BUDAPESTI MŰSZAKI ÉS GAZDASÁGTUDOMÁNYI EGYETEM

Nagy megbízhatóságú elektronikus rendszerek elmélete

Készítette:

Kónya Tamás

Budapest, 2007. május. 14.

Összefoglaló

Ez a leírás a Budapesti Műszaki és Gazdaságtudományi Egyetemen készült „Nagy megbízhatóságú szervomotoros hajtás tervezése” című diplomamunka elméleti részének kivonata. A dolgozat főként hardveres tervezéssel foglakozik, így ez a kivonat is erre fokuszál.

Így ez a munka nem tekinthető teljes értékű leírásnak a nagy megbízhatóság terén, de a szerteágazó irodalomjegyzék révén jó kiindulási alapot nyújt eme speciális terület megismeréséhez.

Amennyiben észrevételük vagy kérdésük van, a tamas PONT konya KUKAC gmail PONT com email címen tehetik ezt meg. Természetesen a PONT helyett '.' és a KUKAC helyet '@' helyettesítve!

Összefoglaló **2**
I. Bevezetés **4**
II. Történelmi áttekintés **5**
III. Hibatűrés elméleti háttere **8**
1. Hibatűrés dimenziói 8
1.1. Általános értelemben vett megbízhatóság (Dependability) 8
1.2. Rendelkezésre állás (Availability) 8
1.3. Megbízhatóság (Reliability) 9
1.4. Hihetőség (Creditability) 10
1.5. Teljesítmény (Performability) 10
1.6. Belső hitelesség (Integrity) 11
1.7. Ellenálló képesség (Security) 11
1.8. Karbantarthatóság (Maintainability) 11
1.9. Tesztelhetőség (Testability) 12
2. Definíciók 12
2.1. Hiba ok (fault), hiba (error), hiba jelenség (failure) 12
2.2. Hiba okok 14
2.3. Hiba tulajdonságok 15
2.4. Hibaterjedés, hibák természete 16
3. Hibatűrő megoldások 18
3.1. Hardveres redundancia 19
3.1.1. Passzív redundancia 19
3.1.1.1. Triple module redundancy (TMR) 19
3.1.1.2. N-modular redundancy (NMR) 21
3.1.1.3. Többségi döntők 21
3.1.2. Aktív redundancia 27
3.1.2.1. Duplikáció összehasonlítással 27
3.1.2.2. Tartalékos rendszerek 28
3.1.3. Hibrid redundancia 32
3.1.3.1. Tartalékolt NMR 32
3.1.3.2. Triplex-duplex 33
Köszönetnyilvánítás **34**
Irodalomjegyzék **35**

I. Bevezetés

„Számítógép egy hatékony eszköz, mellyel sok problémát le tudunk küzdeni. Sőt, a számítógép a mindennapi életünk immáron nélkülözhetetlen része lett. Például egy hibátlanul működő számítógép szükséges a telefonhívások lebonyolításához, a modern repülőgépek és űrrepülők irányításához, a banki tranzakciók feldolgozásához, az atomerőművek felügyeletéhez, de a kritikus állapotú betegek kezeléséhez is.

Mégis, a számítógép – hasonlóan a többi fizikai eszközhöz – képes meghibásodni, aminek következményei a kényelmetlenségtől a katasztrofálisig terjedhetnek. Egy telefonvonalat felügyelő számítógép meghibásodása, átmeneti vonal szakadást eredményezhet, míg egy repülőgép irányító rendszer hibája végzetes balesetet okozhat. Emiatt a számítógép megbízhatósága, hozzáférhetősége és vagy biztonságossága fontos szempontként jelentkezik a tervezése során.

A hiba-tűrő rendszerek fejlesztése a 60'-as évek elejétől bontakozott ki. A folyamat azzal a felismeréssel indult, hogy a digitális rendszerek a meghibásodások széles skálájára érzékenyek. Emiatt a rendszereknek sokrétű hibatűrő megoldást kell alkalmazniuk, hogy egy vagy akár több működést negatívan befolyásoló hiba ellenére is működőképesek maradjanak. Más esetekben a redundanciának képesnek kell lennie, hogy a hiba okozta károsodást minimalizálja, anélkül, hogy a működés folyamatosságát megkísérelné fent tartani. Megint más esetekben, a redundanciának egyszerűen a karbantartás és diagnosztika megkönnyítése a célja, ezáltal csökkentve a rendszer állásidejét. Éppen ezért, a felhasznált hiba-tűrő megoldások száma és fajtája az alkalmazás követelményeitől függ." [Nelson&Carol87]

