutlu**, **O.** (2015). Read disturb errors in MLC NAND flash memory: Characterization, mitigation, and recovery. In: *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE, pp. 438–449.

[29] **Torres-Huitzil**, **C.** and **Girau**, **B.** (2017). Fault and Error Tolerance in Neural Networks: A Review. In: *IEEE Access* 5, pp. 17322–17341.

[30] **Deng**, **J.**, **Fang**, **Y.**, **Du**, **Z.**, **Wang**, **Y.**, **Li**, **H.**, **Temam**, **O.**, **Ienne**, **P.**, **Novo**, **D.**, **Li**, **X.**, **Chen**, **Y.**, and **Wu**, **C.** (2015). Retraining-based timing error mitigation for hardware neural networks. In: *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 593–596.

[31] **Chatzidimitriou**, **A.**, **Papadimitriou**, **G.**, **Gizopoulos**, **D.**, **Ganapathy**, **S.**, and **Kalamatianos**, **J.** (2018). Analysis and Characterization of Ultra Low Power Branch Predictors. In: *2018 IEEE 36th International Conference on Computer Design (ICCD)*, pp. 144–147.

[32] **Salami**, **B.**, **Unsal**, **O. S.**, and **Kestelman**, **A. C.** (2018). On the Resilience of RTL NN Accelerators: Fault Characterization and Mitigation. In: *2018 30th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pp. 322–329.

[33] **Koppula**, **S.**, **Orosa**, **L.**, **Yağlıkçı**, **A. G.**, **Azizi**, **R.**, **Shahroodi**, **T.**, **Kanellopoulos**, **K.**, and **Mutlu**, **O.** (2019). EDEN: Enabling energy-efficient, high-performance deep neural network inference using approximate DRAM. In: *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 166–181.

[34] **Lecun**, **Y.**, **Bottou**, **L.**, **Bengio**, **Y.**, and **Haffner**, **P.** (1998). Gradient-based learning applied to document recognition. In: *Proceedings of the IEEE* 86.11, pp. 2278–2324.

[35] **Krizhevsky.**, **A.** (2014). cuda-convnet. `https://github.com/akrizhevsky/cuda-convnet2`.

[36] **Koizumi**, **H.**, **Morikatsu**, **S.**, **Aida**, **H.**, **Nozawa**, **T.**, **Kakesu**, **I.**, **Uchida**, **A.**, **Yoshimura**, **K.**, **Muramatsu**, **J.**, and **Davis**, **P.** (2013). Information-theoretic secure key distribution based on common random-signal induced synchronization in unidirectionally-coupled cascades of semiconductor lasers. In: *Optics express* 21.15, pp. 17869–17893.

[37] **Sadeghi**, **A.-R.**, **Visconti**, **I.**, and **Wachsmann**, **C.** (2010). Enhancing RFID security and privacy by physically unclonable functions. In: *Towards hardware-intrinsic security*. Springer, pp. 281–305.

[38] **Kaenel**, **V. von** and **Takayanagi**, **T.** (2007). Dual true random number generators for cryptographic applications embedded on a 200 million device dual CPU SoC. In: *2007 IEEE Custom Integrated Circuits Conference*. IEEE, pp. 269–272.

[39] **Jun**, **B.** and **Kocher**, **P.** (1999). The Intel random number generator. In: *Cryptography Research Inc. white paper* 27, pp. 1–8.

[40] **Cherkaoui**, **A.**, **Fischer**, **V.**, **Fesquet**, **L.**, and **Aubert**, **A.** (2013). A very high speed true random number generator with entropy assessment. In: *International Conference on Cryptographic Hardware and Embedded Systems*. Springer, pp. 179–196.

[41] **Bagini**, **V.** and **Bucci**, **M.** (1999). A design of reliable true random number generator for cryptographic applications. In: *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, pp. 204–218.

[42] **Bernstein**, **D. J.**, **Chang**, **Y.-A.**, **Cheng**, **C.-M.**, **Chou**, **L.-P.**, **Heninger**, **N.**, **Lange**, **T.**, and **Van Someren**, **N.** (2013). Factoring RSA keys from certified smart cards: Coppersmith in the wild. In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, pp. 341–360.

[43] **Dichtl**, **M.** (2003). How to predict the output of a hardware random number generator. In: *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, pp. 181–188.

[44] **Steele Jr**, **G. L.**, **Lea**, **D.**, and **Flood**, **C. H.** (2014). Fast splittable pseudorandom number generators. In: *ACM SIGPLAN Notices* 49.10, pp. 453–472.

[45] **Blum**, **L.**, **Blum**, **M.**, and **Shub**, **M.** (1986). A simple unpredictable pseudo-random number generator. In: *SIAM Journal on computing* 15.2, pp. 364–383.

[46] **Mascagni**, **M.** and **Srinivasan**, **A.** (2000). Algorithm 806: SPRNG: A scalable library for pseudorandom number generation. In: *ACM Transactions on Mathematical Software (TOMS)* 26.3, pp. 436–461.

[47] **Eichenauer**, **J.** and **Lehn**, **J.** (1986). A non-linear congruential pseudo random number generator. In: *Statistische Hefte* 27.1, pp. 315–326.

