Hata Toleranslı Mikroişlemci Tabanlı Sistemler

ALİ HADİ ALTUNGÖK

GAZİ ÜNİVERSİTESİ MÜHENDİSLİK FAKÜLTESİ
BİLGİSAAR MÜHENDİSLİĞİ
BİLGİSAYAR MİMARİSİ
Prof.Dr. M.ALİ AKÇAYOL

ARALIK 2020

ÖZET

Hata toleranslı mikro işlemci tabanlı sistemlerde, sistemde meydana gelebilecek herhangi bir hatanın sistemi sekteye uğratamaması gerekmekle birlikte sistemin bu hatayı tolere edebilecek şekilde tasarlanıp bu özelliklerle donatılması sonucu üretilmiş olması gerekmektedir. Bu sistemlerde birçok çeşitli hatalar meydana gelebilmektedir. Dolayısıyla bu hata türlerine özgü olarak hatayı tolere edebilecek tekniklerle geliştirilip hatalar maskelenerek sistemler geliştirilmiştir. Bu araştırmamızda Hata toleranslı sistemlerden bahsedip, mikro işlemcili sistemlerde meydana gelebilecek hataları, çeşitlerini ve özelliklerini ayrıntılı bir şekilde açıklayacağız. Ardından hata toleransının temel unsurlarından bahsedip hata toleransı tekniklerini ayrıntılı bir şekilde inceleyerek bu tekniklerdeki kodlamaları, kaynaklardan elde ettiğimiz şekillerle birlikte ayrıntıları ile anlatacağız. Hata toleranslı sistem tasarımını anlatarak kaynaklardan elde ettiğimiz temel bazı hata toleranslı mikro işlemci mimarilerinde tasarımın nasıl gerçekleştiğini ve bu mimarilerin özelliklerini anlatıp araştırmamızı tamamlayacağız. Bu araştırma ödevini çoğunlukla *Barry W. Johnson' un Fault-tolerant Microprocessor-based Systems* adlı makalesini esas alarak yapıp sınıflandırmaları çoğunlukla oradakine göre yaparak faydalandığım diğer makaleleri de kaynakça da belirterek doğrudan alıntı yaptığım kaynakları referanslandırarak aldığım her bir şemanın kaynakçalarını belirterek kapsamlı ve ayrıntılı araştıramamı gerçekleştirdim.

Hata Toleranslı Mikroişlemci Tabanlı Sistemler
(Fault-Tolerant Microprocessor-Based Systems)

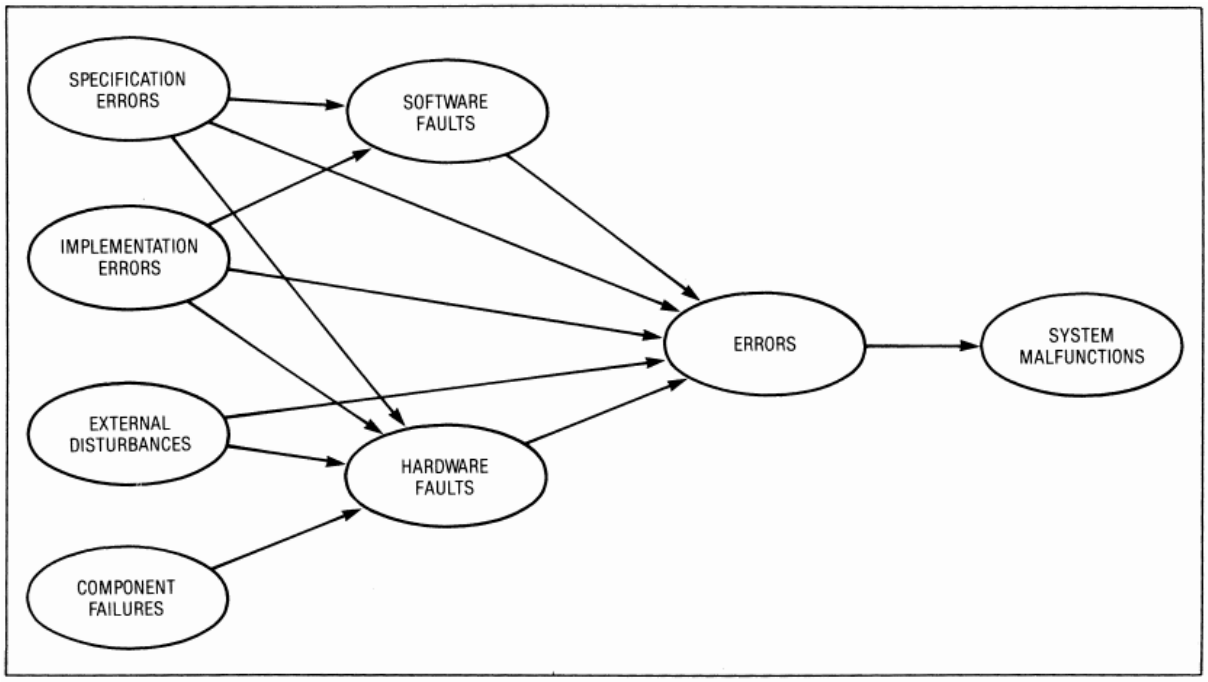
Hata toleransı, bir sistemin bazı birimlerinde yada bileşenlerinde bir yada daha fazla sorun ortaya çıkması halinde sistemin tamamen çökmesi yerine çalışabilililiğini sürdüre bilmesiyle birlikte o sistemin hatalara karşı sahip olduğu toleranstır. Bu sistemlerde hata meydana gelmesi sonucu sistemin işleyiş kalitesinde düşüş meydana gelebilmekte ve bu düşüş hata ciddiyeti ile orantılı olmakta ancak sistem kesintisiz işlemeye devam etmektedir.

Hata toleransı, özellikle High-Availability system (Yüksek kullanılabilirlikli) yani hastaneler, veri merkezleri ve sunucular gibi rutin günlük aktivitelerin zorunlu olarak gerçekleştirilmesi gereken sistemlerde ve Safety-Critical (Güvenlik açısından kritik) yani insanların ölümüne, ciddi şekilde yaralanmalara, çevresel zararlara ve mülk kaybına sebep olabilecek kalp pili, nükleer santraller yada uzay mekikleri gibi sistemlerde olması gerekmektedir.

Hataya toleranslı sistemlerde arızalı bileşenin yerini alarak hizmek kaybını önleyen yedekleme bileşenleri vardır. Donanımsal bir hata toleransı için örnek verecek olursak sunucuya ek olarak paralel çalışan aynı bir sununcunun kullanılması ile tüm işlemler yedeklenen sunucuyada aktarılarak sistem hataya toleranslı bi hale getirilmiş olunur. Yazılımsal bir hata toleransına kabaca örnek verilecek olunursa müşteri bilgileri tutulan bir veri tabanı eş zamanalı olarak başka bir makineye kopyalanacak olunursa, herhangi bir problem ve sorun sonucunda işlemler yedeklenen makine üzerinden gerçekleştirilerek sistem hataya toleranslı bir hale getirilmiş olunur. Kısaca güç kaynaklarında meydana gelebilecek sorunların çözümünün jeneratörler yardımı ile yapımasını da hata toleransına örnek olarak verebiliriz.

Hataları ayrıntılı bir şekilde inceleyecek olursak kaynağımızda 4 tip temel hatadan bahsedilmiştir. Spesifikasyon(Spesification Errors), Uygulama (Implementation Errors) , Bileşen(Component Failures) ve Harici (External Disturbances) hatalar.[1] Spesifikasyon hataları geliştirme ve üretim aşamasındaki bazı durumların sistem hatasına sebep olabildiği durumlardır. Bu tip hatalar algoritmik hatalar, mimari hatalar ve genel itibariyle donanımsal ve yazılımsal tasarım hatalarıdır. Uygulama hataları genellikle kötü tasarım, zayıf bileşen kullanımı, zayıf yapı kullanımı veya yazılım kodlama sonucu oluşan hatalardır. Bileşen arızaları imalat hataları ve kusurları, yarı iletken cihaz arızaları ve bileşen yanması gibi hatalardır. Bileşen arızaları dijital sistemlerde en sık dikkate alınan hata türleridir. Harici hatalar

ise radyasyon, elektromanyetik girişim, savaş hasarı ve çevresel aşırılıklar gibi sorunlardır. Bütün bu hata türleri ile donanımsal ve yazılımsal hata arasındaki ilişkiyi aşşağıda verilen şema üzerinden de inceleyebiliriz.



