
**Gamze AKSU**

**171180005**

**GAZİ ÜNİVERSİTESİ**
**MÜHENDİSLİK FAKÜLTESİ**
**BİLGİSAYAR MÜHENDİSLİĞİ**

**BM311 BİLGİSAYAR MİMARİSİ**

**M.ALİ AKCAYOL**

# Hata Toleranslı Mikroişlemci Tabanlı Sistemler
# (FAULT-TOLERANT MICROPROCESSER-BASED SYSTEM)

**ARALIK 2020**

# ÖZET

Hata toleranslı sistemler sistemde hatalar oluşsa bile belirlenen görevini yerine getirebilir. Bunu yapabilmesi için hataların özelliklerinin bilinmesi gerekir. Hataların kaynağı, doğası, süresi, kapsamı ve değeri gibi özellikler örnek verilebilir. Hataların teloransı için çeşitli teknikler vardır. Bu teknikler bilgi yedekliliği, donanım yedekliliği, zaman yedekliliği ve yazılım yedekliliği olarak ayrılabilir.

Bu araştırma yazısında hata toleransı kısaca açıklanmış, hata toleranslı sistemlerin nerelerde kullanıldığından bahsedilmiştir. Daha sonra sistemlerdeki güvenilirlikten bu güvenilirliğin sağlanması için dört ana teknikten bahsedilmiştir. Ek olarak güvenilirliğin hesaplanması için iki ana ölçek anlatılmıştır. Daha sonra hata toleranslı sistemler geliştirilmesi için üç ana konudan bahsedilmiştir. Bu konulardan iki tanesi bu araştırma yazısında açıklanmıştır. Bunlar hataların özellikleri ve karakterizasyonu ve hata toleransı için tekniklerdir. Hataların özellikleri altında hataların kaynaklarından (spesifikasyon hataları, implementasyon hataları(, bileşen hataları, harici hataları) bahsedilmiştir. Hataların süresinden kalıcı, geçici ve aralıklı olarak ayrılmış şekilde anlatılmıştır. Hataların kapsamı ve değerinden de bahsedilmiştir. Daha sonra hata tolerans teknikleri uzunca anlatılmıştır ve hata toleranslı sistemler (SIFT, Agusta 129, Tandem 16 Nonstop, Stratus ve MARS sistemleri ) anlatılmıştır.

Bu araştırma yazısı hazırlanırken Johnson, Koren ve Krishna, Avizienis ve Dubrova araştırmalarına dayanarak hazırlanmıştır.

## Hata Toleranslı Mikroişlemci Tabanlı Sistemler

Hata toleransı bir sistemdeki bileşenlerin biri ya da daha fazlası arızalanması durumunda bile sistemin aynı şekilde çalışmayı sürdürmesine denir. Yani hata toleransı işletim sistemin yazılımdaki veya donanımdaki işlev bozukluklarına ve hatalarına nasıl tepki verdiğini ifade eder. Hataya dayanıklı bir sistem, bireysel donanım veya yazılım bileşenlerindeki hataları, güç kesintilerini veya diğer beklenmedik sorunları ele alabilmeli ve yine de teknik özelliklerini karşılayabilmelidir. Sistemde hatalar oluştukça sistemin kalitesi düşebilir. Bu düşüş sistemdeki hataların şiddetiyle doğru orantılıdır. Bu demektir ki, hata toleransı olan bir sistemde sistemin bir kısmı arızalandığında tamamen başarısız olmak yerine daha düşük bir kalitede amaçlanan işlevine devam eder. [1]

Hata toleransının nihai amacı, güvenilir bir sistemin geliştirilmesidir. Geniş anlamda, güvenilirlik, bir sistemin amaçlanan hizmet seviyesini kullanıcılarına sunma yeteneğidir. Bilgi işlem her yerde yaygınlaştıkça ve günlük yaşamımıza her ölçekte nüfuz ettikçe, güvenilirlik yalnızca geleneksel güvenlik, görev ve iş açısından kritik uygulamalar için değil, aynı zamanda bir bütün olarak toplumumuz için de önemli hale gelir.

Yarı iletken teknolojisinin ilerlemesiyle birlikte donanımlar daha güvenilir hale geldiler ve genel amaçlı bilgisayarlar için hata tolere etme ihtiyacı azaldı. Bununla birlikte güvenlik, görev ve iş açısından kritik sistemler tasarlanması gerektiğinde hala hata toleransı önemlidir. Güvenlik açısından kritik (Safety-critical) sistemlere örnek verecek olursak nükleer enerji santrali kontrol sistemleri, bilgisayar kontrollü radyasyon tedavisi makineleri, kalp pilleri ve uçuş kontrol sistemleri örnek olarak verilebilir. Görev açısından kritik (Mission-critical) sistemler bir uzay aracı veya uydularda olduğu gibi görevin bitmesi, tamamlanması önemlidir. İş açısından kritik (Business-critical) bir işletmenin sürekli çalışmasını sağlamanın sorun olduğu uygulamalardır. Örnekler, banka ve borsa otomatik ticaret sistemleri, Web sunucuları ve e-ticarettir. [3]

Sistem karmaşıklığı arttıkça sistemdeki güvenilirlik de azalır. Elena Dubrova'nın "Fault-Tolerant Design" kitabında şu örnek verilmiştir. 100 yedekli olmayan bileşenden oluşan ve tek tek güvenirliği %99,99 olan bileşenlerden oluşan bir sistemin güvenirliği %99,01 iken bunu aynı %99,99 bileşen güvenirliği ama daha fazla mesela 10.000 yedekli olmayan bileşenden oluşan bir sistem oluşturduğumuzda sistemin güvenirliği %36,79 olur. Bu güvenilirlik oranı çoğu uygulamada kabul edilemezdir. 10.000 yedeksiz bileşenden oluşan bir sistemde %99 güvenilirlik olabilmesi için sistemdeki her bir bileşenin güvenilirliğinin %99,999

olması gerekir. Bu da sistemin maliyetinin artmasına neden olur. Kusursuz bir sistem oluşturmak neredeyse imkânsızdır. Tasarımcılar tüm donanım ve yazılım hatalarını düzeltmek için ellerinden gelen her şeyi yapsalar bile bazı beklenmedik sorunlar her zaman ortaya çıkar. Bu yüzden sistemin kusursuz bir şekilde tasarlandığı ve uygulandığı durumlarda bile hataların tasarımcıların kontrolü dışında oluşması çok olasıdır.

Hataları engellemek, sistemin performansını arttırmak için dört ana teknik vardır. Bunlar hatadan önleme (fault avoidance), hata maskeleme (fault masking), hata toleransı (fault tolerance), hata tahmini (error forecasting) [2]. Hata önleme hataların nedenlerinin önlemeye çalışılmasıdır. Tasarım incelenmesi, bileşen taranması ve sistem testi gibi işlemler örnek olarak gösterilebilir. Hata maskeleme, hataların yayılmasını önleyen gerçek zamanlı bir işlemdir. Hata toleransı ise sistemdeki hatalara rağmen görevlerini yerine getirme yeteneğidir. Örnek olarak yedekleme verilebilir. Bir sistem çökmesini önlemek için yedek kullanılır. Hata tahmini ise hataların oluşması ve sonuçları hakkında hesaplamalar yapılarak tahmin edilmesine denir.

İki durum arasındaki işlemler başarısızlık ve yenileme arasında bir döngüde devam eder. bu iki durumun değişimini ölçmek için iki ana güvenilebilirlik ölçüsü vardır [2]. Bunlardan biri güvenilirlik (Reliability) ilk andan itibaren sürekli olarak sistemin çalışma başarısının bir ölçüsüdür. Bir diğeri kullanılabilirlik (Availability), başarı ve kesintinin değişimine göre hizmet başarısının bir ölçüsüdür.

Aligardas AVIZIENIS bir tasarımcı ya da araştırmacının bir hata toleranslı bir sistem geliştirmek için üç ana konuyla ilgilenmeli gerektiğinden bahsediyor [5]. Bunlar:

1) tolere edilecek hatalar dizisinin sistematik olarak tanımlanması ve karakterizasyonu;
2) tespit edilen hatalara karşı koruma sağlayan yedekleme tekniklerinin geliştirilmesi ve seçimi;
3) yedekleme tekniklerinin etkililiğinin analitik veya deneysel tahmini

**Hataların Özellikleri**

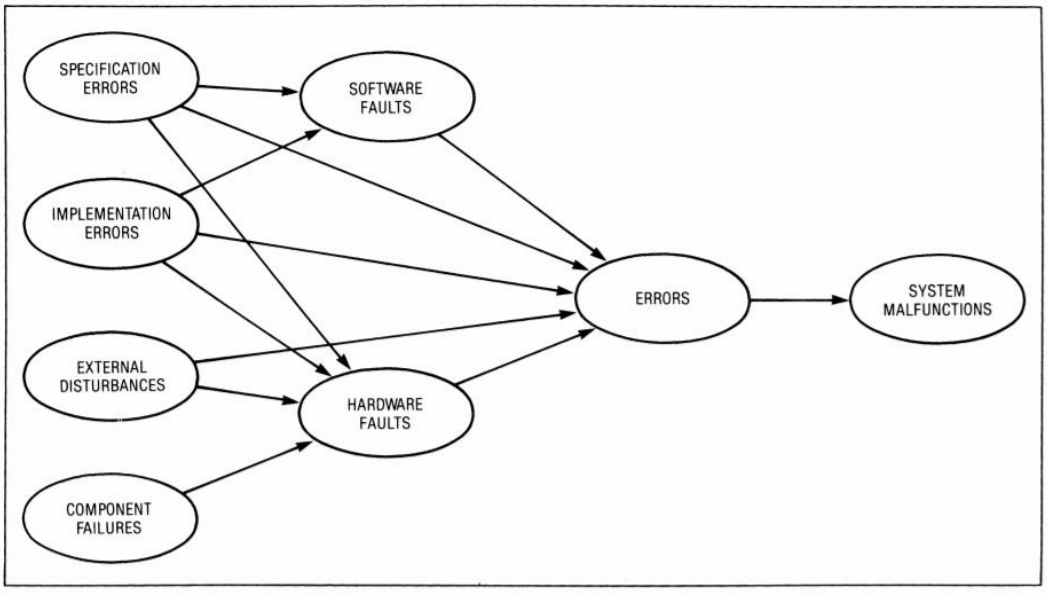
Hataların tespit edilebilmesi için hataların özelliklerinin bilinmesi gerekir. Bu özelliklerin en önemlisi hatanın doğasıdır. Bu hataların yazılım hatası mı donanım hatası mı olduğunun ayırt edilmesi olabilir. Örneğin güç kesintisi donanımsal bir hataya neden olur. Bir diğer özellik ise hataların kaynağıdır. Sistemlerde gerçekleşen hatalar birçok nedenden kaynaklanabilir. Hataların kaynağına inecek olursak 4 grupta inceleyebiliriz. Bunlar spesifikasyon hataları(specification fauls), implementasyon hataları(implementation faults), bileşen hataları(component faults), harici hataları(external faults)'dır. [4]

Spesifikasyon hataları algoritmik hatalardan ve mimari hatalardan veya gereksinimlerden kaynaklanır. Bu hataya tipik bir örnek olarak spesifikasyon

gereksinimlerinin sistemin çalıştığı ortamın özelliklerini göz ardı ettiği durumlar verilebilir. Spesifikasyon hataları en sık görülen donanım ve yazılım hatalarından biridir.