[48] **Gong**, **L.**, **Zhang**, **J.**, **Liu**, **H.**, **Sang**, **L.**, and **Wang**, **Y.** (2019). True random number generators using electrical noise. In: *IEEE Access* 7, pp. 125796–125805.

[49] **Huang**, **M.**, **Wang**, **A.**, **Li**, **P.**, **Xu**, **H.**, and **Wang**, **Y.** (2014). Real-time 3 Gbit/s true random bit generator based on a super-luminescent diode. In: *Optics Communications* 325, pp. 165–169.

[50] **Zbilut**, **J. P.**, **Giuliani**, **A.**, and **Webber Jr**, **C. L.** (2000). Recurrence quantification analysis as an empirical test to distinguish relatively short deterministic versus random number series. In: *Physics Letters A* 267.2-3, pp. 174–178.

[51] **Rahman**, **M. T.**, **Xiao**, **K.**, **Forte**, **D.**, **Zhang**, **X.**, **Shi**, **J.**, and **Tehranipoor**, **M.** (2014). TI-TRNG: Technology independent true random number generator. In: *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, pp. 1–6.

[52] **Srinivasan**, **S.**, **Mathew**, **S.**, **Ramanarayanan**, **R.**, **Sheikh**, **F.**, **Anders**, **M.**, **Kaul**, **H.**, **Erraguntla**, **V.**, **Krishnamurthy**, **R.**, and **Taylor**, **G.** (2010). 2.4 GHz 7mW all-digital PVT-variation tolerant true random number generator in 45nm CMOS. In: *2010 Symposium on VLSI Circuits*. IEEE, pp. 203–204.

[53] **Puglisi**, **F. M.**, **Zagni**, **N.**, **Larcher**, **L.**, and **Pavan**, **P.** (2018). Random telegraph noise in resistive random access memories: Compact modeling and advanced circuit design. In: *IEEE Transactions on Electron Devices* 65.7, pp. 2964–2972.

[54] **Holcomb**, **D. E.**, **Burleson**, **W. P.**, and **Fu**, **K.** (2008). Power-up SRAM state as an identifying fingerprint and source of true random numbers. In: *IEEE Transactions on Computers* 58.9, pp. 1198–1210.

[55] **Holcomb**, **D. E.**, **Burleson**, **W. P.**, **Fu**, **K.**, et al. (2007). Initial SRAM state as a fingerprint and source of true random numbers for RFID tags. In: *Proceedings of the Conference on RFID Security*. Vol. 7, p. 01.

[56] **Leest**, **V. van der**, **Sluis**, **E. van der**, **Schrijen**, **G.-J.**, **Tuyls**, **P.**, and **Handschuh**, **H.** (2012). "Efficient Implementation of True Random Number Generator Based on SRAM PUFs". In: *Cryptography and Security: From Theory to Applications: Essays Dedicated to*

Jean-Jacques Quisquater on the Occasion of His 65th Birthday. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 300–318.

[57] **Zhang**, **X.**, **Jiang**, **C.**, **Dai**, **G.**, **Zhong**, **L.**, **Fang**, **W.**, **Gu**, **K.**, **Xiao**, **G.**, **Ren**, **S.**, **Liu**, **X.**, and **Zou**, **S.** (2020). Improved Performance of SRAM-Based True Random Number Generator by Leveraging Irradiation Exposure. In: *Sensors* 20.21, p. 6132.

[58] **Wang**, **W.**, **Guin**, **U.**, and **Singh**, **A.** (2020). Aging-Resilient SRAM-based True Random Number Generator for Lightweight Devices. In: *Journal of Electronic Testing* 36, pp. 301–311.

[59] **Li**, **D.**, **Lu**, **Z.**, **Zou**, **X.**, and **Liu**, **Z.** (2015). PUFKEY: A high-security and high-throughput hardware true random number generator for sensor networks. In: *Sensors* 15.10, pp. 26251–26266.

[60] **Kim**, **J. S.**, **Patel**, **M.**, **Hassan**, **H.**, **Orosa**, **L.**, and **Mutlu**, **O.** (2019). D-RaNGe: Using commodity DRAM devices to generate true random numbers with low latency and high throughput. In: *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, pp. 582–595.

[61] **Olgun**, **A.**, **Patel**, **M.**, **Yağlıkçı**, **A. G.**, **Luo**, **H.**, **Kim**, **J. S.**, **Bostancı**, **N.**, **Vijaykumar**, **N.**, **Ergin**, **O.**, and **Mutlu**, **O.** (2021). QUAC-TRNG: High-Throughput True Random Number Generation Using Quadruple Row Activation in Commodity DRAM Chips. In: *arXiv preprint arXiv:2105.08955*.

[62] **Talukder**, **B. B.**, **Kerns**, **J.**, **Ray**, **B.**, **Morris**, **T.**, and **Rahman**, **M. T.** (2019). Exploiting DRAM latency variations for generating true random numbers. In: *2019 IEEE International Conference on Consumer Electronics (ICCE)*. IEEE, pp. 1–6.

[63] **Chakraborty**, **S.**, **Garg**, **A.**, and **Suri**, **M.** (2020). True random number generation from commodity NVM chips. In: *IEEE Transactions on Electron Devices* 67.3, pp. 888–894.