Şekil 1[2] **Sistemde hataların oluşumu ve aralaındaki ilişki**

Sistemin normal işleyişini devam ettirmek için hatalara karşı 3 ana teknikten bahsedilmektedir. Birincisi hatalardan kaçınma, ikincisi hataları maskeleme, üçüncüsü hata toleransı. Hatalardan kaçınmak, tasarımı gözden geçirerek, bileşen taraması yapılarak ve test edilerek bahsetmiş olduğumuz temel hata nedenlerini önlemektir. Hata maskeleme sistem hatasının sistemdeki veriler üzerinde yayılmasının önlenmesidir. Araştırma ödevimizin konusu olan hata toleransı ise sistemde hata meydana geldikten sonra belirtilen görevlerin gerçekleştirilmesine devam edilebildiği hatalardır.

Hataların önemli bir başka özellikleride süreleridir. Üzerinde düzeltme işlemi yapılamayacak süresiz olarak var olan hatalar, çok kısa süre içerisinde periyodik olarak tekrarlanabilen geçici hatalar olmakla birlikte gizli hatalarda mevcuttur. Gizli hatalar bir datadan kaynaklanıyor olabilir. Bu hataları belirlemek çok zor olabilir. Başka hatalardan da kaynaklanıyor olabilmeleri sonucu çoklu hata durumu ortaya çıkabilmektedir.

Hata toleransı dört ana temel unsuru içermektedir: hata tespiti(fault detection), hata konumu(fault location), hata önleme(fault containment), hata kurtarma(fault recovery). Hata tespiti, sistemde hata olduğunun tespit edilmesi aşamasıdır. Hata konumu aşaması sistemin

hatanın konumnunu belirleyebilme aşamasıdır. Hata önleme hatayı ve sistem boyunca yayılmasını izole etme aşamasıdır. Hata kurtarma ise sistemin yeniden yapılandırma ile işleyişini kazanabilmesidir.

Hata Toleransı Teknikleri

Hata toleransı tekniklerine genel olarak Redundancy(Yedekleme/fazlalık) denmektedir. Redundancy Information Redundancy(Bilgi Fazlalığı), Donanımsal Redundancy, Yazılımsal Redundancy, Zamansal Redundancy olmak üzere çeşitli formlara sahiptir. Bu araştırma ödevinde mikro işlemcili sistemlerdeki bu teknikleri inceleyeceğiz.

Information Redundancy (**Bilgi Yedekliği/Fazlalığı**) bir fonksiyonun, işlevin uygulamasında gerekli olandan bilgiye ek olarak daha fazla bilgi (data) eklenmesi ile gerçekleştirile bilmektedir. Temel veri yapısına bilgi(data) eklenmesi durumunda oluşan hataların hata tespit kodları ile tespit edilmesi Information Redundancydir. Bir üniteden başka bir üniteye bilgi aktarırken datada hatalar meydana gelmesini önlemek için kulalnılarak bilgi üzerindeki hatalar tolere edilmiş olunur. Geçerli(bozulmamış) ve geçersiz(bozulmuş) kodlar ayırt edilerek bu durum gerçekleşir. Bilgisayarın temelinde datalar 1 ler ve 0 lardan oluştuğundan tek bir bit de meydana gelecek değişim bütün sistemi etkileyebilir. Bu istenmeyen değişimlerin olmaması gerekir aksi taktirede hata meydana gelmiş olacaktır. Information Redundancy ile bu durum kontrol altına alınıp bu hatalar tolere edilmeye çalışılmıştır. Bir kodda sadece bir bitinde bile istenmeyen bir değişimin meydana gelmesi, kodu geçerli olmaktan çıkıp geçersiz kod haline getirmelidir. Herhangi bir kodun tek bir bitinde bile meydana gelen değişim ve tek bir hata durumunda kodun geçerli durumdan geçersiz kod haline dönüştürülmesine "single-error-detecting code (tek hata tespit kodu)" denmektedir.

Hata tespit kodlamasına verilecek en basit kodlama "single-bit parity check(tek bitlik eşlik denetimi)" dır.[3] Bo kodlama, kod satırına bir bit eklenerek tek bir bit ile kod satırında ki veride değişim olup olmadığının kontrol edilmesidir. En basit haliyle kod satırına bir bit eklenir ve bu bit satırın tek sayımı yoksa çift sayı mı olduğu bilgisini tutar. Eğer herhangi bir bitde sadece bir değişim meydana gelecek olursa sayının değeri çift iken tek, tek iken çift olacağından bu da kodun geçersiz kod olmasını sağlayacaktır.

Bir başka hata tespit kodu ise "M-out-of-N coding"dır. Bu form kodlamada satırın N bitlik olacak şekilde seçilir. Ve N satırlık kısımda M tane 1 lik değer olacak anlamına

gelmektdir. Eğer satırda herhangi bir bitte bile bir değişim meydana gelecek olursa bu durumda N tane bit arasında 1 olanların sayısı M den az yada fazla olacaktır ve burumda kodun geçerli(hatasız) yada geçersiz (hatalı) olduğu belirlenmiş olacaktır. Aşşağdaki tabloyu da inceleyerek decimai sistemdeki sayıların 2-out-of-5 deki karşılığını görebilmekteyiz.

Decimal Digit	2-out-of-5
0	00011
1	00101
2	00110
3	01001
4	01010
5	01100
6	10001
7	10010
8	10100
9	11000

Şekil 2 (M-out-of-N coding)

Hata tespit kodlarından biride Checksum(Toplam kontrolü) dür. Bundan önce bahsettiğimiz kontrollerde her ne kadar bir anlamda hatalar tesipit olsada tespit edilemeyecek durumlarında gerçekleşeceği aşikardır(Örneğin 2 bitin birden değişmesi durumunda tek çift kontrolu sorunlu olacaktır). Checksum kontrolu daha güvenilir ve en uygun kontroldür. Bu kontrolün geliştirilmiş farklı halleride vardır. En basit formu veri gönderilceği yerde toplanır, vardığı yerdede toplanır ve verdığı yerdeki toplamın gönderildiği yerdeki toplama eşit olması gerekir. Eşit olmadığı halde veride hata meydana geldiği tespit edilmiş olunur.

Barry W. Jhonsanın 1984de yazmış olduğu Fault Tolernt Microprocessor makalesini esas alıp son hata tespit kodu olarak Aritmetik koddan bahsedeceğim. Aritmetik kodda data üzerinde aritmetik işlem yapıldığında kontrol edilmektedir. Örnek verecek olursak x datası üreinde A aritmetik işlemi gerçekleştirildiğinde ve y datası üzerinde de aritmetik işlem gerçekleştirecek olursak bu işlem sonunudaki değerlerin çarpımı x ve yenini çarmımının bu aritmetik işleme sokulması sonucu çıkacak olan sonuç ile aynı olması gerekir. Yada başka bir örnekde datatı herhangi bir değer ile çarptığımızda çarpım sonucunda datadaki tüm değerler 3 e tam bölünebiliyor olması gerekir. İşte burada hata kontrolü yapılarak hata tespit kodu gerçekleştirilmiş olunur.