Bir diğer hata ise implementasyon hatalarıdır. Tasarım hataları(design faults) olarak da bilinirler. Genellikle implementasyon hataları sistem implementasyonundaki yanlış uygulamalardan kaynaklanır. Donanımsal olarak implementasyon hatalarına zayıf bileşen seçimi, mantıksal hatalar ve zayıf senkronizasyonu örnek olarak verebiliriz. Yazılımsal olarak program kodunda hata olması ya da zayıf yazılımlı bileşenlerin tekrar tekrar kullanılması da uygulama hatalarıdır. *Ariane 5 roket kazası yeniden kullanılan bir yazılım bileşeninin neden olduğu bir arıza örneğidir. Ariane 5 yazılım hatası nedeniyle 4 Haziran 1996'da roket havalandıktan 37 saniye sonra patladı. Bu sorun, 64 bitlik ondalıklı değerlerin 16 bitlik bir tamsayıya dönüştürülmesi sonucu meydana geldi. Değerlerin taşması sonucu, bilgisayar belleğini temizledi ve bu bellek temizlenmesi işlemi roket tarafından kendini patlatma talimatı olarak yorumladı [6].*

Bileşen hataları donanım hatalarının en sık görülen kaynağıdır. Bu bileşen kusurları üretim kusurlarını, rastgele cihaz kusurlarını ve bileşenlerin aşınmaları durumlarıdır. Bileşenlerin düşük güvenilirliği nedeniyle bilgi işlem sistemlerinde hata toleransı teknikleri ilk olarak bu tip kusurlar için kullanılmıştır. Yarı iletken teknolojisinin gelişmesinin ardından, donanım bileşenlerinde meydana gelen bileşen hataları azalarak daha güvenilir hale gelip fabrikasyon kusurlarının neden olduğu hataların yüzdesi büyük ölçüde düşmüş oldu.



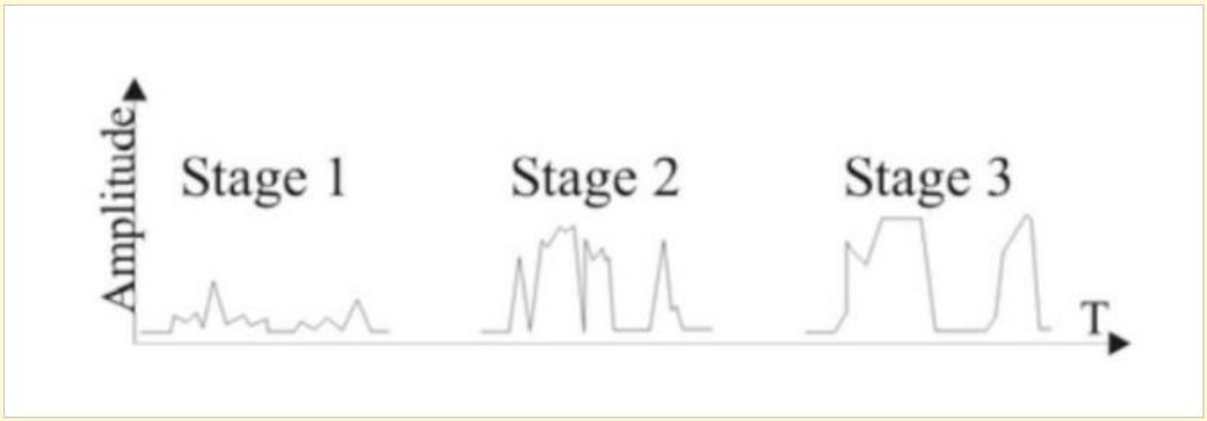

Şekil 1: Hatalar arasındaki ilişki

Son hata kaynağı da dış faktörlerden kaynaklanan harici hatalardır. Bu hataların kaynağı radyasyon, elektromanyetik girişim, savaş hasarı, kullanıcı hataları ve çevresel

aşırılıklar olarak verilebilir. Bu faktörler sistemin çalışmasını doğrudan etkiler. Örneğin radyasyon hafızadaki bir hücrenin değerinin değişmesini sağlar. Kullanıcıların neden olduğu hatalara örnek olarak kullanıcının uygulamaya girdiği yanlış değerler verilebilir. Virüsler ve hackerlerin izinsiz girişi de bu grup altında verilebilir.

Hataların özelliklerinden biri süresidir. Hataların süresine göre ayırmamız gerektiğinde 2 gruba bölebiliriz. Geçici (transient), kalıcı (permanent). Geçici hatalar, sistemde kısa süre için etkin kalırlar. Genellikle dış faktörler nedeniyle ortaya çıkarlar. Kurtarma için geçici hata azalıncaya kadar bekleyip bitince verilerin doğru halinin yeniden kurulması gerekir. Genellikle verilerin doğru hali rollback noktaları ya da kontrol noktaları ile sağlanır. Bir geçici hata için maksimum süre belirlidir. Bu maksimum süreden daha fazla süren hatalar kurtarma algoritmaları tarafından kalıcı hata olarak anlaşılır. Belleklerdeki baskın hata türleridir. Örneğin Random Access Memory (RAM)'deki hataların yaklaşık %98'i geçici hatalardan oluşur. Geçici hatalar genellikle çevresel etmenler sonucu oluşur. Bu etkiler arasına örnek olarak alfa parçacıkları, atmosferik nötronlar, elektrostatik deşarj, elektriksel güç düşüşleri veya aşırı ısınma verilebilir. Geçici hatalar sisteme kalıcı hata vermese bile sistemin çökmesine neden olabilir.

Geçici hataların altında aralıklı hatalar (intermittent faults) da incelenebilir. Sadece belirli zamanlarda belirli koşullar altında çıkar. Bu hatalar implementasyon kusurlarından, eskimeden ve yıpranmadan ve beklenmedik çalışma koşullarından kaynaklanabilir. Örneğin sadece titreşimde ortaya çıkan gevşek bir lehim bağlantısının sebep olduğu hatalar aralıklı hata olarak sayılır. En yıkıcı hatalardan sayılırlar çünkü tespit edilmeleri çok zordur. Çünkü örnekte de görüldüğü üzere başka bir hatadan sonucu olarak ortaya çıkabilirler ve çoklu bir hata durumu yaratabilirler. Çoğu aralıklı hatada sistemler kademeli olarak bozulur. Bu Şekil 1'de görülmektedir [7]. Bu hatalar nadiren onarılırlar. Genellikle kalıcı hatalara dönüşürler. Yani bir aralıklı hata oluşumu bir kalıcı hatanın başlangıcıdır, denilebilir.




Şekil 2:Bir cihazın hayatı boyunca geçirdiği aralıklı hataların genlikleri

Bir kalıcı hata sistemdeki bir veya daha fazla bileşendeki kalıcı hasarlardan oluşur. Birçok sebepten meydana gelebilirler örneğin üretim hataları, yaşlanma, radyasyon gibi etkiler kalıcı hatalara neden olabilir ama genel olarak kalıcı hataların sebebi fiziksel hatalardır. Bu hatalar genellikle kısa devre olduğunda, devrede kopuk bağlantılar olduğunda ya da bellekteki bir bitin aynı değere takılı kaldığında ortaya çıkar. Kurtarma için düzeltici bir eyleme ihtiyaç duyar. Kurtarma için yedek parçaların olması gerekir ya da arızalanan parça olmadan çalışması için bir provizyon (bir koşul hüküm karşılık) olması gerekir.

Hataların bir diğer tanımlayıcı özelliği ise kapsamıdır. Kapsam hatanın donanımda aynı anda kaç mantık değişkenini etkilediğini belirtir. Yerel hatalar (local faults) ve dağıtılmış hatalar (distrubuted faults) olarak ikiye ayrılabilir. Yerel hatalar tek bir mantık değişkenini etkileyen hatalardır. Dağıtılmış hatalar ise birden fazla değişkenin etkilendiği bir modülü veya tüm sistemi etkileyebilen hatalardır. Büyük ölçekli entegresyan (Large Scale Integration - LSI) devrelerinde mantık elemanlarının bir birine fiziksel olarak yakın olması dağıtılmış hataların oluşma olasılığını arttırır. Dağıtılmış hatalar bazen sistemi etkileyen tek bir kritik bileşende oluşan hatadan da kaynaklanabilir. Örneğin saatler, güç kaynakları, veri yolları bu kritik bileşenler arasına girebilir.

Hataların son tanımlayıcı özelliği değeridir (value). Hataların değeri belirli ya da belirsiz olabilir. Belirli değeri olan hatalarda bit hata süresi boyunca aynı hatada takılı kalmıştır. Bu "0" ya da "1" olabilir ama tasarım özelliklerinin tersi bir değerdedir. Belirsiz bir değeri olan hatalarda ise bitin değeri değişkenlik gösterir.