[64] **Wang**, **Y.**, **Yu**, **W.-k.**, **Wu**, **S.**, **Malysa**, **G.**, **Suh**, **G. E.**, and **Kan**, **E. C.** (2012). Flash memory for ubiquitous hardware security functions: True random number generation and device fingerprints. In: *2012 IEEE Symposium on Security and Privacy*. IEEE, pp. 33–47.

[65] **Kiamehr**, **S.**, **Golanbari**, **M. S.**, and **Tahoori**, **M. B.** (2017). Leveraging aging effect to improve SRAM-based true random number generators. In: *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. IEEE, pp. 882–885.

[66] **Rahman**, **M. T.**, **Forte**, **D.**, **Wang**, **X.**, and **Tehranipoor**, **M.** (2016). Enhancing noise sensitivity of embedded SRAMs for robust true random number generation in SoCs. In: *2016 IEEE Asian Hardware-Oriented Security and Trust (AsianHOST)*. IEEE, pp. 1–6.

[67] **Sadhu**, **A.**, **Das**, **K.**, **De**, **D.**, and **Kanjilal**, **M. R.** (2020). SSTRNG: self starved feedback SRAM based true random number generator using quantum cellular automata. In: *Microsystem Technologies* 26.7, pp. 2203–2215.

[68] **Yeh**, **P.-S.**, **Yang**, **C.-A.**, **Chang**, **Y.-H.**, **Chih**, **Y.-D.**, **Lin**, **C.-J.**, and **King**, **Y.-C.** (2019). Self-convergent trimming SRAM true random number

generation with in-cell storage. In: *IEEE Journal of Solid-State Circuits* 54.9, pp. 2614–2621.

[69] **Clark**, **L. T.**, **Medapuram**, **S. B.**, and **Kadiyala**, **D. K.** (2018). SRAM circuits for true random number generation using intrinsic bit instability. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 26.10, pp. 2027–2037.

[70] **Wang**, **R.**, **Selimis**, **G.**, **Maes**, **R.**, and **Goossens**, **S.** (2020). Long-term continuous assessment of SRAM PUF and source of random numbers. In: *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, pp. 7–12.

[71] **Cortez**, **M.**, **Dargar**, **A.**, **Hamdioui**, **S.**, and **Schrijen**, **G.-J.** (2012). Modeling SRAM start-up behavior for physical unclonable functions. In: *2012 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*. IEEE, pp. 1–6.

[72] **Xilinx** (n.d.[a]). Xilinx Zynq ZC702 FPGA Board. `https://www.xilinx.com/products/boards-and-kits/ek-z7-zc702-g.html`.

[73] **Bassham III**, **L. E.**, **Rukhin**, **A. L.**, **Soto**, **J.**, **Nechvatal**, **J. R.**, **Smid**, **M. E.**, **Barker**, **E. B.**, **Leigh**, **S. D.**, **Levenson**, **M.**, **Vangel**, **M.**, **Banks**, **D. L.**, et al. (2010). Sp 800-22 rev. 1a. a statistical test suite for random and pseudorandom number generators for cryptographic applications.

[74] **WikiChip** (n.d.). Cascade Lake SP - Intel. `https://en.wikichip.org/wiki/intel/cores/cascade_lake_sp`.

[75] **Flautner**, **K.**, **Kim**, **N. S.**, **Martin**, **S.**, **Blaauw**, **D.**, and **Mudge**, **T.** (2002). Drowsy caches: Simple techniques for reducing leakage power. In: *ACM SIGARCH Computer architecture news* 30.2, pp. 148–157.

[76] **Binkert**, **N.**, **Beckmann**, **B.**, **Black**, **G.**, **Reinhardt**, **S. K.**, **Saidi**, **A.**, **Basu**, **A.**, **Hestness**, **J.**, **Hower**, **D. R.**, **Krishna**, **T.**, **Sardashti**, **S.**, et al. (2011). The gem5 simulator. In: *ACM SIGARCH computer architecture news* 39.2, pp. 1–7.

[77] **Henning**, **J. L.** (2006). SPEC CPU2006 benchmark descriptions. In: *ACM SIGARCH Computer Architecture News* 34.4, pp. 1–17.

[78] **Sabbagh**, **M.**, **Gongye**, **C.**, **Fei**, **Y.**, and **Wang**, **Y.** (2019). Evaluating Fault Resiliency of Compressed Deep Neural Networks. In: *2019 IEEE International Conference on Embedded Software and Systems (ICESS)*, pp. 1–7.

[79] **Li**, **G.**, **Hari**, **S. K. S.**, **Sullivan**, **M.**, **Tsai**, **T.**, **Pattabiraman**, **K.**, **Emer**, **J.**, and **Keckler**, **S. W.** (2017a). Understanding Error Propagation in Deep Learning Neural Network (DNN) Accelerators and Applications. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC '17. Denver, Colorado: Association for Computing Machinery.

[80] **Pavlov**, **A.** and **Sachdev**, **M.** (2008). CMOS SRAM circuit design and parametric test in nano-scaled technologies: process-aware SRAM design and test. Vol. 40. Springer Science & Business Media.

[81] **Pavlov**, **A. S.** (2005). Design and test of embedded SRAMs. University of Waterloo.

[82] **Weste**, **N. H.** and **Eshraghian**, **K.** (1985). Principles of CMOS VLSI design: a systems perspective. Addison-Wesley Longman Publishing Co., Inc.