Şimdiye kadar hatalardan ve hataların nasıl tespit edilebildiğinden ayrıntılı bir şekilde bahsettik. Hata toleranslı sistemlerde sadece hatayı tespit etmek değil hataların düzeltilmesi de

gerekmektedir. Gerçek zamanlı düzeltmeler kritik hesaplamalar yapan sistemde operasyonunun herhangi bir interrupt oluşmadan gerçekleşmesidir. Hata düzeltme kodunun bir formu basit olarak temel eşlik şeması ile gerçekleştirilmektedir. Şöyle düşünecek olursak 4 bitlik kelimelerden oluşan birden fazla datayı mesela 5 datayı düşünecek olursak her datayı her satırda bir bit ile değişim olup olmadığını kontrol etmekle birlikte her, dataları alt alta düşününce her sütunu kontrol eden bitlerle işlem gerçekleştiğinde hata meydana gelince hangi satırda ve hangi sütunda hatanını oluştuğuna bakılarak basit bir şekilde hata düzeltilebilri.

Hata düzeltme kodlarından en h-yaygın olanı Hamming kodudur. Bu kod mantığında datanın/kelimenin satır uzunluğu M + k olacak şekilde dir. M data miktarı ve k miktar bit de kontrol biti miktarıdır. Hata meydana gelince k da tutulan değer hangi bitte hata meydana geldiği bilgisidir. Ve hata meydana gelen bite gidilip bit değiştirilerek hata düzeltilmiş olunur. Aşşağıdaki tabloyu inceleyerek Hammington kodunun mantığını daha iyi anlatabileceğiz.

DECIMAL DIGITS	BCD $d_3d_2d_1d_0$	PARITY CHECK BITS $p_1p_2p_3$
0	0 0 0 0	0 0 0
1	0 0 0 1	1 1 1
2	0 0 1 0	0 1 1
3	0 0 1 1	1 0 0
4	0 1 0 0	1 0 1
5	0 1 0 1	0 1 0
6	0 1 1 0	1 1 0
7	0 1 1 1	0 0 1
8	1 0 0 0	1 1 0
9	1 0 0 1	0 0 1

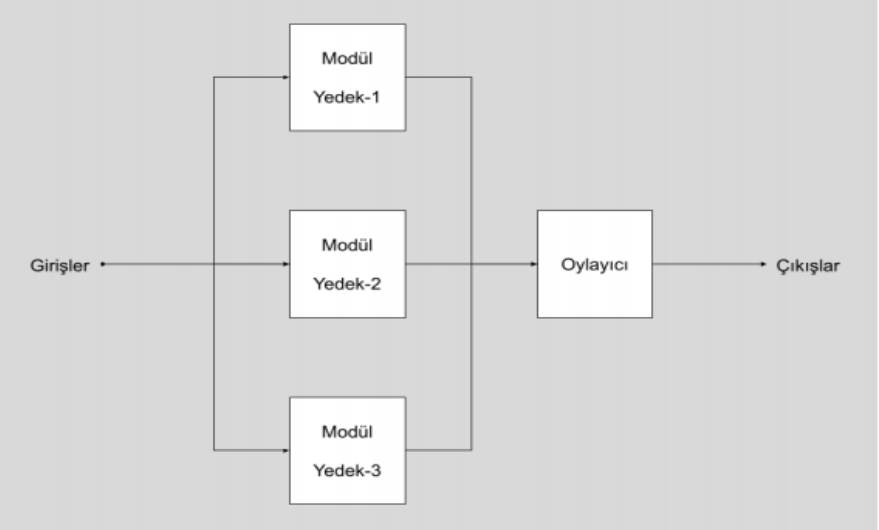
Şekil 3 Hammington hata düzeltme kodu örneği [4]

Tabloyu inceleyecek olursak "Parity check bits" yani P1, P2 ve P3 bitleri kontrol bitleridir. P1 biti d0, d2 ve d3 ü kontrol etmekte, P2 biti d0, d1, d3 bitlerini kontrol etmektedir. P3 ise d0, d1 v d2 bitlrini kontrol etmektedir. Eğer d0 da hata olacak olursa bütün bitler hatayı gösterecektir çünkü d0 ı hepsi kapsamaktadır. d1 de meydana gelcek olan hatayı da d2 ve d3 ün göstereceğini de anlamaktayız.

Bu kısıma kadar anlattığımı hata toleransı tekbiği Information Redundancy (Bilgi Yedekliği/Fazlalığı)ydı. Veri üzerinde bir hata meydana gelebileceği durumda bu hatayı tespit edip tolere etmek amaçlanmış idi.

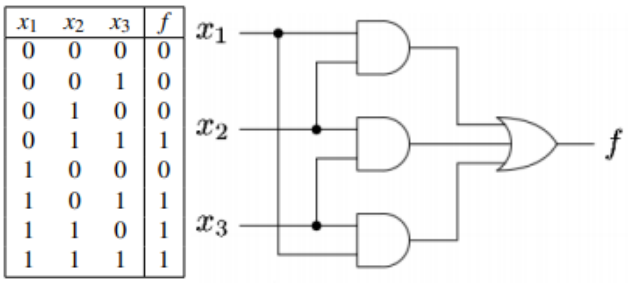
Hardware Redundancy (**Donanım Yedeği/Yedekliği/Fazlalığı**), hata toleranslı sistemlerde en çok kullanılan teknik olmakla birlikte donanım parçalarının arttırılarak kullanıldığı tekniktir. Yarı iletken bileşenlerin daha ucuz maliyet sahip olması ile birlikte daha pratik olmasından ötürü kullanımı yaygınlaşmıştır. Donanım çoğaltma yönteminin temel olarak üç türü vardır. Pasif çoğaltma(passive replication), Aktif çoğaltma (active replication) ve Hibrit Metodu (hybrid method).

Pasif çoğaltma yöntemi hataların oluşumunu maskeler fakat hatayı tespit etme izole etme ve onarma işlevlerini gerçekleştirmez. En yaygın olarak kullanılan metot üçlü modülleme yedeklemesidir(triple modular redundancyTMR).



Şekil 4 Üçlü Modelleme Şeması (TMR) [5]

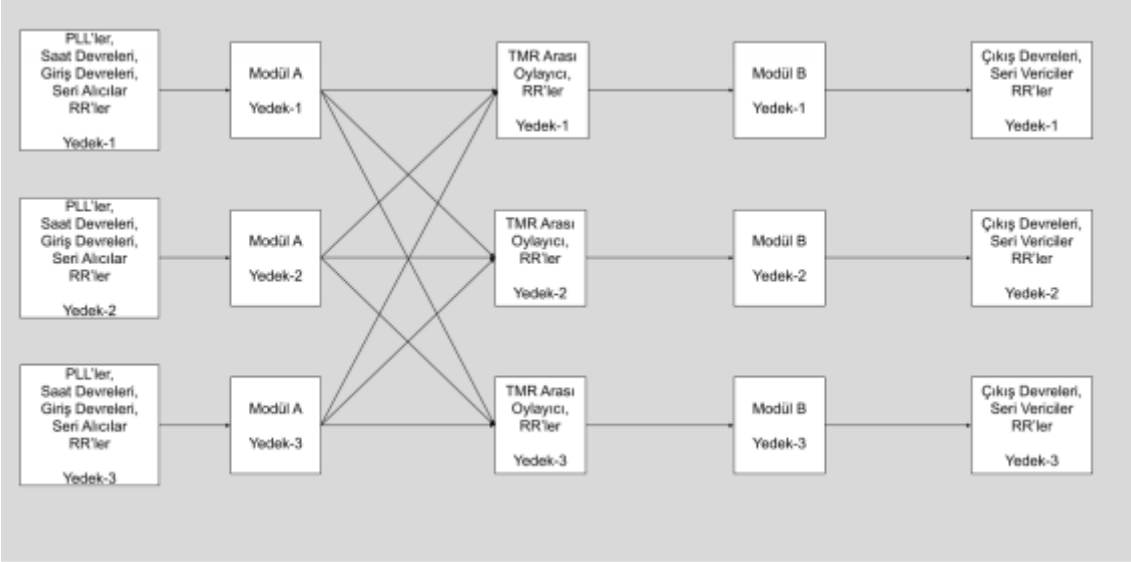
Bu metotda devreler üçlenerek paralele olarak çalıştırılırlar. Doğru çıkışa Oylayıcı(Voter) karar verir. En yaygın oylayıcı standar çoğunluk oylayıcı olarak çalışmaktadır. Gelen modül, sonuçlarından çoğunluk ne ise onu doğru sonuç olarak kabul etmektedir. Eğer devrelerden birinde hata oluşursa Oylayıcı diğer iki devreden gelen sonucun doğru olduğuna karar vererek hatayı maskeler. Bu devre şemasını şekil 4 de görebileceğimiz gibi şekil 5de de doğruluk tablosunu ve mantıksal kapılarla nasıl tasarlandığını görebiliriz.