**Hata tolerans teknikleri**

Hata toleransı dört ana aşamadan oluşur. Bu aşamalar hata tespiti (fault detection), hata konumunu belirleme (fault location), hata çevreleme (fault containment), hatayı giderme (fault recovery) olarak sıralanabilir. Hata tespiti, sistemde bir hatanın gerçekleştiğini fark etme yeteneğidir. Bir kurtarma mekanizmasının çalışması için genellikle önce hatanın tespit edilmesi gereklidir. Herhangi bir kurtarma mekanizmasının çalışabilmesi için hatanın nerede meydana geldiğinin bilinmesi gereklidir. Hata çevrelemenin amacı hatanın yayılmasını engellemektir. Hata izole edilir ve tüm sistemi etkilemesini önlemeye çalışılır. Kurtarma işlemi için hatanın çevrelenmesi zorunludur. Son olarak hatayı kurtarma sistem hataların varlığında bile çalışır durumda kalması yeteneğidir.[4]

Tüm hata tolerans tekniklerindeki başrol yedekleme (redundancy). Normal sistemin çalışması için gerekenin yani sıra bir de fazladan eklenen bilgi, kaynak veya zaman eklenmesidir. Fazlalık bilgi fazlalığı (information redundancy), donanım fazlalığı (hardware

redundancy), yazılım fazlalığı (software redundancy) ve zaman fazlalığıdır olarak gruplanabilir.

**Bilgi Fazlalığı (Information Redundancy):**

Bilgi fazlalığı bir işlemin yapılabilmesi için gerekenden daha fazla bilgi eklenmesidir. Bilgi fazlalığına en iyi örnek hata tespit kodlarıdır. Fazla bilgi ile geçerli ve geçersiz kod kelimelerinin ayırt edilmesi işlemi yapılır. Başka bir örnek olarak binary (ikili) kodlarda tek bir bitin değişmesi kodu değiştirebilir ama yine de geçerli olarak kalacaktır. Sonuç olarak alıcının kod sözcüğünün doğru olup olmadığını tamamen belirlemenin yolu yoktur. [9]

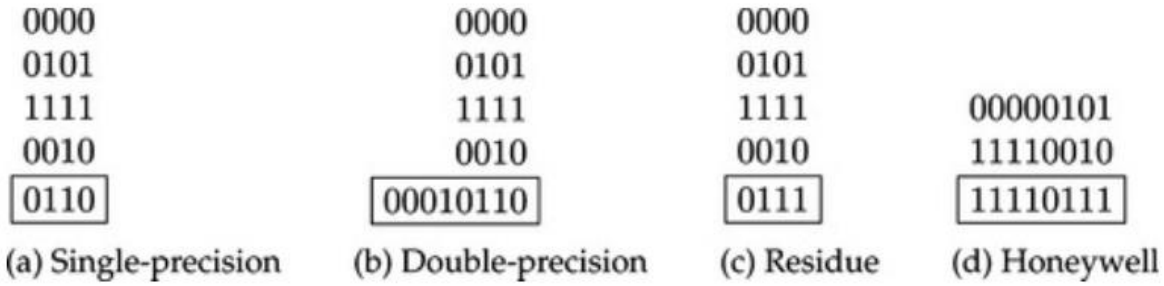
Hata tespit kodlarının belki de en basit şekli tek bit eşlik (single-bit parity) kontrolüdür. Orijinal veri bitlerine fazladan bitler eklenir. Koddaki birlerin sayısını çift ya da tek olmaya zorlar. Eşlik bitindeki birlerin toplamı tek sayı ise (odd parity) çift sayı ise (even parity) denir. Bu şekilde tek bitlik hataların tespiti yapılır.

Başka bir hata tespit kodu da M/N kodlarıdır (M-out-of-N Coding). Tek yönlü bir hata tespit kodudur. Tek yönlü hatalardan etkilenen tüm bitler 0'dan 1'e ya da 1'den 0'a değişir ama iki yönlü değişme olmaz tek yönlü değişmeler gerçekleşir. N bit kod sözcüğünün değeri 1 olan M tane biti vardır. Herhangi bir tek bitlik hata M sayısını değiştirir. Bu şekilde tek yönlü hataların tespiti gerçekleşir. M of N kodlamanın en basit örneği 2 of 5 kodlamadır. Bu ondalık basamakları kodlamaya yarar. 10 kod sözcüğünü ondalık basamağa dönüştürmenin 10 farklı yolu vardır. Tablo 1'de tek parity (odd parity) ve 2 out of 5 code örneği verilmiştir. Jet Propulsion Laboratory'da geliştirilen STAR (Self-Test and Repair) bilgisayarı, talimat işlenenlerini kodlamak için M-out-of-N kodlamasını kullandı. [4]

**Table 1.**
**Examples of parity and 2-out-of-5 coding.**

| DECIMAL DIGITS | BCD ODD PARITY | 2-out-of-5 CODE |
|---|---|---|
| 0 | 0 0 0 0 1 | 0 0 0 1 1 |
| 1 | 0 0 0 1 0 | 1 1 0 0 0 |
| 2 | 0 0 1 0 0 | 1 0 1 0 0 |
| 3 | 0 0 1 1 1 | 0 1 1 0 0 |
| 4 | 0 1 0 0 0 | 1 0 0 1 0 |
| 5 | 0 1 0 1 1 | 0 1 0 1 0 |
| 6 | 0 1 1 0 1 | 0 0 1 1 0 |
| 7 | 0 1 1 1 0 | 1 0 0 0 1 |
| 8 | 1 0 0 0 0 | 0 1 0 0 1 |
| 9 | 1 0 0 1 1 | 0 0 1 0 1 |

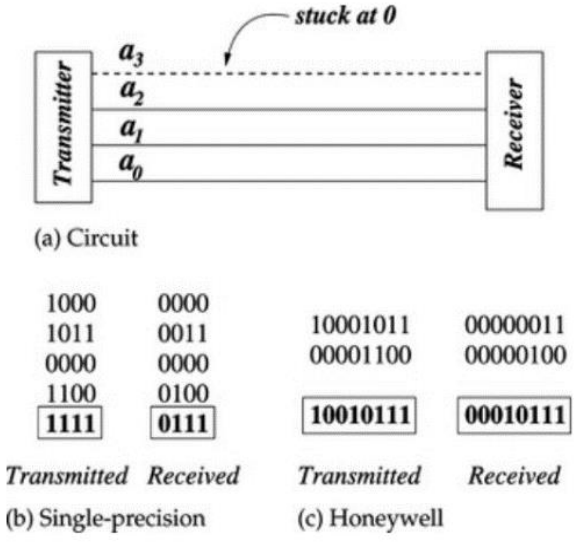
Başka bir hata saptama kodlama checksum'lardır. Veri blokları bir noktadan diğerine aktarılacağı zaman uygulanabilir. Veri iletimindeki hataları tespit eder. Checksum iletilen veri bloğunu toplar ve bu toplam sonucunu iletir aynı şekilde alıcı da veri bloğunu toplar ve toplamların doğruluğuna bakılır. Toplamlar uyuşmazsa hata ortaya çıkmış demektir. Birkaç çeşidi vardır. Bunlar tek duyarlılıklı checksum (single-precision checksum), çift duyarlılıklı checksum (double-precision checksum), residue checksum, Honeywell checksum [9]. Tek duyarlılıklı checksum iletilecek olan tüm bitlerin toplanması ve taşmaların atılması ile oluşturulur. Alım noktasında tekrar toplama yapılır ve karşılaştırma yapılır. Uyuşmazlıklar hata olduğu anlamına gelir.




Bir diğer checksum çeşidi ise çift duyarlılıklı checksumdır. Hata kapsamı çift hassasiyetle tek duyarlılıklı checksuma göre önemeli ölçüde geliştirilebilir. Çift duyarlılıklı checksumlarda çift duyarlılıklı toplama yapılır. Burada taşma gerçekleşmez.

Residue checksum, veriler için bir kalıntı hesaplayarak ve bunu verilere ekleyerek oluşturulan ayrılabilir aritmetik kodlardır. Kalıntı verilerin ondalık temsili olan bir tamsayıya karşılık gelen N sayısına bölünerek oluşturulur. taşıma toplamın en önemsiz bitine eklenir. Bu yüzden daha güvenilirdir.

Honeywell checksum, kelimeleri çiftler halinde birleştirir. Bu şekilde aynı konumda oluşan hatalara karşı koruma sağlanmış olur. Yandki Şekil 3'de a3 değerini taşıyan bit 0'da takılı kalmıştır [9]. Bir hata vardır ama tek duyarlılıklı checksum yapıldığında eşleşme görülür. Ancak honeywell cheksum'da bu hata tespit edilir.




Şekil 3:Honeywell ile tek duyarlıklı checksum

Hataların tespiti için son olarak da aritmetik kodları inceleyeceğiz. Aritmetik kodlar genellikle toplama ve çarpma gibi aritmetik işlemlerle hataları tespit etmeye çalışır. X bir veri kelimesini bir kod kelimesine A(X) dönüştürülür. Örneğin veri kelimeleri X ve Y, işlem gerçekleştirilmeden önce kodlanır. Ortaya çıkan kod sözcükleri A(X) ve A(Y) üzerinde işlem gerçekleştirilir.”*” herhangi bir işlem olmak üzere A(X)*A(Y). İşlemden sonra kod çözülür ve hatalar kontrol edilir. Aritmetik kod “*” işlemine göre değişmezlik özelliğine dayanır.(A(X*Y) = A(X) *A(Y)) Aritmetik kodların en basit örneği AN kodudur. Seçilen ölçek faktörü (scale factor) sistemin kullandığı sayı sisteminin bir üssü olmamalıdır. Örneğin tüm veri kelimelerinin üçe çarpılmış haliyle oluşan 3N kodunu yapalım. Veri kelimeleri X ve Y ortaya çıkan kod 3*X ve 3*Y olur. Kod kelimeleri toplanırsa toplamın kodlanmış hali olarak 3*(X+Y) çıkar. Daha sonra hatalar, tüm toplamların üçe eşit olarak bölünebildiğini doğrulayarak tespit edilebilir.[3,4]

Hamming kodları, çift hataların tespiti (double error defecting) ve tek bitlik hataların düzeltilmesi (single error correcting) işlemi için tasarlanmıştır. Hamming hata düzeltme işleminde M bitlik bilgiler M+k uzunluğunda kod sözcüğü oluşturmak için k bitlik eşlik kontrol (parity checking) bitleriyle genişletilir. M+k kod sözcüğünün her bitinin konumuna en anlamlı (most significant) bit için 1 ve en az anlamlı (least significant) bit için M + k arasında bir ondalık değer verilir. Daha sonra seçilen bitler üzerinde k eşlik kontrolü gerçekleştirilir. Her bir bit sırasıyla c1,c2, ... ck olarak kaydedilir. Eşlik bitlerinin toplanması, bir ikili sözcük (binary word) oluşturur. Hatalı bir bit görünürse, ikili sözcük ondalık değeri hatalı bitin konumuna verilen ondalık değere eşit olacak şekilde oluşturulur. Hatalı bir bit yoksa eşlik kontrollerinin ondalık değeri sıfır olacaktır.