[83] **Ishibashi**, **K.** and **Osada**, **K.** (2011). Low power and reliable SRAM memory cell and array design. Vol. 31. Springer Science & Business Media.

[84] **Upadhyay**, **P.**, **Ghosh**, **S.**, **Kar**, **R.**, **Mandal**, **D.**, and **Ghoshal**, **S. P.** (2014). Low static and dynamic power MTCMOS based 12T SRAM cell for high speed memory system. In: *2014 11th International Joint Conference on Computer Science and Software Engineering (JCSSE)*, pp. 212–217.

[85] **Azam**, **T.**, **Cheng**, **B.**, and **Cumming**, **D. R. S.** (2010). Variability resilient low-power 7T-SRAM design for nano-scaled technologies. In: *2010 11th International Symposium on Quality Electronic Design (ISQED)*, pp. 9–14.

[86] **Chen**, **G.**, **Sylvester**, **D.**, **Blaauw**, **D.**, and **Mudge**, **T.** (2010). Yield-Driven Near-Threshold SRAM Design. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 18.11, pp. 1590–1598.

[87] **Leng**, **J.**, **Buyuktosunoglu**, **A.**, **Bertran**, **R.**, **Bose**, **P.**, and **Reddi**, **V. J.** (2015). Safe limits on voltage reduction efficiency in GPUs: A direct measurement approach. In: *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 294–307.

[88] **Tang**, **A.**, **Sethumadhavan**, **S.**, and **Stolfo**, **S.** (2017). {CLKSCREW}: exposing the perils of security-oblivious energy management. In: *26th {USENIX} Security Symposium ({USENIX} Security 17)*, pp. 1057–1074.

[89] **Chatzidimitriou**, **A.**, **Panadimitriou**, **G.**, **Gizopoulos**, **D.**, **Ganapathy**, **S.**, and **Kalamatianos**, **J.** (2019). Assessing the Effects of Low Voltage in Branch Prediction Units. In: *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 127–136.

[90] **Rivière**, **L.**, **Najm**, **Z.**, **Rauzy**, **P.**, **Danger**, **J.**, **Bringer**, **J.**, and **Sauvage**, **L.** (2015). High precision fault injections on the instruction cache of ARMv7-M architectures. In: *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pp. 62–67.

[91] **Kaliorakis**, **M.**, **Tselonis**, **S.**, **Chatzidimitriou**, **A.**, **Foutris**, **N.**, and **Gizopoulos**, **D.** (2015). Differential Fault Injection on Microarchitectural Simulators. In: *2015 IEEE International Symposium on Workload Characterization*, pp. 172–182.

[92] **Lecun**, **Y.** and **Cortes**, **C.** (1999). The MNIST database of handwritten digits. `http://yann.lecun.com/exdb/mnist/`.

[93] **Krizhevsky**, **A.**, **Nair**, **V.**, and **Hinton**, **G.** (n.d.). The CIFAR-10 dataset. `https://www.cs.toronto.edu/~kriz/cifar.html`.

[94] **Zhou**, **A.**, **Yao**, **A.**, **Guo**, **Y.**, **Xu**, **L.**, and **Chen**, **Y.** (2017). Incremental Network Quantization: Towards Lossless CNNs with Low-Precision Weights. arXiv: 1702.03044 [cs.CV].

[95] **Han**, **S.**, **Liu**, **X.**, **Mao**, **H.**, **Pu**, **J.**, **Pedram**, **A.**, **Horowitz**, **M. A.**, and **Dally**, **W. J.** (2016). EIE: Efficient Inference Engine on Compressed Deep Neural Network. arXiv: 1602.01528 [cs.CV].

[96] **Zhu**, **Z.**, **Sun**, **H.**, **Lin**, **Y.**, **Dai**, **G.**, **Xia**, **L.**, **Han**, **S.**, **Wang**, **Y.**, and **Yang**, **H.** (2019). A Configurable Multi-Precision CNN Computing Framework Based on Single Bit RRAM. In: *Proceedings of the 56th Annual Design Automation Conference 2019*. DAC '19. Las Vegas, NV, USA: Association for Computing Machinery.

[97] **Molchanov**, **P.**, **Tyree**, **S.**, **Karras**, **T.**, **Aila**, **T.**, and **Kautz**, **J.** (2017). Pruning Convolutional Neural Networks for Resource Efficient Inference. arXiv: 1611.06440 [cs.LG].

[98] **Yazdani**, **R.**, **Riera**, **M.**, **Arnau**, **J.**, and **González**, **A.** (2018). The Dark Side of DNN Pruning. In: *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pp. 790–801.

[99] **Han**, **S.**, **Pool**, **J.**, **Tran**, **J.**, and **Dally**, **W. J.** (2015). Learning both Weights and Connections for Efficient Neural Networks. arXiv: 1506.02626 [cs.NE].

[100] **Shen**, **Y.**, **Ferdman**, **M.**, and **Milder**, **P.** (2017). Escher: A CNN Accelerator with Flexible Buffering to Minimize Off-Chip Transfer. In: *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 93–100.