x_1	x_2	x_3	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Şekil 5 Üçlü Modelleme Doğruluk tablosu ve Mantıksal Devre Tasarımı[6]

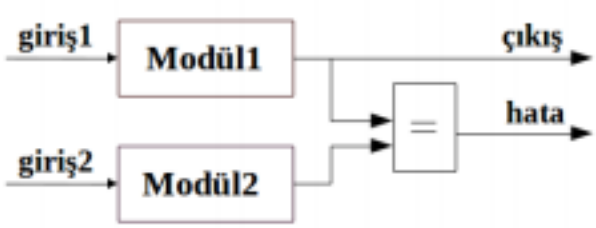
"İdeal Voter(Oylayıcı) oluşturmak imkansızdır" [7] Göstermiş olduğumuzdan basit TMRnin çok daha güveniler olan başka bir formu ise sistemde mümkün olduğu sürece veri yollarının üçlenmesi ile gerçekleştirilir. Havacılık ve uzay uygulamalarında güvenliğin çok daha ön planda olması gerektiği için sonuç oldukça güvenilir actuator (çalıştırıcı)ler ile üretilmektedir. Saat devrelerinde, giriş devreleri ve seri alıcılarda TMR kullanılmaktadır. Şekil 6 da görüleceği gibi bütün oylayıcılar (Voters) modül sonuçlarını alıp doğru sonucu seçmektedirler. Böylece oylayıcı sayısı arttırılarak daha güvenilir bir devre oluşturulmuş olunur.



Şekil 6 Büyük Ölçüde Güvenlikli TMR [8]

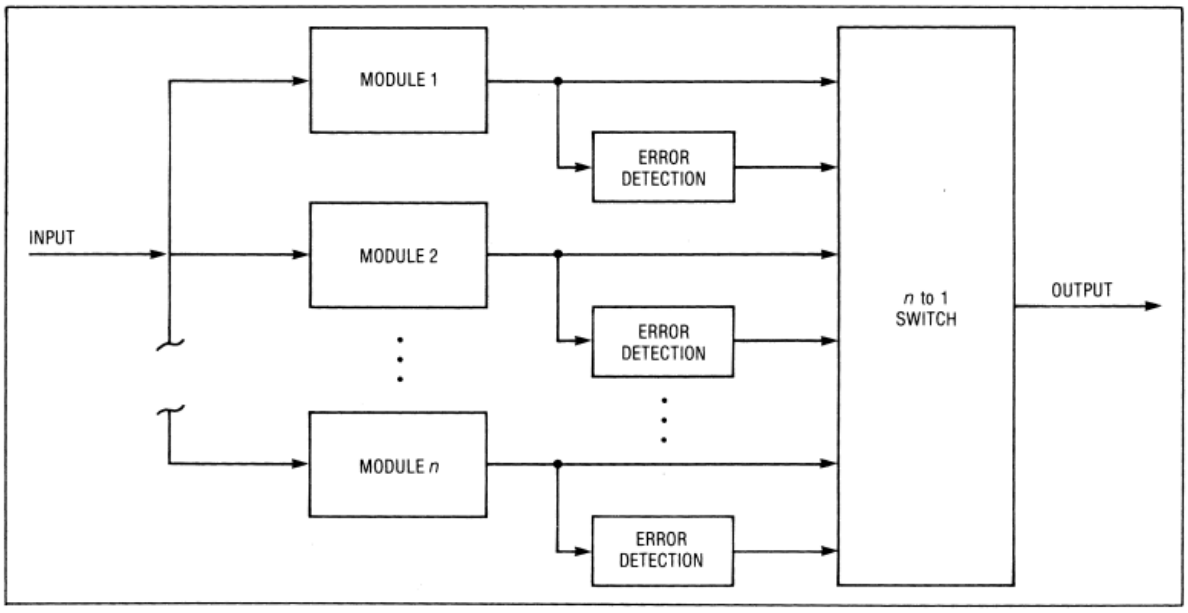
Aktif çoğaltma (active replication) metodu ise hatayı maskelemez fakat hataları tespit edip bulup yedek parça ile hatalı bileşen değiştirilerek sorun çözülmüş olunur. Basit olarak devrenini mantığında aynı moduldenn iki tane bulunmaktadır ve modüllerden çıkan sonuçlar karşılaştırılır. Sonuçların doğru olması beklenirken eğer farklı gelecek olursa hata olduğu

anlamına gelmektedir. Hata tespit edildiğinde birimi tekrar çalışacak hale getirecek bir mekanizma bulunmamaktadır. Böylece sorunu tespit edip çözme pahasına direk hata da pasif replikasyondaki gibi maskelenmemiş olur. Bu durumun şemasını da aşşağıdaki şekilden görebilmekteyiz.



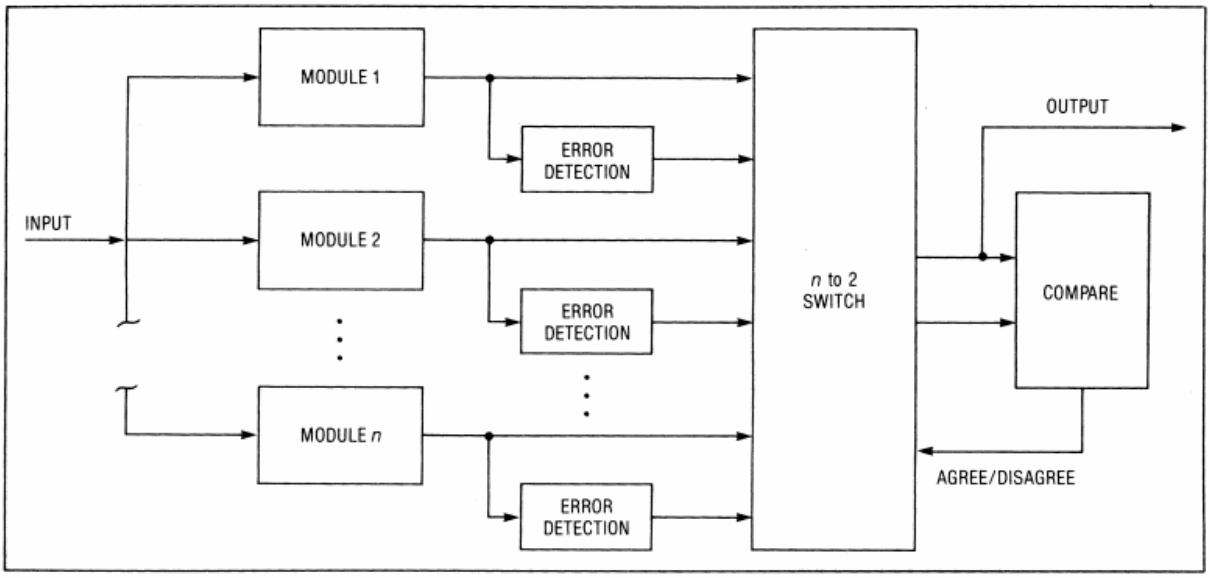
Şekil 7 İşlem Birimini Çoğaltılması-Aktif Yedekleme [9]

Aktif yedekleme metodunun ikinci bir formu da standby replacement dir. Bu metodda birden fazla birimdem biri çalışırken diğerleri yedek olarak çalışmaktadır. Birçok birim için hata tespit şemaları, birimleri kullanılmaktadır. Eğer bir hata tespit edilirse çevrim içi olan hatalı birim kullanılmadan kaldırılır ve yedek birim onun yerine geçer. Bu teknikte bir hata meydana geldiğinde hatalı birim çıkarılıp yerine diğer birimin gelmesi ile sistem işleyişine devam eder fakat hatalı birim çıkarılırken sistem işleyişinde bir kesinti meydana gelebilir. Eğer bu kesinti tolere edilemeyecek bir kesinti ise “hot sparing(sıcak koruma)” kullanılabilir. Bu tip durumlarda yedek parçalar çalışmakta olan ana birim ile senkronize bir şekilde, hata meydana gelir gelmez hatalı birimin yerini alırlar ve kesintinin meydana gelmesi engellenmiş olunur. Bu durum aşşağıdaki devre şeması incelenerek anlaşılabilmektedir.