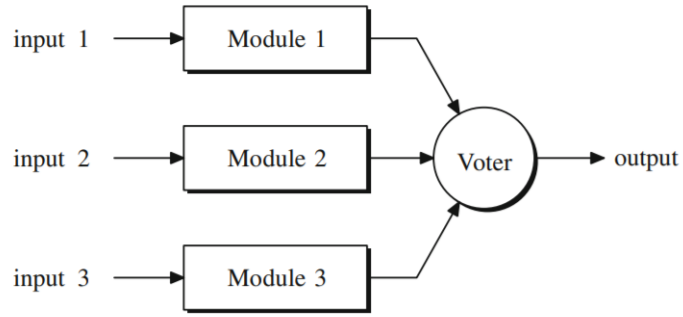
**Donanım Yedekliliği (Hardware Redundancy):**

Hataya dayanıklı sistemler kullanılan bir başka yöntemde fiziksel parçaların kopyalanmasıdır. Bir sistemin fazla işlemcisi, bellekleri, veri yolları veya güç kaynakları olabilir. Bazı zamanlar donanım fazlalığı yöntemi tek çare olabilir. Bir sistemin bakımı yapılamadığında mesela iletişim uyduları yedek parçalar ile çalışma süresi arttırılabilir. Sistemin içine yedek parçalar koymak yararlı olsa da yanında dezavantajlarını da getirir. Örnek olarak ağırlık, boyut, güç tüketimi, maliyet ve tasarım, üretim ve test süresinde artış verilebilir. Burada da bir optimizasyon yapılabilir. Örneğin yüksek seviyeli parçaların yedeği olabilir. Ek olarak sistemin güvenilirliğindeki iyileşme ile bakım masraflarındaki azalma olursa maliyeti daha aza indirebiliriz.

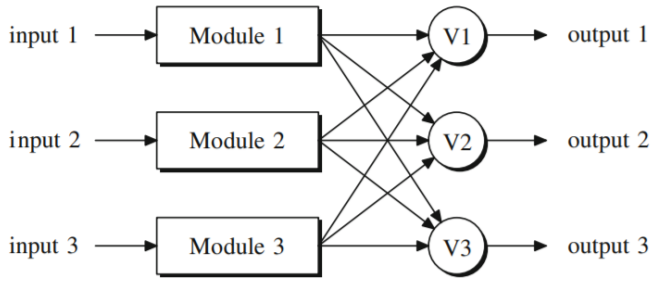
Temel olarak üç tür donanım yedekleme yöntemi vardır: Pasif yedekleme (passive redundancy), aktif yedekleme (active redundancy), hibrit yedekleme (hybrid redundancy).[3,4,9]

İlk olarak pasif yedekleme sistemden veya operatörden herhangi bir işlem yapılmasına gerek kalmadan oluşan hataları maskeleyerek hata toleransı sağlar. Hatanın nerede olduğunun algılanması, yalıtılmasını veya onarımını sağlamaz. Pasif yedekleme genellikle sistemin tamirinin mümkün olmadığı yüksek güvenilirlikli uygulamalarda kullanılır. Bu uygulamalarda sistem çalışmasındaki kısa kesintiler bile kabul edilemez. Örnek olarak uçak uçuş kontrol sistemleri, kalp pili gibi gömülü tıbbi cihazlar ve derin uzay elektroniği verilebilir.

Pasif yedeklemede en yaygın kullanılan yöntem üçlü modüler yedeklemedir (triple modular redundancy - TMR). Aynı hesaplamayı yapmak için bileşenler üç katına çıkarılır. Doğru sonuç belirlenmek için oylama yapılır ve çoğunluk oyu kazanır. Modüllerden birinde hata olsa bile çoğunluk oyu ile maskelenir. İdeal seçmenlerle (voter) yapılan TMR kullanımı Şekil 4'de gösterilmiştir. İdeal seçmenler oluşturmak imkânsız olduğunda her seçmen tüm sonuçları inceler ve doğru sonuç olduğunu düşündüğü sonucu üretir. Bu da şekil 5'de gösterilmiştir. Bir TMR sistemi yalnızca tek bir modüldeki hataları maskeleyebilir. Diğer modüllerde olan hatalar hala seçmenlerin sonuçlarının hatalı olmasına neden olabilir. Gereken yerlerde TMR'ın genellemesi olan NMR (N-modular redundancy) kullanılır.



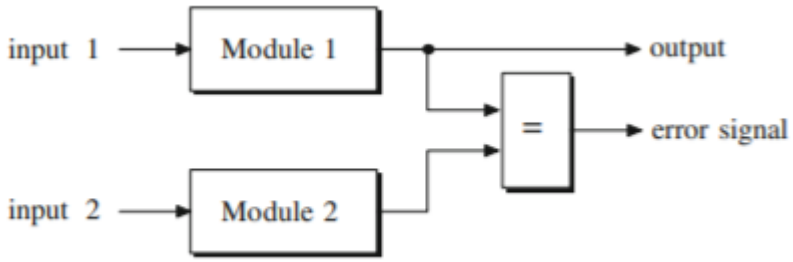

Şekil 4: İdeal seçmen ile TMR




Şekil 5: İdeal olmayan seçmenlerle TMR

İkinci olarak aktif yedekleme bir diğer yöntemdir. Aktif yedeklemede maskeleme yapılmaz bunun yerine hataları tespit eder ve hataları kurtarma işlemi yapar. Genellikle yüksek kullanılabilirlik (high-availability) gerektiren örneğin zaman paylaşımlı bilgisayar sistemleri gibi sistemlerde kullanılırlar. Bu sistemlerde belirli bir süre sonra normal çalışmasına geri döndüğü sürece hatalar kabul edilebilir. Aktif yedekleme yapabilmek üç yaygın yöntem vardır. Bunlar karşılaştırmalı çoğaltma (duplication with comparison), yedekte bekleme (standby) ve çiftt-ve-yedek(pair-and-a-spare).[3]

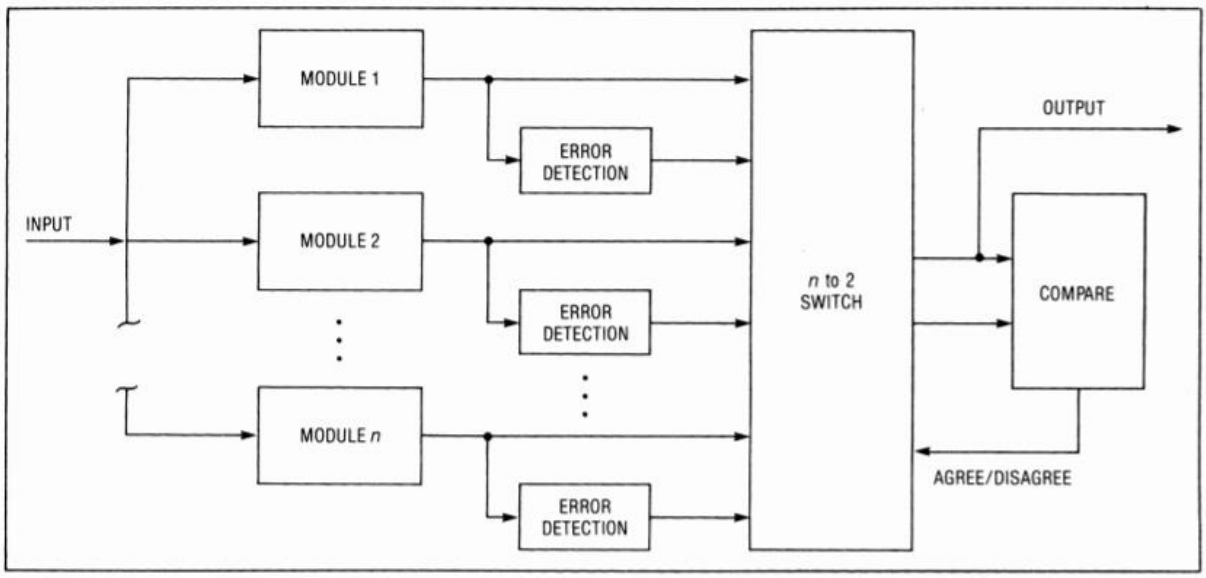
Karşılaştırmalı çoğaltmada şekil 6'de gösterildiği üzere iki özdeş modül paralel olarak çalışır ve sonuçları karşılaştırılır. Eğer sonuçlar uyuşmuyorsa bir hata sinyali üretilir. Sistem yalnızca hata sinyali üretir. Hatayı kurtarmaz.




Şekil 6: Karşılaştırmalı çoğaltma

Bir diğer aktif yedekleme tekniği yedekte bekleme (standby redundancy). Sistemde N tane modül vardır. Yalnız biri aktif olarak çalışır ve diğerleri bekleme kalır. Aktif modülde oluşan herhangi bir arızada beklemede olan parçalardan biri aktif modülün yerini alır. İki tür yedekte bekleme vardır: sıcak bekleme (hot standby) ve soğuk bekleme (cold standby). Sıcak beklemede hem yedek hem de aktif parça senkronize olarak çalışır. Bu şekilde aktif parça arızalandığında hemen yedek parça işe devam eder. Bu yöntem yedek parçaya geçiş yapılırken yapılandırma süresini en aza indirir. Soğuk beklemede yedek parçalar aktif parça arızalanana kadar çalıştırılmaz. Yeniden yapılandırma süresi sıcak belemeye göre daha çoktur. Bu yöntem güç tüketiminin kritik olduğu sistemlerde kullanılır. Örneğin uydu sistemlerinde bu yöntem tercih edilir.

Çift ve yedek (pair and a spare) tekniğinde karşılaştırmalı çoğaltma ve yedekte bekleme teknikleri birleştirilmiştir. Bu yöntemde iki tane aktif modül vardır. Aktif modüllerin çıkardığı sonuçlar karşılaştırılır. Karşılaşma sonucu hata çıkarsa iki modülden hatalı olan yedek modül ile değiştirilir. Şekil 7'de gösterilmiştir.