[101] **Deng**, **C.**, **Liao**, **S.**, **Xie**, **Y.**, **Parhi**, **K. K.**, **Qian**, **X.**, and **Yuan**, **B.** (2020). PERMDNN: Efficient Compressed DNN Architecture with Permuted Diagonal Matrices. arXiv: 2004.10936 [cs.CV].

[102] **Shen**, **Y.**, **Ferdman**, **M.**, and **Milder**, **P.** (2018). Maximizing CNN Accelerator Efficiency Through Resource Partitioning. arXiv: 1607.00064 [cs.AR].

[103] **Zhang**, **C.**, **Li**, **P.**, **Sun**, **G.**, **Guan**, **Y.**, **Xiao**, **B.**, and **Cong**, **J.** (2015). Optimizing FPGA-Based Accelerator Design for Deep Convolutional Neural Networks. In: *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. FPGA '15. Monterey, California, USA: Association for Computing Machinery, 161–170.

[104] **Riera**, **M.**, **Arnau**, **J.**, and **Gonzalez**, **A.** (2018). Computation Reuse in DNNs by Exploiting Input Similarity. In: *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pp. 57–68.

[105] **Courbariaux**, **M.** and **Bengio**, **Y.** (2016). BinaryNet: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1. In: *CoRR* abs/1602.02830. arXiv: 1602.02830.

[106] **Jacob**, **B.**, **Kligys**, **S.**, **Chen**, **B.**, **Zhu**, **M.**, **Tang**, **M.**, **Howard**, **A. G.**, **Adam**, **H.**, and **Kalenichenko**, **D.** (2017). Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. In: *CoRR* abs/1712.05877. arXiv: 1712.05877.

[107] **Wu**, **J.**, **Leng**, **C.**, **Wang**, **Y.**, **Hu**, **Q.**, and **Cheng**, **J.** (2015). Quantized Convolutional Neural Networks for Mobile Devices. In: *CoRR* abs/1512.06473. arXiv: 1512.06473.

[108] **Ueyoshi**, **K.**, **Ando**, **K.**, **Hirose**, **K.**, **Takamaeda-Yamazaki**, **S.**, **Kadomoto**, **J.**, **Miyata**, **T.**, **Hamada**, **M.**, **Kuroda**, **T.**, and **Motomura**, **M.** (2018). QUEST: A 7.49TOPS multi-purpose log-quantized DNN inference engine stacked on 96MB 3D SRAM using inductive-coupling technology in 40nm CMOS. In: *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*, pp. 216–218.

[109] **Laurenciu**, **N. C.** and **Cotofana**, **S. D.** (2015). Low cost and energy, thermal noise driven, probability modulated random number generator. In: *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, pp. 2724–2727.

[110] **Fischer**, **V.** and **Drutarovskỳ**, **M.** (2002). True random number generator embedded in reconfigurable hardware. In: *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, pp. 415–430.

[111] **Brown**, **J.**, **Gao**, **R.**, **Ji**, **Z.**, **Chen**, **J.**, **Wu**, **J.**, **Zhang**, **J.**, **Zhou**, **B.**, **Shi**, **Q.**, **Crowford**, **J.**, and **Zhang**, **W.** (2018). A low-power and high-speed True Random Number Generator using generated RTN. In: *2018 IEEE Symposium on VLSI Technology*. IEEE, pp. 95–96.

[112] **Vasyltsov**, **I.**, **Hambardzumyan**, **E.**, **Kim**, **Y.-S.**, and **Karpinskyy**, **B.** (2008). Fast digital TRNG based on metastable ring oscillator. In: *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, pp. 164–180.

[113] **Salami.**, **B.** (2018). FPGA BRAMs Undervolting Study. https://github.com/behzadsalami/FPGA-BRAMs-Undervoltig-Study.

[114] **Reagen**, **B.**, **Gupta**, **U.**, **Pentecost**, **L.**, **Whatmough**, **P.**, **Lee**, **S. K.**, **Mulholland**, **N.**, **Brooks**, **D.**, and **Wei**, **G.** (2018). Ares: A framework for quantifying the resilience of deep neural networks. In: *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pp. 1–6.

[115] **Jia**, **Y.**, **Shelhamer**, **E.**, **Donahue**, **J.**, **Karayev**, **S.**, **Long**, **J.**, **Girshick**, **R.**, **Guadarrama**, **S.**, and **Darrell**, **T.** (2014). Caffe: Convolutional Architecture for Fast Feature Embedding. In: *Proceedings of the 22nd ACM International Conference on Multimedia*. MM '14. Orlando, Florida, USA: Association for Computing Machinery, 675–678.

[116] **Milde**, **M. B.**, **Neil**, **D.**, **Aimar**, **A.**, **Delbruck**, **T.**, and **Indiveri**, **G.** (2017). ADaPTION: Toolbox and benchmark for training convolutional neural networks with reduced numerical precision weights and activation. In: *arXiv preprint arXiv:1711.04713*.

[117] **NVIDIA.** (2017). NVIDIA-caffe extension. https://github.com/NVIDIA/caffe.

[118] **Micikevicius**, **P.**, **Narang**, **S.**, **Alben**, **J.**, **Diamos**, **G.**, **Elsen**, **E.**, **Garcia**, **D.**, **Ginsburg**, **B.**, **Houston**, **M.**, **Kuchaiev**, **O.**, **Venkatesh**,

G., et al. (2017). Mixed precision training. In: *arXiv preprint arXiv:1710.03740*.