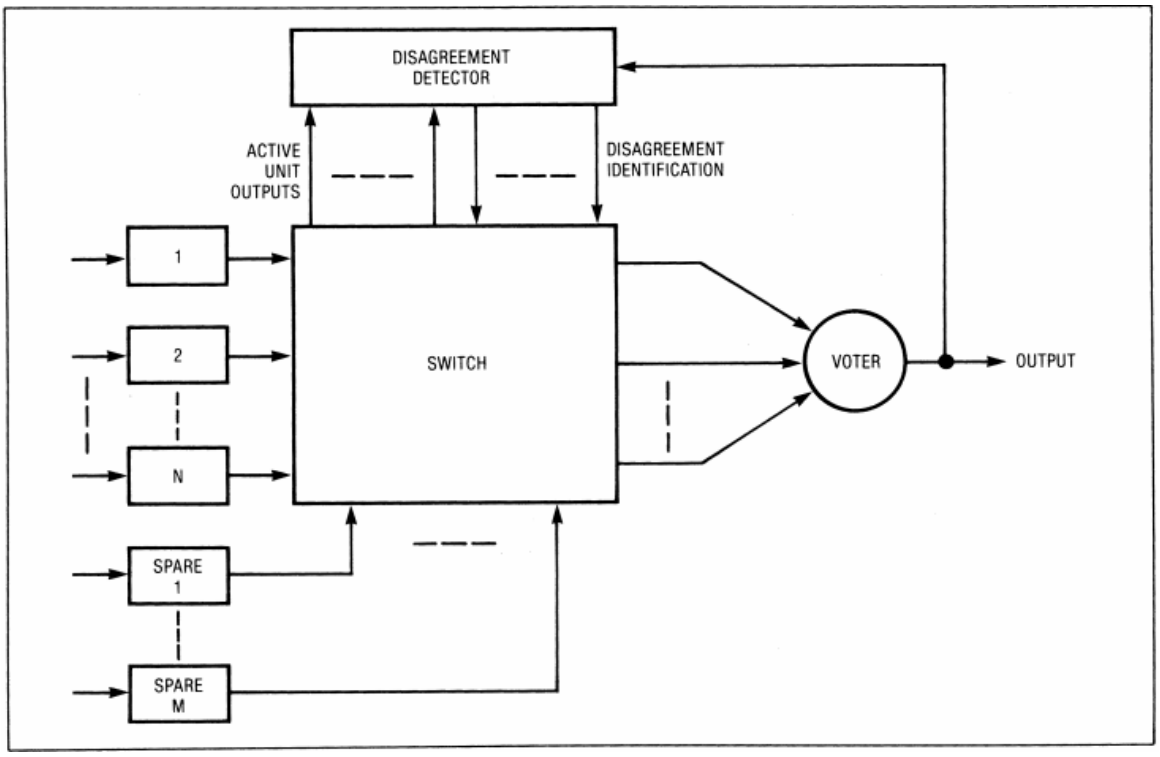
Şekil 8 Hot Sparing(sıcak koruma)- Aktif yedekleme [10]

Aktif yedekleme Metodunun üçüncü ve son formu ise ilk iki formun beraber kullanımıdır. Burada iki birim aynı hesaplamayı yapar. Sonuçlar karşılaştırılır. İki ünite arasında bir tutarsızlık olacak olursa hata olduğu anlamına gelğir ve bir yedek etkinleştirilir. Böylelikle hata tespit prosedürü ile hata tespit edilmiş olunur. Bu konsepte genellikle "pair-and-a-spare (bir çift ve bir yedek)" denmektedir.



Şekil 9 Pair-and-a-Spare [11]

Hibrit yöntem (hybrid method) ise pasif yedekleme(redundancy) ve aktif yedekleme (redundancy) yöntemlerinin birleştirilerek kullanılması ile gerçekleşmektedir. Bu yöntemde hata maskeleme, hatayı ve hata yerini tespit edebilmekle birlikte yedek birimlerle hatalı birimi değiştireme özelliklerine sahip olarak hep pasif yedekleme hemde aktif yedekleme özelliklerini birleştirmiş olur. Bu metodda N modul kullanılarak TMR(3lü modülleme) yerine TMR nin NMR(N modüllü) formu geliştirilmiştir. Sistem yedek parça havuzuna sahip olmakla birlikte NMR çekirdeğine sahiptir denilmektedir. Bir hata meydana geldiğinde hatalı birim çıkarılarak yedek birim havuzundan birim eklenerek sistem işleyişine devam eder. Yedek birim havuzu tükenmedikçe NMR sisteminin güvenliği korunur. Oylama (voter) her zaman aktif birimler arasında gerçekleşmektedir böylelikle hatalı birim maskelenerek sürekli hatasız hesaplar sağlanmış olunur. Böylece hibrit sistemde hep aktif yedeklemnin hemde pasif yedeklemenin birlikte kullanıldığını görmekteyiz.



Şekil 10 NMR (N modüllü Hibrit sistem) [12]

Time Redundancy (Zamansal Yedekleme/Yedekliği/Fazlalığı), sistemde çeşitli fonksiyonlarla gerçekleştirilebilir. İlk teknik esas olarak geçici hataların tespiti için kullanılır. Bir sistemde hataları tespit etmek içib aritmetik yöntemin kullanıldığını varsayacak olursak bir hata tespit edildiğinde iki durum gerçekleşmiş olabilmektedir. Kalıcı hata meydanagelmiş olabilir ve yapılması gereken hatalı birimin kapatılmasıdır. Diğer durumda ise hata geçici olarak meydana gelmiş olabilir. Bu durumda donanım sağlamdır ve işlem birimini direk kapatmak kaynaf istrafı olacaktır. Bu durumda kalıcı ve geçici hataları tespit etmek için Zaman yedeklemesi (Time Redundancy) kullanılmaktadır. İşlemci bir hata tespit edildiğinde hesaplamayı birden fazla kez tekraralayarak bakar ve hata durumunun ortadan kalktığını görecek olursa işlemci hatanın geçici bir durum olduğunu kabul edebilir. Zaman yedeklemesinin zor tarafı ise hata meydana geldikten sonra, gerekli olan işlemin tekrar tekrar gerçekleştirilmesi ve kontrolün yapılabilmesi için aynı dataya ihtiyaç duyulmasıdır. Bir geçici hata meydana gelecek olursa işlemcinin verileri karıştırmamasını sağlamak ve veri tutarlılığını sağlamak zorlaşabilir. Aynı zamana işlemci her durumda farklı kodlama şemaları kullanacak olursa farklı sonuçlar elde edebilir ve böylece her defasında farklı sonuçlar elde etme durumu gerçekleşebilir. Fakat zaman yedekliği tespiti yapabilmesi için çalışma varsayımı, hatanın kullanılan belirli bir koda bağlı olarak meydana gelmesi gerekliliğidir.

Software Redundancy (Yazılım Yedekleme/Yedekliği/Fazlalığı), yazılımsal hataların gerçekleşmesi sonucu bu hataların tespit edilip tolere edilmesi için geliştirilmiştir. Yazılımsal hata meydana gelmesi durumunda hata sistemde yıpranmalara fiziksel sorunlara yol açmaz. Fabrika üretim hatasında gerçekleşen rastkeler sorunlar sonucu oluşmuş değildir. Yazılımın belirli özellikleri hataların kaynağını belirlemektedir. Yazılımsal yedekleme yazılımın bir kopyasına kod geçerliliğini kontrol etmek için küçük programların eklenmesi ile yapılabilmektedir. Bu tip teknikler sistemde bulunabilecek yazılım hatalarına karşılık koruma sağlanması için geliştirilmiştir.