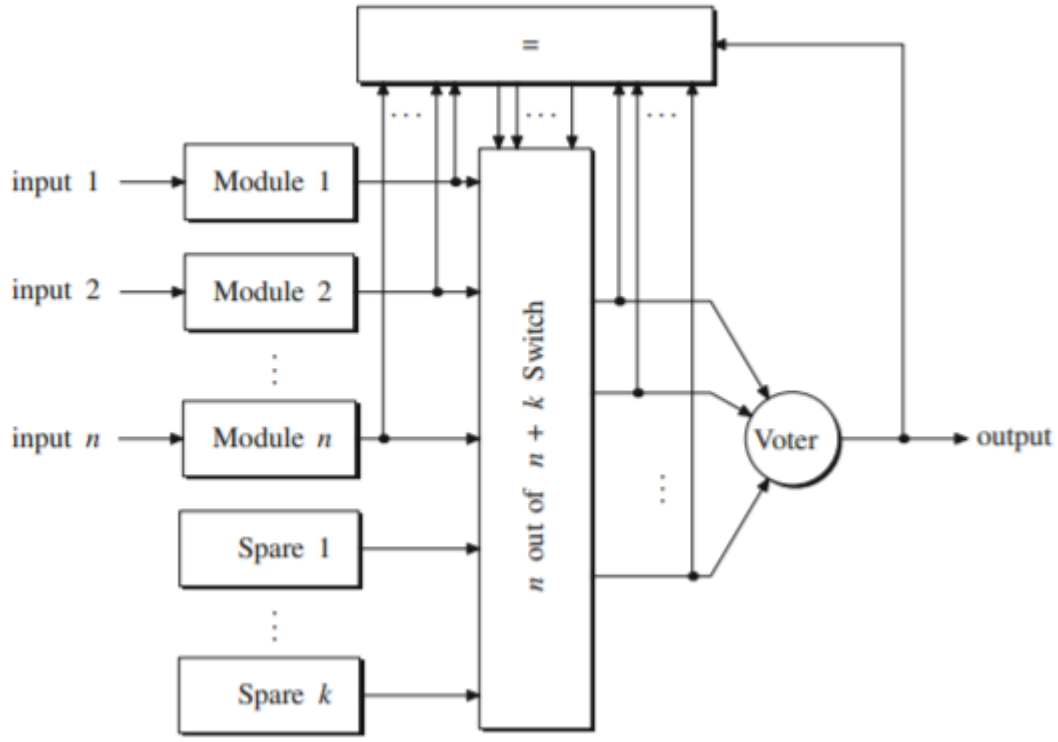



Şekil 7: Pair and a spare tekniği

Son olarak hibrit yedekleme tekniği, aktif ve pasif yedeklemenin iyi yönlerini alıp birleştirmiştir. Hata maskeleme, hata yeri bulma ve tespiti ve yedekte bekleme hepsini birleştiren bir sistem oluşturulur. Bu teknik genellikle güvenlik açısından kritik (safety-critical) uygulamalarda örneğin nükleer enerji santralleri, silahlar, tıbbi ekipmanlar, kimyasal süreçler için kontrol sistemlerinde kullanılır.

Kendi kendini temizleyen yedeklilik (self-purging redundancy), N adet özdeş modül paralel olarak aynı işlemi yapar. Aktif oylama yapılır. Uyuşmazlık tespit edilirken seçmenlerin çıktıları ile her bir modülün çıktısı karşılaştırılır. Bir hata tespit edilirse hatalı modül sistemden çıkarılır veya temizlenir.

K yedeklilikle N modüler yedeklemede (N-Modular Redundancy with Spares) N tane modül seçmenlere girdi verir ve k tane modül de yedekte bekler. Birincil modüllerden biri arızalanırsa yedek modül ile değiştirilirler. Hatalı modülleri tanımlamak için birkaç yöntemden biri seçmenlerin çıktılarıyla modüllerin çıktılarının karşılaştırılmasıdır. Genellikle çoğunlukla aynı fikirde olmayan modül hatalı ilan edilir. Yedek havuzu bitince sistem pasif NMR sistemi olarak çalışmaya devam eder. Oylama her zaman aktif birimler arasında olur ve k kadar yedeğe sahip n modüler yedekleme sistemi n / 2 + k kadar modül hatalarını maskeleyebilir.



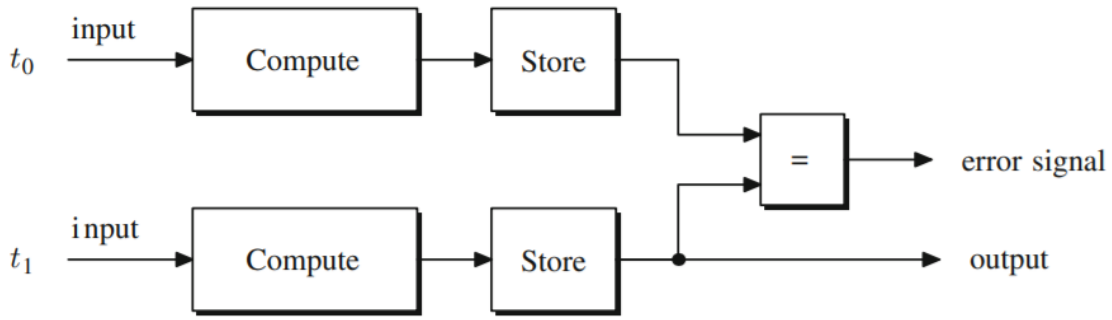

Şekil 8: K yedeklilikle N modüler yedekleme

**Yürütme (Zaman) Yedekliliği (Time Redundancy):**

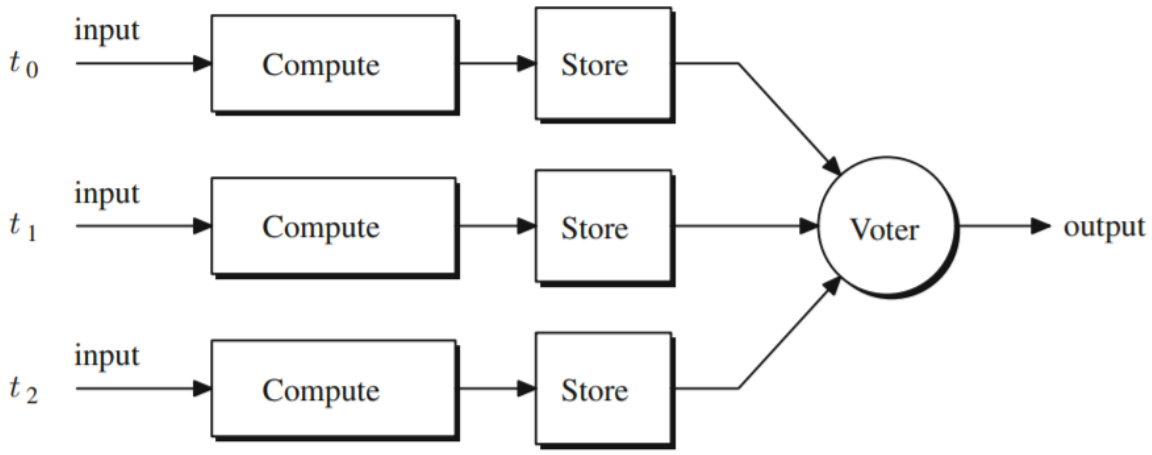
Bazı uygulamalarda hata tespiti ve toleransı sağlamak için ek yürütme (execution) kullanılabilmektedir. Bu tekniğe zaman (yürütme) fazlalığı (time redundancy) denir. Genellikle donanım ve yazılım yedekleme yöntemleriyle beraber kullanılmaktadır. Tekrarlanan yürütme yoluyla yeniden başlatma veya işlemi yeniden çalıştırma ile hatanın geçici ya da kalıcı olduğu anlaşılarak hata tespiti yapılıp düzeltme işlemleri yapılabilir. [4]

Yürütme fazlalığı hesaplamayı, veri iletimini iki veya daha fazla kez tekrarlayarak ardından sonuçları önceden depolanan doğru sonuçların kopyalarıyla karşılaştırmayı içerir. Karşılaştırma sonuçlarında bir uyuşmazlık olursa hata var demektir. Şekil 9'a iki kez tekrarlama yapılırsa hata tespiti yapılabilir. Şekil 10'a üç kez tekrarlama yapılırsa hata düzeltilebilir [3]. Donanım yedeklemesine benzer olarak oylama teknikleri ile doğru sonucu seçme işlemi yapılır. Yürütme fazlalığı geçici ve kalıcı hataların ayırt edilmesinde yararlıdır. Yeniden hesaplamadan sonra hata kaybolursa bu hataya geçici diyebiliriz. Yürütme fazlalığında temel sorun bir hesaplamayı tekrarlamak için gereken verilerin sistemde mevcut olması gerekliliğidir. Hatanın meydana geldiği durumda tekrar kontrol yapıldığından kontrollerin, hatanın meydana geldiği veri ile yapılması gerekmektedir. Bu tip durumlarda da işlemci kontrol amaçlı olan doğru veriyi

sadece okuma işlemi yapılan bloklarda bulundurup oradan okuyarak gerekli işlemi yerine getirmektedir.[4]




Şekil 9: Yürütme fazlalığında hata tespiti




Şekil 10:Yürütme fazlalığında hata düzeltme

**Yazılım Yedekliliği (Software Redundancy):**

Yazılım yedekliliği (software redundancy), hata tolerans özelliği sağlamak için sisteme ek yazılımlar eklenmesidir. Yazılım hata toleransı teknikleri iki gruba ayrılabilir: tek sürüm (single version) ve çoklu sürüm (multi version) .Tek sürümlü teknikler, bir yazılım bileşenine hata tespiti, sınırlama ve kurtarma mekanizmaları ekleyerek onun hata toleransını iyileştirmeyi amaçlar. Çok sürümlü teknikler, tasarım çeşitliliği kurallarına göre geliştirilen fazladan yazılım bileşenlerini kullanır. [3,4]

Yazılım yedeklemede tek sürümlü tekniklerde hata tespiti için çeşitli kontrol testleri kullanılır. Bir program teste tabi tutulur ve sonuçta testi geçerse program çalışmaya devam eder. Hızlı bir şekilde kontrol edilebilen testler daha etkilidir. Bir test programla neredeyse aynı süreyi alıyorsa, programın farklı versiyonunu oluşturup iki versiyonun sonuçlarının karşılaştırması yapılabilir. Testlerin bir başka istenilen özelliği uygulama bağımsızlığıdır. Bu test başka uygulamalarda da tekrar tekrar kullanılabilmesi demektir. Bazı testler şunlardır:

zamanlama kontrollerini (timing checks), kodlama kontrollerini (coding checks), ters kontroller (reversal checks), makullük kontrollerini (reasonableness checks) ve yapısal kontrolleri (structural checks) içerir [3].