[119] **Markidis**, **S.**, **Der Chien**, **S. W.**, **Laure**, **E.**, **Peng**, **I. B.**, and **Vetter**, **J. S.** (2018). Nvidia tensor core programmability, performance & precision. In: *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, pp. 522–531.

[120] **Rakin**, **A. S.**, **He**, **Z.**, **Li**, **J.**, **Yao**, **F.**, **Chakrabarti**, **C.**, and **Fan**, **D.** (2021). T-BFA: Targeted Bit-Flip Adversarial Weight Attack. arXiv: 2007.12336 [cs.LG].

[121] **He**, **Z.**, **Rakin**, **A. S.**, **Li**, **J.**, **Chakrabarti**, **C.**, and **Fan**, **D.** (2020). Defending and Harnessing the Bit-Flip Based Adversarial Weight Attack. In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 14083–14091.

[122] **Rakin**, **A. S.**, **He**, **Z.**, and **Fan**, **D.** (2019). Bit-Flip Attack: Crushing Neural Network with Progressive Bit Search. arXiv: 1903.12269 [cs.CV].

[123] **Liu**, **Y.**, **Wei**, **L.**, **Luo**, **B.**, and **Xu**, **Q.** (2017). Fault injection attack on deep neural network. In: *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 131–138.

[124] **Kelly**, **M. S.**, **Mayes**, **K.**, and **Walker**, **J. F.** (2017). Characterising a CPU fault attack model via run-time data analysis. In: *2017 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pp. 79–84.

[125] **Li**, **G.**, **Hari**, **S. K. S.**, **Sullivan**, **M.**, **Tsai**, **T.**, **Pattabiraman**, **K.**, **Emer**, **J.**, and **Keckler**, **S. W.** (2017b). Understanding Error Propagation in Deep Learning Neural Network (DNN) Accelerators and Applications. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC '17. Denver, Colorado: Association for Computing Machinery.

[126] **Neggaz**, **M. A.**, **Alouani**, **I.**, **Lorenzo**, **P. R.**, and **Niar**, **S.** (2018). A Reliability Study on CNNs for Critical Embedded Systems. In: *2018 IEEE 36th International Conference on Computer Design (ICCD)*, pp. 476–479.

[127] **Salavati**, **A. H.** and **Karbasi**, **A.** (2012). Multi-level error-resilient neural networks. In: *2012 IEEE International Symposium on Information Theory Proceedings*, pp. 1064–1068.

[128] **Stutz**, **D.**, **Chandramoorthy**, **N.**, **Hein**, **M.**, and **Schiele**, **B.** (2020). Bit Error Robustness for Energy-Efficient DNN Accelerators. arXiv: 2006.13977 [cs.LG].

[129] **Ganapathy**, **S.**, **Kalamatianos**, **J.**, **Kasprak**, **K.**, and **Raasch**, **S.** (2017). On characterizing near-threshold SRAM failures in FinFET technology. In: *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–6.

[130] **Givaki**, **K.**, **Salami**, **B.**, **Hojabr**, **R.**, **Tayaranian**, **S. R.**, **Khonsari**, **A.**, **Rahmati**, **D.**, **Gorgin**, **S.**, **Cristal**, **A.**, and **Unsal**, **O. S.** (2020). On the Resilience of Deep Learning for Reduced-voltage FPGAs. In: *2020 28th Euromicro International Conference on*

Parallel, Distributed and Network-Based Processing (PDP). IEEE, pp. 110–117.

[131] **Aysu**, **A.**, **Gulcan**, **E.**, **Moriyama**, **D.**, **Schaumont**, **P.**, and **Yung**, **M.** (2015). End-to-end design of a PUF-based privacy preserving authentication protocol. In: *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, pp. 556–576.

[132] **Chatterjee**, **U.**, **Chakraborty**, **R. S.**, and **Mukhopadhyay**, **D.** (2017). A PUF-based secure communication protocol for IoT. In: *ACM Transactions on Embedded Computing Systems (TECS)* 16.3, pp. 1–25.

[133] **Che**, **W.**, **Deng**, **H.**, **Tan**, **W.**, and **Wang**, **J.** (2008). A random number generator for application in RFID tags. In: *Networked RFID systems and lightweight cryptography*. Springer, pp. 279–287.

[134] **Chamon**, **C.**, **Ferdous**, **S.**, and **Kish**, **L. B.** (2021). Deterministic Random Number Generator Attack Against the Kirchhoff-Law-Johnson-Noise Secure Key Exchange Protocol. In: *Fluctuation and Noise Letters*, p. 2150046.

[135] **Šimka**, **M.** and **Komenského**, **P.** (2006). Active non-invasive attack on true random number generator. In: *PhD Student Conference and Scientific and Technical Competition of Students of FEI TU Košice, Košice, Slovakia*. Citeseer.

[136] **Qin**, **Y.**, **Sheng**, **Q. Z.**, **Falkner**, **N. J.**, **Dustdar**, **S.**, **Wang**, **H.**, and **Vasilakos**, **A. V.** (2016). When things matter: A survey on data-centric internet of things. In: *Journal of Network and Computer Applications* 64, pp. 137–153.