II. Történelmi áttekintés

A 70'-es évektől, egy újabb ugrással, dinamikus fejlődésnek indult, amely a mai napig töretlen maradt. Ekkor a módszerek, elméletek többségét már legalább húsz éve publikálták. Ezek alkalmazása az évek előre haladtával egyre szélesebb körben vált gyakorivá, így értékelődött át újra és újra a felhasználás gyakorlati oldala. Immár nem csak hadászati, űrrepülés, atomerőműi, banki területeken alkalmazták, hanem a polgári repülésben[1], és az autóiparban[2] is egyre szélesebb körben terjedt a használata.

A hibatűrés kezdetei, egyik "marslakó[3]" matematikusunkhoz vezethetőek vissza. Neumann János 1952-ben egy cikksorozatot publikált, amiben már lerakta az elmélet alapjait. Majd 1956 kiadott összefoglaló írásában a "Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components" [Neumann56] foglalkozik a többségi szavazással és a rendszer hibák elkerülésének lehetőségeivel. Ezen elméleti munkáját az ENIAC[4] programban, majd ennek utódaként szánt EDVAC[5] projektben szerzett tapasztalatai indikálták. Ahol megoldást keresett a több ezer vákuumcsőből álló, éppen ezért megbízhatatlan és állandóan elromló szerkezetek olyan változtatására, mely a rendszert felépítő elemeknél nagyobb megbízhatósággal üzemel, de legalább az eredmény helyességéről nagyobb bizonyosságot nyújt. Ennek eredménye volt az 50'-es évek elején kifejlesztett EDVAC. Ez volt az első elektromos számológép, mely duplikált központi aritmetikai egységgel (ALU) készült. Így az eredmény helyességéről a két egység kimenetének összehasonlításával lehetett meggyőződni.

---

1 Az utasszállító gépek, legtöbb rendszere használ valamilyen megbízhatóság növelő eljárást.

2 Ilyen alkalmazás a steering-by-wire vagy a drive-by-wire.

3 [Marx00] szerepel: "A tudományos elmék azon galaxisa, amelyik az atomenergia fölszabadításán dolgozott, valójában a Marstól érkezett a Földre. Nehezükre esett idegenszerű kiejtés nélkül beszélniük angolul, ez pedig elárulta volna őket. Ezért azt állították magukról, hogy ők magyarok, hiszen közismert e nép azon sajátsága, hogy anyanyelvén kívül semmi más nyelvet nem tud furcsa kiejtés nélkül használni. Ezt az állítást azonban nehéz volna ellenőrizni, hiszen Magyarország oly messze van Amerikától." - Ezek Fritz Houtermans szavai. Ő ismerte föl elsőként a nap- és csillagenergia nukleáris eredetét … vagyis az akkoriban Amerikában dolgozó magyar tudósok (Gábor Dénes, Kármán Tódor, Neumann János, Teller Ede, Szilárd Leó, Wigner Jenő) titkát.

4 Angol betűszó (Electronic Numerical Integrator And Computer) 18000 elektroncsövet és 1500 jelfogóból állt, 1946-re készült el. Az első programozható, elektronikus, digitális számítógép volt.

5 Angol betűszó (Electronic Discrete Variable Automatic Calculator) 6000 elektroncsövet és 12000 diódából állt, 1951-re indult be. Az ENIAC utóda volt, egészen 1961-ig üzemben maradt.

Ebben az időben alkotta meg Antonín Svoboda Prágában az első cseh számítógépet[6], amely háromszoros redundanciát használ (Triple Module Redundancy, TMR), illetve a memóriaegységben hibajavítást is alkalmazott. Ha hiba merült fel, a gép automatikusan megpróbálta újra végrehajtani a műveletet. [Rennels99]

Az 50'es években megjelent tranzisztor magával hozta a technológia nagyobb megbízhatóságát, ezáltal a megbízhatóság növelő megoldások kezdődő fejlődése egy lélegzetvételnyi időre megállt. Ugyanis az új eszköz alkalmazása esetén, a megbízhatóság már nem jelentett kritikus problémát, mint ahogy az a vákuumcsöves áramköröknél korábban jelentkezett. Változást csak az űrprogramok felfutása jelentette, ugyanis a követelmények növekedésével, a tranzisztoros technológia folyamatos fejlődése sem tudta tartani a lépést.