Zamanlama kontrolleri, olması gereken davranışlardan sapmaları gözlemlemek için kullanılabilirler. Bir "watchdog timer" bekçi zamanlayıcısı zamanlama kontrollerine bir örnektir. Kodlama kontrolleri, verileri kodlanılabilen programlar için kullanılırlar. Ters kontroller, programın çıkış değerini ters çevirerek ve ona karşılık gelen giriş değerini hesaplanın mümkün olduğu zamanlarda kullanılır. Burada gerçek girdiler ile hesaplanan girdiler karşılaştırılır. Makullük kontrolleri, hataları tespit etmek için verilerin anlamsal özelliklerini (semantic properties) kullanır. Örneğin overflow ve uderflow incelenebilir. Yapısal kontroller, veri yapılarının bilinen özelliklerine göre yapılır. Örneğin bir listedeki öğeler sayılabilir. Yapısal kontroller yedek data eklenilerek daha verimli hale getirilebilir.

Hata çevreleme tekniklerine gelecek olursak dört yaygın teknik şunlardır: modülerleştirme (modularization), bölümleme (partitioning), sistem kapatma (system closure) ve atomik eylemler (atomic actions) [3].

Modülerleştirme, bir sistemi modüllere ayırarak, paylaşılan kaynakları ortadan kaldırarak ve modüller arasındaki iletişim miktarını dikkatle izlenen mesajlarla sınırlandırarak hataların yayılmasını önlemeye çalışır. Bölümleme, işlevsel olarak bağımsız modüller arasındaki çevreleme işlemini yapar. Sistemi yatay veya dikey boyutlarda modüler boyutlama hiyerarşisi oluşturulur. Sistem kapatmada sistemin herhangi bir bileşenine sadece işlevini yerine getirmek için gerekli yetenek verilir. Bu nedenle herhangi bir hatayı kontrol altına almak daha kolaydır. Atomik eylemler, sistem bileşenleri arasındaki etkileşimleri tanımlamak için atomik eylemleri kullanır. Bir gruptaki bileşenlerin yalnızca birbiriyle etkileşime girdiği faaliyet süresi boyunca sistemin geri kalanıyla bileşenler arasında herhangi bir etkileşim olmaz.

Sistemde hata tespit edildiğinde ve çevrelendiğinde sıra sistemi kurtarma işlemine gelir. Etkili bir kurtarma mekanizması için hatanın çevrelendiği yerin belli olması gerekir. İstisna işleme (exception handling), bazı anormal durumlarla başa çıkabilmek için sistemin normal çalışması kesilir. Anormal koşullar düzgün ele alınmazsa sistemin çökmesine neden olabilir. Anormal koşullardan kaynaklanan hataların sistem çökmelerinin üçte ikisine ve güvenlik sorunlarının %50 sine neden olduğu tahmin edilmektedir [6]. Denetim noktası ve yeniden başlatma (checkpoint and restart) bir diğer kurtarma mekanizmasıdır. Yazılım hatalarının çoğu bazı beklenmedik girişlerden kaynaklanan tasarım hatalarıdır. Bu tür hatalar, aralıklı donanım arızalarına benzer. Bu nedenle modülün yeniden başlatılması tekrardan normale dönüp başarılı çalışmaya devam etmesi için yeterlidir.

Çok sürümlü tekniklerde tasarım çeşitliliğini esas alarak aynı yazılım bileşeninin birden çok sürümü oluşturulur. Örneğin, farklı sürümlerin ortak hataları olmaması olasılığını en üst düzeye çıkarmak için farklı ekipler, farklı kodlama dilleri veya farklı algoritmalar kullanılabilir. Üç yaygın tekniği vardır. Bunlardan kurtarma blokları (recovery blocks), bir yazılım modülünün birden çok sürümüne denetim noktası ve yeniden başlatma tekniği uygulanır. Bir diğeri N-sürüm programlama (N-version programming), aynı yazılımın eş zamanlı olarak çalıştırılan N farklı yazılım uygulamasından oluşur. Tüm sürümlerin işlevi aynıdır. Donanım yedeklemedeki N modüler yedeklemeye (N-moduler radundancy) benzerdir. Bir diğer teknik N kendi kendini kontrol eden programlama (N self-checking programming), kurtarma blokları tekniği ile N sürüm programlama tekniğini birleştirir. Testler ya da modül çiftlerinin karşılaştırılması ile kontroller yapılır.

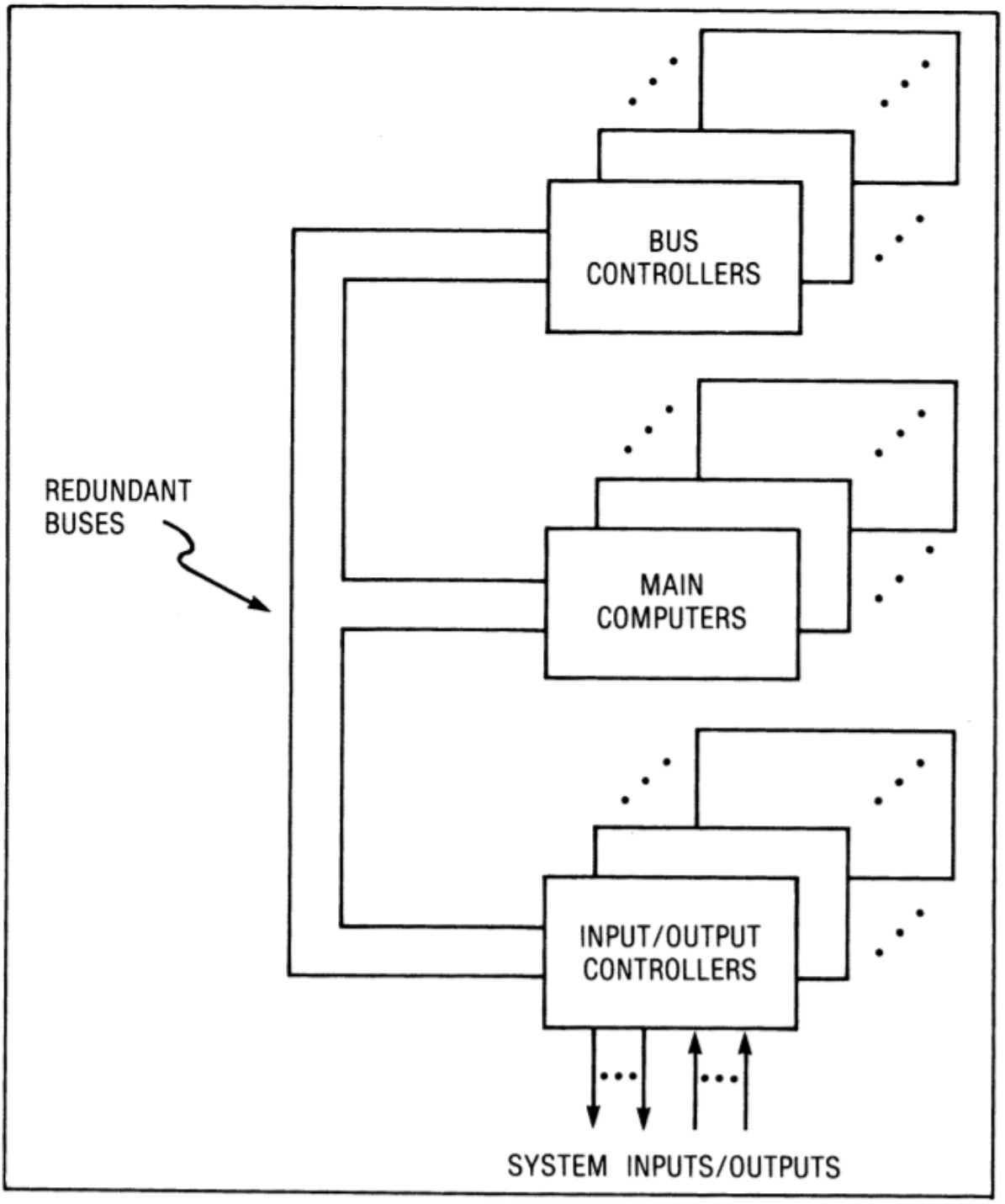
## Hata Toleranslı Sistemler

### SIFT

SIFT (Software-Implemented Fault Tolerance) Ulusal Havacılık ve Uzay Dairesi (National Aeronautics and Space Administration) için Stanford Araştırma Enstitüsü'nde (Stanford Research Institute) geliştirilmiştir. Yüksek düzeyde güvenilirliğe sahip ve aynı zamanda programlanması, çalıştırılması ve bakımı kolay olan bir bilgisayar geliştirilmeye çalışmışlardır. SIFT, kritik kontrol yasaların hesaplanması için geliştirilmiştir bu yüzden SIFT için hata maskeleme önemlidir.

SIFT çok işlemcili mimari (multiprocessing architecture) kullanılmıştır. Bu şekilde işlemcilerin periyodik olarak kapatılmasına ve temel sistemin işlevsel kapasitesi azalmadan test edilmesine izin verilir. Hataların oluşumunu maskelemek için tüm kritik işlemler üçlü yazılım oylama ile yapılır. Tüm işlem birimlerinin yedekleri havuzda hazır halde herhangi bir hataya karşı hazırda bekletilirler. Yedek kaynakların tüketilmesi durumunda en kritik işlevlerin daha az kritik işlevler pahasına gerçekleştirildiği sabit öncelikli bir sisteme göre zarif bozulma sağlanır. Sistem, yinelemeli bir görev dizisi gerektiren uygulamalar için tasarlanmıştır. Yani, bir göreve yapılan girdinin önceki bir görevin çıktısı olduğu durumlar denilebilir. Bu şekilde sadece çeşitli görevlerin girdi ve çıktı sonuçları oylanır. Bu şekilde oylama miktarından tasarruf edilir. Oylamalar yazılımlarla gerçekleştirilir ve basit çoğunluk oyu yöntemi kullanılır. Bir işlemci çıktısının sonucunu kendi ayrılmış belleğine kaydeder ve bu belleklere diğer işlemciler de erişebilir. Tüm işlemciler tüm bellekleri okuyabilse de sadece bir belleğe yazabilme izni vardır. Bu şekilde hatalı işlemci diğer işlemcilerin belleğini de bozamaz.