[137] **Schreckling**, **D.**, **Köstler**, **J.**, and **Schaff**, **M.** (2013). Kynoid: real-time enforcement of fine-grained, user-defined, and data-centric security policies for android. In: *information security technical report* 17.3, pp. 71–80.

[138] **Shao**, **M.**, **Zhu**, **S.**, **Zhang**, **W.**, **Cao**, **G.**, and **Yang**, **Y.** (2008). pDCS: Security and privacy support for data-centric sensor networks. In: *IEEE Transactions on Mobile Computing* 8.8, pp. 1023–1038.

[139] **Liu**, **D.**, **Liu**, **Z.**, **Li**, **L.**, and **Zou**, **X.** (2016). A low-cost low-power ring oscillator-based truly random number generator for encryption on smart cards. In: *IEEE Transactions on Circuits and Systems II: Express Briefs* 63.6, pp. 608–612.

[140] **Avaroğlu**, **E.**, **Tuncer**, **T.**, **Özer**, **A. B.**, **Ergen**, **B.**, and **Türk**, **M.** (2015). A novel chaos-based post-processing for TRNG. In: *Nonlinear Dynamics* 81.1, pp. 189–199.

[141] **Galajda**, **M. D.** (2006). Chaos-based true random number generator embedded in a mixed-signal reconfigurable hardware. In: *Journal of Electrical Engineering* 57.4, pp. 218–225.

[142] **Drutarovsky**, **M.** and **Galajda**, **P.** (2007). A robust chaos-based true random number generator embedded in reconfigurable switched-capacitor hardware. In: *2007 17th International Conference Radioelektronika*. IEEE, pp. 1–6.

[143] **Danger**, **J.-L.**, **Guilley**, **S.**, and **Hoogvorst**, **P.** (2009). High speed true random number generator based on open loop structures in FPGAs. In: *Microelectronics journal* 40.11, pp. 1650–1656.

[144] **Grujić**, **M.**, **Rožić**, **V.**, **Yang**, **B.**, and **Verbauwhede**, **I.** (2018). A closer look at the delay-chain based trng. In: *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, pp. 1–5.

[145] **Rashid**, **M. I.**, **Ferdaus**, **F.**, **Talukder**, **B. B.**, **Henny**, **P.**, **Beal**, **A. N.**, and **Rahman**, **M. T.** (2020). True Random Number Generation Using Latency Variations of FRAM. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 29.1, pp. 14–23.

[146] **Ferdaus**, **F.**, **Talukder**, **B. B.**, **Sadi**, **M.**, and **Rahman**, **M. T.** (2021). True Random Number Generation using Latency Variations of Commercial MRAM Chips. In: *2021 22nd International Symposium on Quality Electronic Design (ISQED)*. IEEE, pp. 510–515.

[147] **Cambou**, **B.**, **Telesca**, **D.**, **Assiri**, **S.**, **Garrett**, **M.**, **Jain**, **S.**, and **Partridge**, **M.** (2021). TRNGs from Pre-Formed ReRAM Arrays. In: *Cryptography* 5.1, p. 8.

[148] **Bond**, **M.**, **Choudary**, **O.**, **Murdoch**, **S. J.**, **Skorobogatov**, **S.**, and **Anderson**, **R.** (2014). Chip and Skim: cloning EMV cards with the pre-play attack. In: *2014 IEEE Symposium on Security and Privacy*. IEEE, pp. 49–64.

[149] **Bauke**, **H.** and **Mertens**, **S.** (2007). Random numbers for large-scale distributed Monte Carlo simulations. In: *Physical Review E* 75.6, p. 066701.

[150] **Xilinx** (n.d.[b]). Xilinx Zynq-7000 SoC (Z-7007S, Z-7012S, Z-7014S, Z-7010, Z-7015, and Z-7020): DC and AC Switching Characteristics. `https://www.xilinx.com/support/documentation/data_sheets/ds187-XC7Z010-XC7Z020-Data-Sheet.pdf`.

[151] **PMBus** (n.d.). PMBus Specification. `http://pmbus.org/`.

[152] **Yazdanshenas**, **S.**, **Tatsumura**, **K.**, and **Betz**, **V.** (2017). Don't forget the memory: Automatic block RAM modelling, optimization, and architecture exploration. In: *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 115–124.

[153] **Chen**, **Q.**, **Mahmoodi**, **H.**, **Bhunia**, **S.**, and **Roy**, **K.** (2005). Modeling and testing of SRAM for new failure mechanisms due to process variations in nanoscale CMOS. In: *23rd IEEE VLSI Test Symposium (VTS'05)*. IEEE, pp. 292–297.

[154] **Guo**, **Z.**, **Carlson**, **A.**, **Pang**, **L.-T.**, **Duong**, **K. T.**, **Liu**, **T.-J. K.**, and **Nikolic**, **B.** (2009). Large-scale SRAM variability characterization in 45 nm CMOS. In: *IEEE Journal of Solid-State Circuits* 44.11, pp. 3174–3192.

[155] **Kim**, **D.**, **Chandra**, **V.**, **Aitken**, **R.**, **Blaauw**, **D.**, and **Sylvester**, **D.** (2011). Variation-aware static and dynamic writability analysis for voltage-scaled bit-interleaved 8-T SRAMs. In: *IEEE/ACM International Symposium on Low Power Electronics and Design*. IEEE, pp. 145–150.