Yazılım yedekleme tekniğinin en yaygın formu geçerlilik yada makul olma kontrolüdür. Üretilen sonuçların belirli aralıklarda olması kontrol edilmektedir. Örnek olarak, bir kontrol sistemi geçerli değerler aralığına sahip bir diziye sahip olup kontrol edilecek soucun bu dizinin bir alt kümesi olması durumunda geçerli kabul edilebileceği aksi taktired geçersis kabul edileceği durum verilebilir.

İkinci bir Yazılım Redundancy ise periyodik olarak işlemin kendi kendini sınamasıdır. Genellikle hataların büyük çoğunluğu bir zamanlayıcı ile periyodik olarak işlem yapılıp hatanın tespit edilmesi ile gerçekleşir. Örnek olarak işlemci tanımlı bir veri kümesi üzerinde işlem yapıp işlemin doğru sonuçlarını read only (sadece okuma işlemi yapılan veri blokları) bloklarında tuttuğu veri ile karşılaştırıp bir tutarsızlık olmadığı durumda zamanlayıcı sıfırlanarak kontrol işlemi gerçekleştirilebilir. Eğer bir hata tespit edilecek olursa zamanalayıcı bir interrupt (kesinti) oluşturacak şekilde tasarlanır. Bu teknik genellikle çeşitli hatalara karşı çok iyi koruma sağlamaktadır. Bu tür Redundancy aslında Yazılım Yedeklemesi ve Zamansal Yedeklemenin karışımıdır. Çünkü ek yazılım gerektirmekde böylece yazılımsal yedeklemesi olmakta, ek işlemci süresi kullanmakta böylecede zamansal yedekleme olmaktadır.

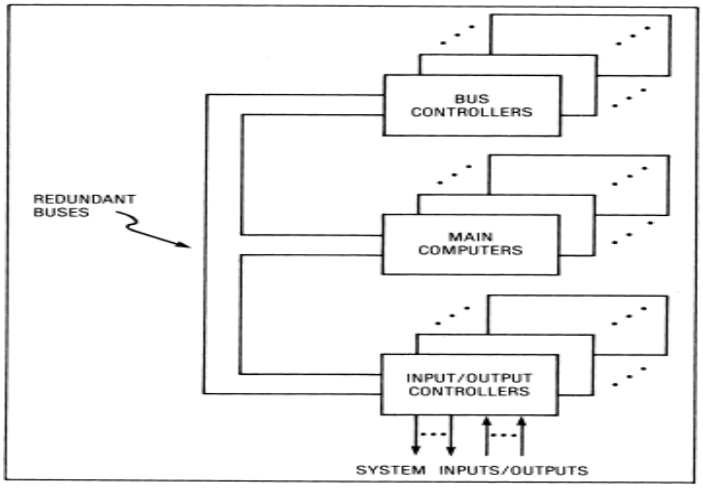
Üçüncü bir yazılımsal yedeklemede ise programın birden çok kopyası kullanılmaktadır. Genellikle bu programlar ortak sorunların ortaya çıkmasını önlemek için farklı takımlar, gruplar tarafından olşturulur. Burada temel mantık farklı yöntemler veya farklı kod satırları ile aynı görevler yerine getirilerek kullanılmasıdır. Bu işleyiş birden fazla işlemci ile yapılabileceği gibi bir işlemci ile de yapılabiliyor olabilir. Elde edilen sonuçlar bir hata olup olmadığını kontrol etmek için karşılaştırılır. Bir fark olmadığı takdirde hata olmadığı anlamına gelmektedir. Bu yazılım yedekliği (Software Redundancy) “N-version programming.” olarak adlandırılmaktadır.

Şimdiye kadar hata türlerinden bahsedip mikroişlemcili sistemlerde hata toleransı teknikleri ayrıntılı bir şekilde işledik. Bundan sonraki bölümde Hata toleranslı sistemler ve tasarımlarından bahsedeceğiz.

Hata Toleranslı Sistem Tasarımı

Daha öncede söylediğimiz gibi hata toleranslı sistemlerde hata toleransı sistemde meydana gelebilecek donanımsal ve yazılımsal gibi hataların sistem işleyişini bozup çökertmesi yerine maskelenerek tolere edilerek sistemin işleyişine devam edebilmesi olduğunu söylemiştik. Hata toleransı olan bütün bilgisayarlar mikroişlemcili sistemler bu tür uygulamalar için tasarlanmış optimize edilmiştir.

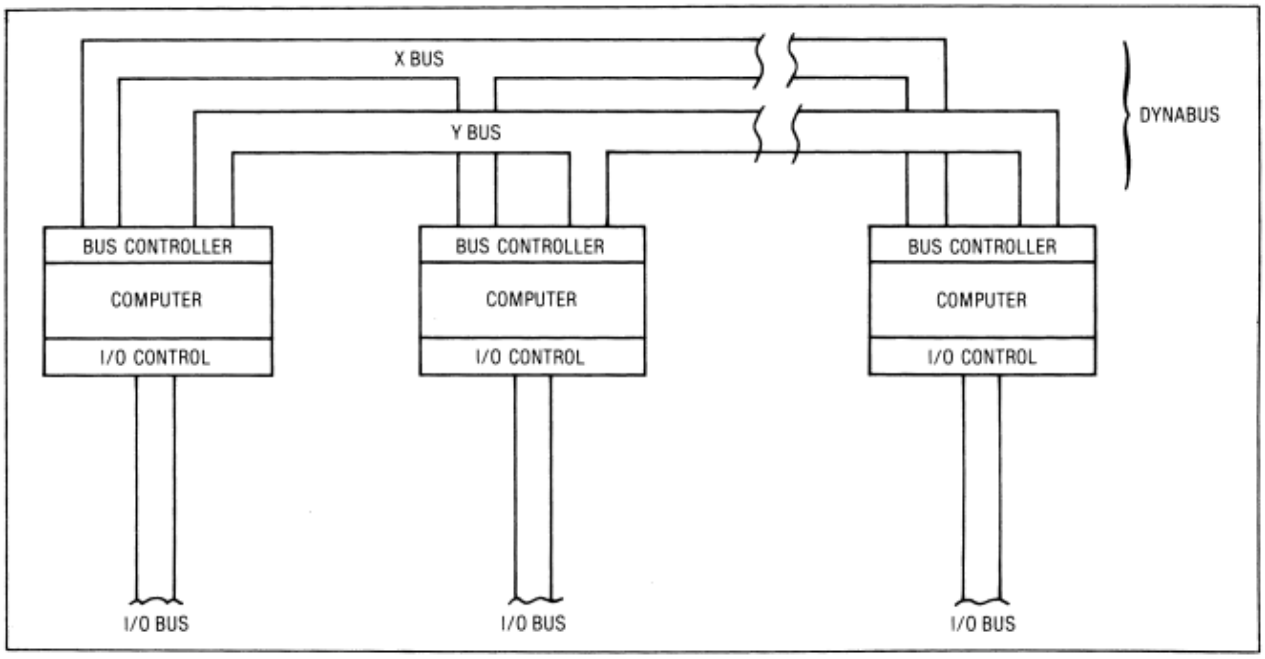
Barry W. Johnson ın makalesinde ilk olarak bahsettiği hata toleranslı bilgisayar **SIFT (Software-Implemented Fault Tolerance)** dir. İlk olarak ticari uçaklar için geliştirilmesi amaçlanmıştır. Dolayısıyla Yüksek derecede hata toleransı kabiliyetine sahip olması amaçlanmıştır. Aynı zamanda kolay test edilebilirlik ve kolay bakım yapılabilmesi hedeflenmiştir. Çoklu işlem mimarisi kullanılarak sistem devre dışı bırakılmadan hataların tolere edilmesi sağlanmıştır. Tüm kritik işlemlerde daha önce anlatmış olduğumuz üçlü modülleme (TMR) kullanılmıştır. Tüm işlem birimlerinin yedekleri bulunmakta ve yedek havuzuna sahip olmaktadırlar. Yedek havuz tükenecek olursa Kritiklik önemi yüksek olan birimlerin düşük olanlara göre daha öncelikli olduğu bir tasarım gerçekleştirilmiştir. Özellikle bir görevin çıktısının başka bir görevin girdisi olduğu yinelemeli durumlarda bu sistem kullanılmaktadır. Aşşağıdaki şemadanda SIFT bilgisayarının şemasını görebilmekteyiz.