SIFT sisteminde tüm işlemcilerin bağımsız bir saati vardır. Buna gevşek senkronizasyon denir. İşlemcilerin komut döngüsü senkronizasyonunda çalışılması gerekmez. Bu özellikle birlikte bozulursa tüm sistemin bozulmasını sağlayacak bir orta saat yoktur.



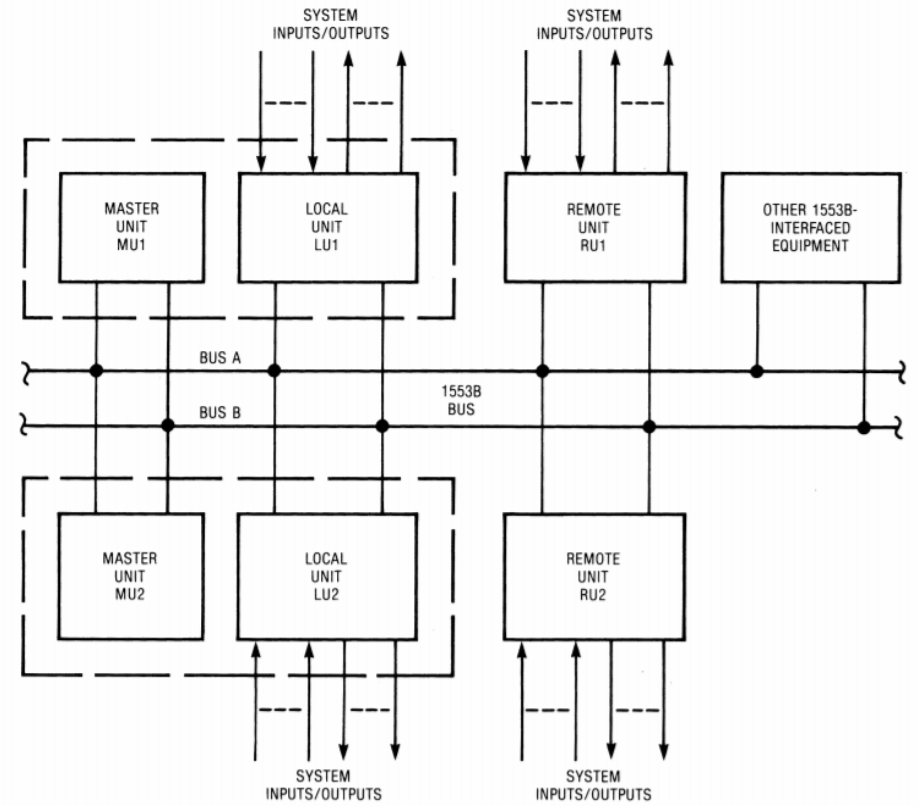

Şekil 11:SIFT Sistemi mikroişlemci mimarisi

**Agusta 129**

Agusta A129 Integrated Multiplex System Harris Corporation tarafından tasarlanmıştır. Çoklu işlemcili bir sistem tasarlanmıştır. Otomatik helikopter uçuş kontrolü, stabilite artırma, navigasyon, motor izleme, performans izleme, yakıt izleme, rotor ve şanzıman izleme gibi görevler için tasarlanmıştır. Mimarisi Şekil 11'de gösterilmiştir. Şekilde görülen Yerel birimler (local units) fiziksel olarak ana birimlerle (main units) aynı kutularda bulunurken, ana birimler yerel birimlere yalnızca 1553B veri yolu aracılığıyla erişebilir. Yerel birimler ve uzak birimler (remote units), sensörler ve aktüatörlerle arayüz sağlarken, ana birimler işleme kapasitesi sağlamak için tasarlanmıştır. Ana birimlerden sadece biri herhangi bir zamanda 1553B veri yolunun kontrolüne sahiptir.

Agusta sisteminin ikili yedekli yapısı nedeniyle her bir birimin hataları tanımlamak için gerçek zamanlı testler yapılması konsepti seçilmiştir.

Agusta sisteminde bazı yazılım hatası algılama araçları vardır. Örneğin belirleyici görevler için zamanlayıcılar kullanılmaktadır. Dijital kontrol algoritmaları gibi bu görevler, çalışmak için belirli bir süresi vardır ve bu süreyi aşmamalıdırlar. Bu süre aşıldığında donanım zamanlayıcıları işlemciyi interrupt eder ve yazılımdaki sonsuz döngüler işlemcinin kapanmasına sebep olan belirli donanım hataları olarak tespit edilir.

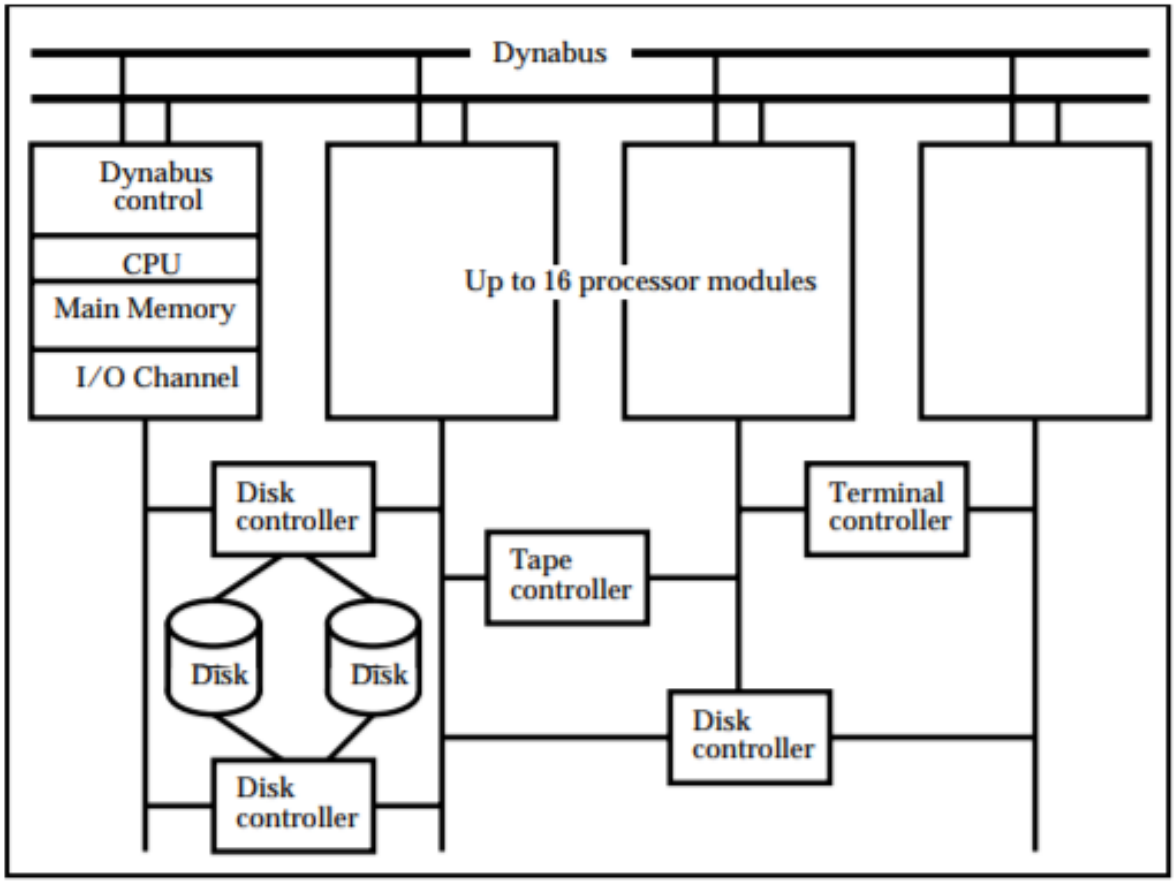



Şekil 12: Agusta129 mikroişlemci mimarisi

**Tandem 16 Nonstop**

Tandem 16 Nonsttop ilk yüksek kullanılabilirlik için tasarlanmış ilk ticari sistemdir. Bu sistemin üç ana hedefi vardı. Bunlardan ilki kesintiye uğramadan hataların düzeltilmesi, ikincisi tüm tek hataların tolere edilmesi, üçüncüsü ise sistemin daha kolay genişletilebilmesi için modülerliğin kullanılmasıdır.

Tandem sistemlerinde çift veri yolu ile bağlanmış 2 den 16 ya kadar ulaşabilen işlemci desteklenmektedir. Her bir işlemci 16 bitlik merkezi işlem birimi, bir I/O kanalı ve 512KB ulaşabilen ana bellekten oluşur. Sitemdeki her işlemci kendi özel güç kaynağından güç alır. Veri yolu ve belleklerdeki hataların algılanması için checksum, eşlik hatası denetimi, hata düzeltme kod bellekleri ve watchdog zamanlayıcıları kullanılır. Örneğin watchdog zamanlayıcıları için örnek verilecek olursa her işlemci sistemdeki diğer tüm işlemcilere saniyede bir sağlığını belirtmek için özel bir mesaj iletir ve iki saniyede bir her işlemci diğer işlemcilerden bu mesajı alıp almadıklarını kontrol eder. Bir işlemciden mesaj alınmadığı zaman bir hata olduğu ortaya çıkar.