[156] **Shannon**, **C. E.** (2001). A mathematical theory of communication. In: *ACM SIGMOBILE mobile computing and communications review* 5.1, pp. 3–55.

[157] **Larimi**, **S. S. N.**, **Salami**, **B.**, **Unsal**, **O. S.**, **Kestelman**, **A. C.**, **Sarbazi-Azad**, **H.**, and **Mutlu**, **O.** (2021). Understanding Power Consumption and Reliability of High-Bandwidth Memory with Voltage Underscaling. In: *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, pp. 517–522.

[158] **Brooks**, **S.** and **Cicchetti**, **A.** (2014). Design of a Low Power Latch Based SRAM Sense Amplifier. In: *Worcester Polytechnic Institute*.

[159] **Do**, **A.-T.**, **Kong**, **Z.-H.**, **Yeo**, **K.-S.**, and **Low**, **J. Y. S.** (2009). Design and sensitivity analysis of a new current-mode sense amplifier for low-power SRAM. In: *IEEE transactions on very large scale integration (VLSI) systems* 19.2, pp. 196–204.

[160] **Arslan**, **U.**, **McCartney**, **M. P.**, **Bhargava**, **M.**, **Li**, **X.**, **Mai**, **K.**, and **Pileggi**, **L. T.** (2008). Variation-tolerant SRAM sense-amplifier timing using configurable replica bitlines. In: *2008 IEEE Custom Integrated Circuits Conference*. IEEE, pp. 415–418.

[161] **Karl**, **E.**, **Sylvester**, **D.**, and **Blaauw**, **D.** (2005). Timing error correction techniques for voltage-scalable on-chip memories. In: *2005 IEEE International Symposium on Circuits and Systems*. IEEE, pp. 3563–3566.

[162] **Bhargava**, **M.**, **Sheikh**, **K.**, and **Mai**, **K.** (2015). Robust true random number generator using hot-carrier injection balanced metastable sense amplifiers. In: *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, pp. 7–13.

[163] **Kammoun**, **M.**, **Elleuchi**, **M.**, **Abid**, **M.**, and **BenSaleh**, **M. S.** (2020). FPGA-based implementation of the SHA-256 hash algorithm. In: *2020 IEEE International Conference on Design & Test of Integrated Micro & Nano-Systems (DTS)*. IEEE, pp. 1–6.

[164] **Guilford**, **J.**, **Yap**, **K.**, and **Gopal**, **V.** (2012). Fast SHA-256 implementations on Intel architecture processors. In: *IA Architects*.

[165] **minio** (n.d.). Accelerate SHA256 computations. `https://github.com/minio/sha256-simd`.

[166] **Muralimanohar**, **N.**, **Balasubramonian**, **R.**, and **Jouppi**, **N. P.** (2009). *CACTI 6.0: A Tool to Model Large Caches*. Tech. rep. HPL-2009-85. HP Laboratories.

[167] **Bostancı**, **F. N.**, **Olgun**, **A.**, **Orosa**, **L.**, **Yağlıkçı**, **A. G.**, **Kim**, **J. S.**, **Hassan**, **H.**, **Ergin**, **O.**, and **Mutlu**, **O.** (2022). DR-STRaNGe: End-to-End System Design for DRAM-based True Random Number Generators. In: *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, pp. 1141–1155.

[168] **AMD** (n.d.). AMD Random Number Generator, howpublished = `https://www.amd.com/system/files/TechDocs/amd-random-number-generator.pdf`.

[169] **ARM** (n.d.). Arm® True Random Number Generator Technical Reference Manual, howpublished = `https://documentation-service.arm.com/static/5f22d7d9f3ce30357bc2b392`.

[170] **Intel** (n.d.). Intel® Digital Random Number Generator (DRNG) Software Implementation Guide, howpublished = `https://www.intel.com/content/dam/develop/external/us/en/documents/drng-software-implementation-guide-2-1-185467.pdf`.

[171] **Sutar**, **S.**, **Raha**, **A.**, and **Raghunathan**, **V.** (2016). D-PUF: An intrinsically reconfigurable DRAM PUF for device authentication in embedded systems. In: *2016 International Conference on Compliers, Architectures, and Sythesis of Embedded Systems (CASES)*. IEEE, pp. 1–10.

[172] **Keller**, **C.**, **Gürkaynak**, **F.**, **Kaeslin**, **H.**, and **Felber**, **N.** (2014). Dynamic memory-based physically unclonable function for the generation of unique identifiers and true random numbers. In: *2014 IEEE international symposium on circuits and systems (ISCAS)*. IEEE, pp. 2740–2743.

[173] **Eckert**, **C.**, **Tehranipoor**, **F.**, and **Chandy**, **J. A.** (2017). DRNG: DRAM-based random number generation using its startup value behavior. In: *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*. IEEE, pp. 1260–1263.

[174] **Tehranipoor**, **F.**, **Yan**, **W.**, and **Chandy**, **J. A.** (2016). Robust hardware true random number generators using DRAM remanence effects. In: *2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, pp. 79–84.

[175] **Ray**, **B.** and **Milenković**, **A.** (2018). True random number generation using read noise of flash memory cells. In: *IEEE transactions on electron devices* 65.3, pp. 963–969.