Az OAO[7] műholdak és a korai Apolló program irányító egységei, az egyik legutolsó számítógépek voltak, amelyek diszkrét tranzisztorokból épültek fel. Ahol minden tranzisztor helyett, egy négy tranzisztorból álló soros-párhuzamos áramkört használtak, amely egy tranzisztor meghibásodása esetén még működőképes maradt. [Rennels84] Ezen rendszerek után az űrkutatás, már csak hibatűrő megoldásokat használt az űreszközökben.

Skylab[8] volt az első „szokványos"[9], azaz a polgári alkalmazásokba is kapható alkatrészekből felépülő vezérlési rendszerű űreszköz. Architektúráját tekintve egy blokk redundáns szimplex rendszerről volt, ami e mellett meleg tartalékkal is rendelkezett. Az elsődleges és a meleg-tartalék is az IBM 4Pi/TC-1[10] rendszert alkalmazta. [Caldwell98]

Miután az űrkutatás megtette az első lépéseket, a polgári alkalmazása sem váratott sokáig magára. A 60'-es évek közepétől megjelennek a telekommunikációban az elektronikus kapcsolóközpontok (Electronic Switching System, ESS). A Bell Laboratories No. 1 ESS rendszere, duplikált központi egységgel, a hibák lokalizálására még szoftver és hardver háttérrel is rendelkezett. A rendszer sikerén felbuzdulva a rendszert folyamatosan továbbfejlesztették. [Serlin84]

---

[6] A memória mágneses volt, míg a logika relés.

[7] Angol betűszó (Orbiting Astronomical Observatory), NASA szatellitek a 60'-as években főleg UV-tartományú megfigyeléseket végeztek, ezzel megalapozták az űrbe telepített csillagászati megfigyelést, azaz a mai Hubble űrtávcső elődjének tekinthetőek.

[8] Az első űrállomás, 1973-tól 1979-ig működött, három missziót szolgált ki.

[9] A „szokványos" kifejezést a szakzsargon a polcról levehető, azaz „Commercial of The Self" (COTS) néven használja.

[10] Az IBM 360 rendszerének robusztus változata.

A katonai repülőgépen jelentek meg először[11] a Digital-Fly-By-Wire, azaz a teljesen elektronikus repülőgép irányító rendszer, amely leváltotta a nehéz és sérülékeny hidraulikus rendszereket és új dimenziókat nyitott meg a repülőgépek előtt. Ugyanis napjainkban már olyan katonai gépek (LM F-117 Nighthawk) is repülhetnek, melyek a számítógép felügyelete nélkül instabilak, azaz repülésre képtelenek lennének. Hasonló digitális irányítástechnikai rendszerek jelentek meg a nagy utasszállító gépeken is (pl.: Boeing, Airbus). Ezeknél rendkívül komplex és kiterjedt rendszerekben a vezérlés és az irányítás a komplexitásából adódóan elképzelhetetlen lenne számító Bevezetés gép segítsége nélkül. A rendszerek megbízhatósága megkérdőjelezhetetlen, ezért komplex hibatűrő rendszereket alkalmaznak. Ilyen például a Boeing 777 sorozatú repülőgépein a Primary Flight Computer (PFC) egység is, mely három különböző processzort[12] használ, három különböző architektúrában. [Yeh96]

Eközben persze más polgári alkalmazásban is megjelent az igény a megbízhatóság növelésére. A hatalmas mennyiségű, és fontos adatokkal dolgozó bank szektor is kereste a megbízható számítógépes megoldásokat. Sorra alakultak a hibatűrő megoldásokat szállító vállalkozások (pl.: Tandem Systems, ma a HP része). Majd 1971-ben az Institute of Electrical and Electronics Engineers (IEEE) is megtartotta az első, a témával foglalkozó konferenciáját (Fault-Tolerant Computing Symposium, FTCS) néven.

Napjainkban a mikrokontrollerek, FPGA-k (SoC[13]) és egyéb digitális áramkörök olcsóvá válásával és a számítógépes CAD[14] rendszerek hathatós támogatásával széles körben foglalkoznak hibatűrő áramkörök fejlesztésével (pl.: nano műholdak, polgári űrhajó, „home-made” repülőgépgyártás). Emellett a félvezető iparban is egyre többször merül fel e technológiák alkalmazása az egyre kisebb méretű (néhány atomos) és ezért egyre megbízhatatlanabb tranzisztor technológia alkalmazása kapcsán. Ebben az iparágban az első ilyen megoldás a memória gyártás területén (tartalék sorok és oszlopok) az IBM-nek köszönhető.