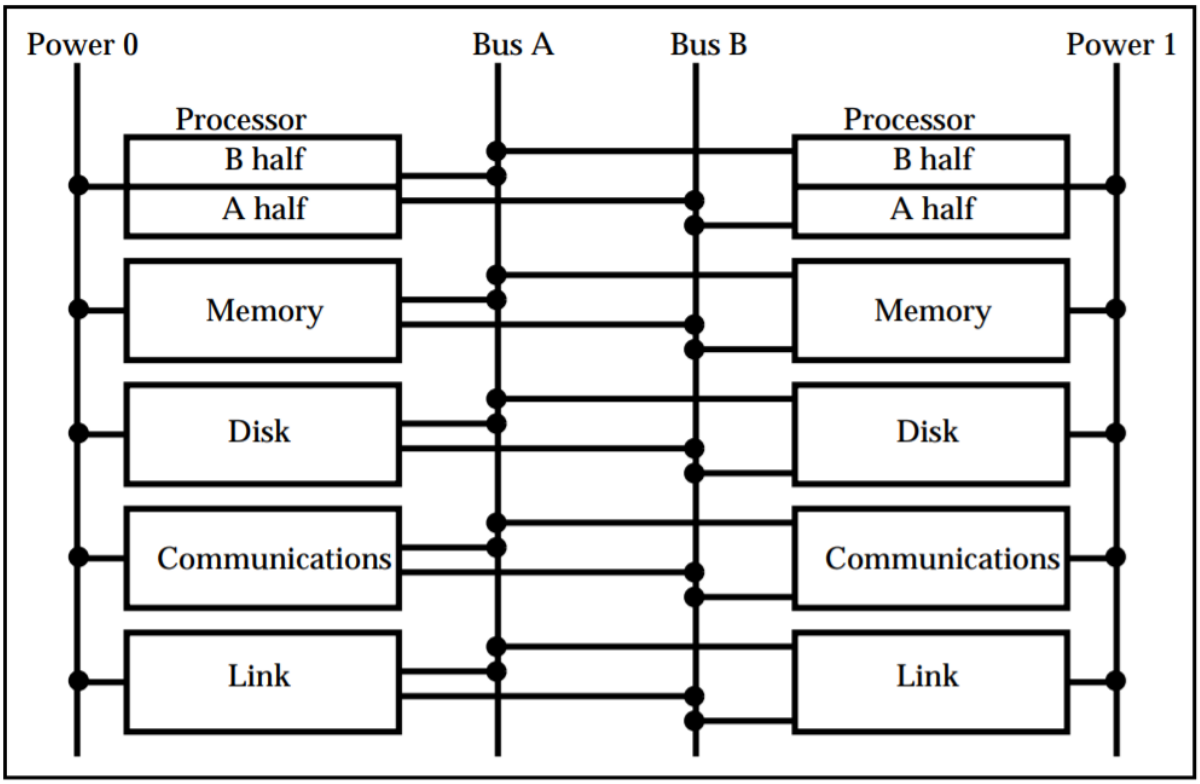



Şekil 13: Tandem 16 NonStop Mimarisi

**Stratus**

Stratus da tandem gibi yüksek kullanılabilirlik için tasarlanmış bir ticari sistemdir. Stratus sistemi pair and a spare yöntemi ile çalışır. Donanım yoğunlukludur. Örneğin bir işlemcinin işini dört işlemcinin yapabilmesi gibi bir sorunu vardır.

Stratus sisteminde her işlemci kartı iki işlemciden oluşur ve bu işlemciler aynı saat tarafından kontrol edilirler. İşlemci çıktıları karşılaştırılır ve çıktılar eşleşmezse o kart kendini hizmet dışı bırakır. İşletim sistemi bu kart üzerinde hataları tanımlamak için testler yapar. Geçici bir hataysa kart geri hizmete döner ama kalıç bir hataysa kartın değiştirilmesi gerekir. bu kart değiştirme işlemi sistem çalışır haldeyken yapılabilir. Çünkü iki işlemci kartı aynı anda aynı işlemi yapar. Bellek de aynı şekilde çalışır. Startus sisteminde tüm donanım hatalarının tolere edilmesi bekleniyordu ama yine de hatalar olabilir bu yüzden sistem çökmelerini en aza indirebilmek için yazılım kurtarma tekniklerini de kullanır.

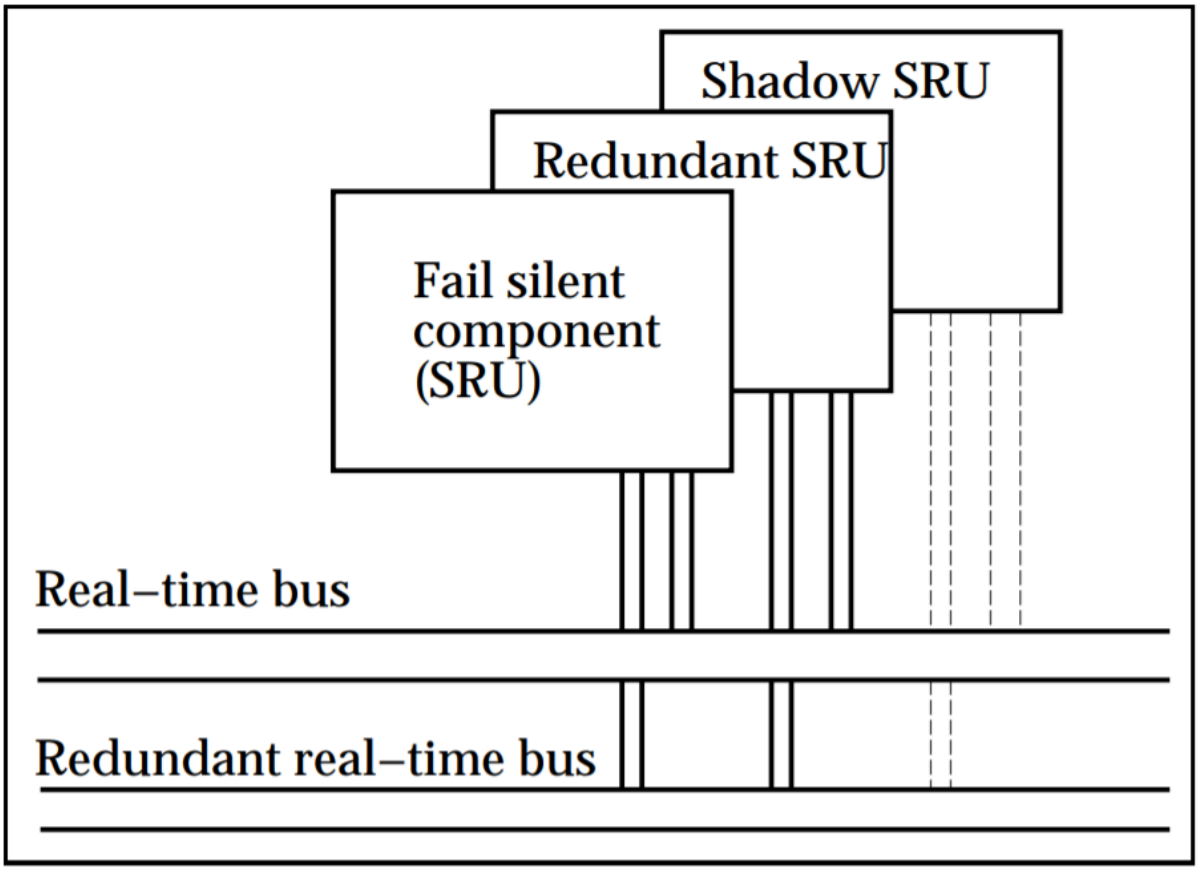



Şekil 14: Stratus Mimarisi

**MARS**

Mars sistemi Viyana Teknik Üniversitesinde geliştirilmiştir. dağıtılmış hataya dayanıklı bir sistem gerçekleştirilmeye çalışılmıştır. Zamanlama hatalarının önemli olduğu gerçek zamanlı uygulamalar için tasarlanmıştır.

Dağıtılmış sistemler genel olarak hata toleranslı bir sistem için merkezi sistemlere göre daha uygundur. Tüm parçaların değil de bazı parçaların iyi olduğu sürece sistem çalışabilir. Bu durumda uygun yazılımlar kullanılarak fazladan maliyet oluşturmadan hata toleranslı bir sistem oluşturulur. Ek olarak dağıtılmış sistemlerde en küçük değiştirilebilir birimler (Smallest Replaceable Units SRUs) oluşturulabilir. İki veya daha fazla hata-sessiz (fail-silent) yani ya doğru bir şekilde çalışacak ya da hiç hizmet vermeyen SRU'lar birlikte gruplandırılıp hataya dayanıklı birimler (FTU) oluşturulabilir [11]. Bu MARS sisteminde üç SRU bir FTU oluşturacak şekilde tasarlanmıştır. Bir SRU gölge (shadow) olarak çalışır diğer iki SRU'dan birinde hata olması durumunda onun yerine geçmek için bekler. Şekil 15'de gösterildiği gibi SRU'lar çoğaltılmış veri yolu üzerinden iletişim kurarlar. MARS sisteminde bir SRU uygulama ve iletişim olarak iki bölümden oluşur. Hata tespiti sağlamak için RAM ve FIFO'larda eşlik bitleri kullanılır. İletişim bölümünde ise watchdog zamanlayıcı kullanılır.




Şekil 15. MARS mimarisi

## KAYNAKÇA

1-https://en.wikipedia.org/wiki/Fault_tolerance

2-J.-C. Laprie. Dependable computing and fault tolerance: Concepts and terminology. In The 15th International Symposium On Fault-Tolerant Computing, pages 2–11, 1985.

3-Dubrova, E., Fault Tolerant Design, Springer, New York, 2013.

4-B. W. Johnson, "Fault-Tolerant Microprocessor-Based Systems," in IEEE Micro, vol. 4, no.6, pp. 6-21, Dec. 1984, doi: 10.1109/MM.1984.291277.

5-A. Aviziens,"Fault-Tolerant Systems", in IEEE TRANSACTIONS ON COMPUTERS, VOL. C-25, NO. 12, DECEMBER 1976

6-Maxion, R.A., Olszewski, R.T.: Improving software robustness with dependability cases. In: Proceedings of the The Twenty-Eighth Annual International Symposium on Fault-Tolerant Computing, pp. 346–355 (1998)

7-A. Correcher, E. Garcia, F. Morant, E. Quiles and L. Rodriguez,"Diagnosis of Intermittent Faults and its dynamics",March 2010

8-I.Koren,C.M. Krishna, (2007), "FAULT TOLERANT SYSTEMS"

9-https://en.wikipedia.org/wiki/Fail-silent_system

10- A. D. JALNAPURKAR, (1988), “A FAULT-TOLERANT MULTI-MICROCOMPUTER SYSTEM”

11-K. Nørvag, “An Introduction to Fault-Tolerant Systems” , IDI Technical Report 6/99, Revised July 2000 ISSN 0802-6394