Şekil 11 SIFT [14]

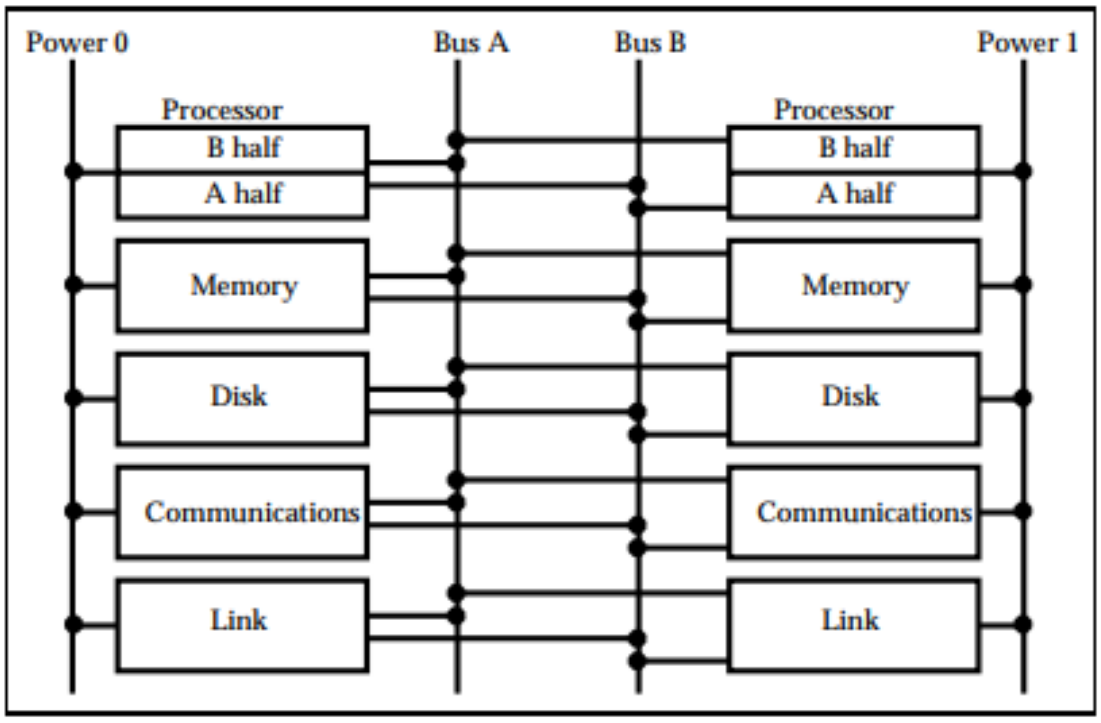
Ganel amaçlı olarak bilgi işlem teknolojisinde yüksek kullanılabilirlik için tasarlanmış ticari olan ilk sistemin **Tandem** 16 NonStop sistemi olduğu söylenmektedir [13]. Bu sistem üç ana hedef doğrultusaunda geliştirilmiştir. İlki otonom arıza tespiti, sistemin kesintisiz bir şekilde yapılandırılmasıdır. İkincisi tüm hataların tolere edilmesi, sistemdfe meydana gelen hata sonucu hemen hatanın yerine yedek modülün geçerek hatnın tolere edilmesi. Üçüncüsü ise sistemi genişletmek amacı ile ve kontrol kolaylığının sağlanıp arayüzü basitleştirmek için modüllerin kullanılmasıdır. Bu sistemlerde donanımsal olarak işlem ve hafıza birimlerinin arttırılarak yedeklenmesi hedeflenmiştir. Tüm modüller arızasız bir şekilde tasarlanmıştır. Bu sistemde ikili bus sistemi ile çift veri yolu oluşturularak birleştirilen 2 ile 16 arasında mikroişlemci bulunmaktadır. Bu sistemlerde hata tespiti için daha önce anlatmış olduğumuz checksum (toplam kontrolü), parity error detection, watchdog timers uygulamaları bus ve hafıza için kullanılmıştır. Bu sistemde her işlemci kendine ait güç kaynağından güç almaktadır. Bu durumun görsel şemasını aşşağıdaki şemadan görebilmekteyiz

.



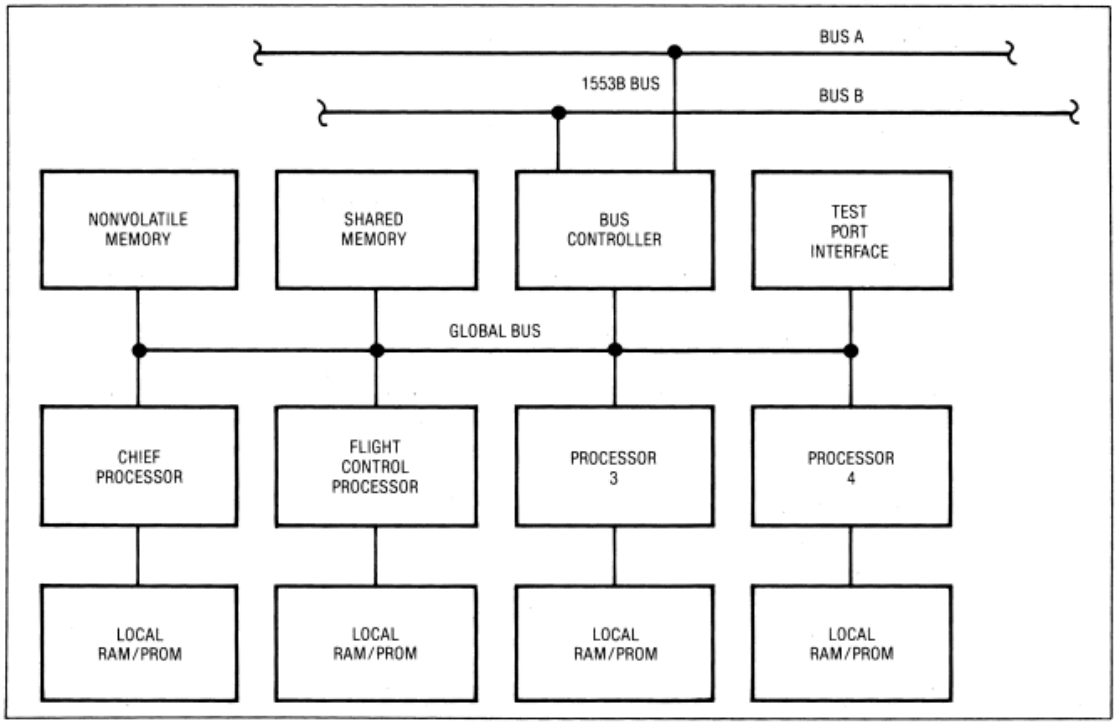
Şekil 12 Tandem [15]

Hata toleranslı sistemlerden bir başkası da **Stratus** dur. Bu sistem de ticari amaçlı geliştrilmiştir. Bu sistem çift yedek bileşen şeklinde tasarlanmıştır. Hatalı bileşn değiştirilene kadar yedek bileşen çalışmaya devam etmektedir. Yazılımda veri hataları görülmemektedir böylece kolay programlanabilirlik sağlanmaktadır ve performans düşüşünü önlemek içinde donanım birimlarinin sayısı arttırılmıştır. Her işlemci kartında iki işlemci bulunmaktadır. İşelmler iki işlemci iel de gerçekleştirilmekle birlikte işlemler gerçekleştirildikten sonra eğre sonuçlar örtüşmüyosa hata olduğu anlamına gemektedir ve işlemciler devre dışı bırakılıp hataların kalıcı mı yoksa geçici hatalar mı olduğuna bakılır. Hataler eğer geçici hatalar ise işlemciler tekrar devreye sokulur. Aynı zamanda aynı işi yapan başka bir kart da bulunmaktadır böylelikle hata meydana geldiğinde sistem işleyişi sekteye uğratılmadan işleyişe devam edilmiştir. Kendi kendini kontrol etme mantığı ile devreler datadaki hatalar kontrol edilmektedir. İşlemci modülleri çoğaltılmış veri yolu ile kontrol edilmektedir. Bu mimari şemasını aşşağıdaki şekli inceleyerek de daha iyi anlayabiliriz.



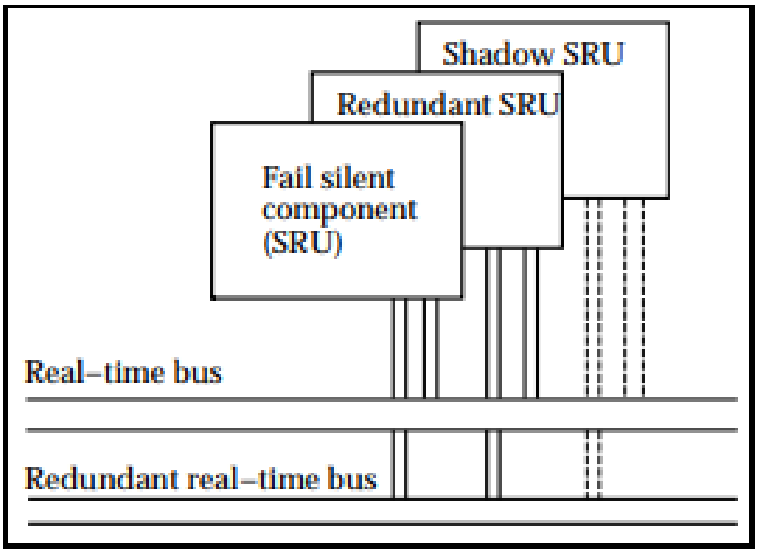
Şekil 13 Stratus [16]

Bir başka hata toleranslı bilgisayar ise **Agusta 129** dur. Bu sistem otomatik helikopter kontrolu için tasarlanmıştır. Birçok işlevi yerine getiren çok işlemcili bir sistemdir. Bu sistemdeki ana birimler şekil 12 deki şemada gösterilen mimariye sahiptir. Herhangi bir anda herhangi bir ana birim yalnızca bir tane 1553B bus yoluna sahip olmaktadır. Anlatılana göre ikinci bir ana birim, aktif ana birimler birlikte çalışmakta ve aynı hesaplamaları gerçekleştirmektedir. Bu sistemde hata toleransı temel mantığında her birim hata tespiti için gerçek zamanlı olarak kendi kendine test yapmaktadır. Eğer işlemciler karşılıklı işlem yapıp sonuçlarını karşılaştıracak olsaydı hatalı birimi bulmak çok zorlaşacaktı. Hatayı bulan işlemci hata birimini devre dışı bırakıp 1553B yolunu başka bir işlemciye devererek işleyiş devam etmektedir.



Şekil 14 Agusta [17]

Son olarak inceleyeceğim hata toleranslı sistem ise **MARS** dır. Bundan önce anlattığımız sistemlerde her nekadar hata toleransı yapılsada zaman hataları konusunda extra olarak hata toleransına sahip oldukları söylenemez. Fakat MARS sistemi zaman hatalarına karşı ekstra yüksek bir hata toleransına sahiptir. Dolayısı ile gerçek zamanlı uygulamalar başarılı bir şekilde gerçekleştirilmiştir. Bu mimaride en küçük değiştirilebilinen birimler SRU(smallest replaceable units) lardır. İki veya daha fazla SRU birleşerek FTU(fault-tolerant unit) yu hata toleranslı birimi oluşturmaktadırlar. MARS sisteminde bir FTU üç SRU dan oluşmaktadır. Burada iki SRU da sorun çıkmadığı sürece üçüncü SRU(gölge SRU olarak / Shadow) bus yoluna iletim yapmaz yani devre dışı olarak yedekte beklemektedir. Çalışan SRU larda hata meydana geldiği taktirde yedekteki SRU hatalı sereunun yerine geçerek hata tolere edilmiş olunur. MARS sisteminde SRU lar iki bileşenden oluşur. Uygulama birimi (application unit) ve İletişim birimi(communication unit). Ek olarak Time yedekliğinde iyi olan bu sistemde elmette daha öncede anlattığımız watchdog timer bulunmaktadır. Aşşağıdaki şekli inceleyerekde MARS sisteminin şemasını daha iyi anlayabilmekteyiz.



Şekil 13 MARS [18]

KAYNAKÇA :

[1] Barry W. Johnson . Fault-tolerant Microp rocessor – based System(1984) s.6-7

[2] Barry W. Johnson . Fault-tolerant Microprocessor – based System(1984) s.7 Figure1

[3] *Barry W. Johnson . Fault-tolerant Microprocessor – based System(1984) s.9*

[4] *Barry W. Johnson. Fault-tolerant Microprocessor – based System(1984) s.10 Table2*

[5] ALİCAN DÖNMEZ(Senior Digital Design Engineer at Qualcomm). FAULT TOLERANT FPGA DESIGN FOR AEROSPACE APPLICATIONS WITH USING PARTIAL DYNAMIC RECONFIGURATION AND TRIPLE MODULE REDUNDANCY(YÜKSEK LİSANS TEZİ) (2019) s.23 Şekil 3-1

[6] Buse USTAOGLU. Hataya Bağışıklı Mikroişlemci Tasarımı (Yüksek Lisans Tezi) (2015) s.26 Şeki 4.3-Çizelge 4.1

[7] Barry W. Johnson . Fault-tolerant Microprocessor – based System(1984) s.10

[8] ALİCAN DÖNMEZ(Senior Digital Design Engineer at Qualcomm). FAULT TOLERANT FPGA DESIGN FOR AEROSPACE APPLICATIONS WITH USING PARTIAL DYNAMIC RECONFIGURATION AND TRIPLE MODULE REDUNDANCY(YÜKSEK LİSANS TEZİ) (2019) s.29 Şekil 3-7

[9] Buse USTAOGLU(). Hataya Bağışıklı Mikroişlemci Tasarımı (Yüksek Lisans Tezi) (2015) s.25 Şekil 4.1

[10] Barry W. Johnson . Fault-tolerant Microprocessor – based System(1984) s.12 Figure6

[11] Barry W. Johnson . Fault-tolerant Microprocessor – based System(1984) s.12 Figure7

[12] Barry W. Johnson . Fault-tolerant Microprocessor – based System(1984) s.13 Figure8

[13] Kjetil Nørv˚ag .An Introduction to Fault-Tolerant Systems. IDI Technical Report 6/99, Revised July 2000 ISSN 0802-6394

[14] Barry W. Johnson . Fault-tolerant Microprocessor – based System(1984) s.15 Figure9

[15] Barry W. Johnson . Fault-tolerant Microprocessor – based System(1984) s.18 Figure12

[16] Kjetil Nørv˚ag .An Introduction to Fault-Tolerant Systems. IDI Technical Report 6/99, Revised July 2000 ISSN 0802-6394 s.13 Figure 6 Stratus Architecture

[17] Barry W. Johnson . Fault-tolerant Microprocessor – based System(1984) s.11 Figure11

[18] Kjetil Nørv˚ag .An Introduction to Fault-Tolerant Systems. IDI Technical Report 6/99, Revised July 2000 ISSN 0802-6394 s.14 Figure 7