---

11 1972-ben repül először az átalakított F-8C Crusader, berepülő pilótája Neil Armstrong.
12 AMD, Intel és Motorola
13 Angol betűszó, System on a Chip
14 Angol betűszó, Computer Aided Design

III. Hibatűrés elméleti háttere

Ebben a fejezetben a hibatűrés, azaz az elektronikai rendszerek megbízhatóságának növelési stratégiáinak elméleti hátterével fogok foglalkozni. Ami felöleli a témakör pontosabb meghatározását és definíciók bevezetésével a témakör tárgyalását teszi lehetővé.

1. Hibatűrés dimenziói

A hibatűrés tág fogalom, túl általános. Így ebben a fejezetben ennek kisebb témakörökre bontásával szeretném pontosítani az általános fogalmat és megvilágítani annak egyes esetekben igen különböző – néha egymásnak ellentmondó – követelményeit.

1.1. Általános értelemben vett megbízhatóság (Dependability)

A megbízhatóság fogalmának általános és komplex meghatározója. Benne foglaltatik az összes ebben a témakörben tárgyalt meghatározás, azaz a rendelkezésre állás (availability), hihetőség (creditability), megbízhatóság (reliability), karbantarthatóság (maintainability), belső hitelesség (integrity) és az ellenálló képesség (security). Az összemosó jellegéből fakadóan jól jellemzi a rendszert a hibatűrés szempontjából, de éppen az összefoglaló mivolta a gyengéje is, mivel egy-egy alkalmazás más és más mértékét követeli meg a fent felsorolt dimenziókból, így egy bank automatát és egy műhold irányító rendszert e mérték alapján nehézkesen lehet csak összehasonlítani. [Johnson89]

1.2. Rendelkezésre állás (Availability)

Mint azt a neve is sugallja, a rendelkezésre állás a rendszer leállása és a működése közti idő hányadosa. Szabatosan megfogalmazva, egy időfüggő változó, mely megmutatja, hogy a rendszer milyen valószínűséggel működik helyesen. Nagymértékben hasonlít a megbízhatóság (reliability) fogalmára, de az előbbi egy időpillanatra vonatkozik, míg utóbbi egy időtartományra.

$A = \frac{MTTF}{MTBF}$, rendelkezésre állás definíciója, MTTF mean time to failure, MTBF mean time between failure

Jellemezni leginkább a teljes működési időre jutó leállási idővel szokták. Erre példaként hozható a 60'-as évek közepén gyártott Bell Labs No. 1. ESS telefon kapcsolóközpontja, tervezett rendelkezésre állása 40 év alatt mindössze két órányi leállás (A = 0,999995). [Johnson89]

1.3. Megbízhatóság (Reliability)

Annak a feltételes valószínűsége, hogy a rendszer adott $[t_0, t]$ időtartamban működő képes, feltéve, hogy a t_0 időpontban még az volt. Gyakorlatilag annak az esélyét becsli, hogy az egység működőképes az adott időtartam alatt. A meghibásodási hajlandóság $F(t)$, a megbízhatóság $R(t)$, ezek komplementer események, azaz a két függvény összege mindig egy.

$$R(t) + F(t) = 1 \qquad F(t) = 1 - R(t)$$

A megbízhatóság nagyon jól leírja a rendszer működését, de ezt két paraméter mentén teszi, a kérdéses időtartományon, adott működési valószínűséggel. Vannak rendszerek ahol a rendkívül hosszú működési idő szükséges, karbantartási lehetőség nélkül. Tipikus példa erre a missziós rendszerek. Ilyen például a Voyager 1 és 2, melyek lassan harminc éve állnak szolgálatban és lassan elérik Naprendszerünk szélét is. Illetve vannak olyan rendszerek is, ahol rendkívül nagy[15] megbízhatósággal kell üzemelniük, de azt csak órákban kifejezhető időtartamokra kell biztosítaniuk. Ilyenek a repülőgépek rendszerei, melyek a repülési időn kívül szervizelhetőek ugyan, de repülés közben ez szinte kizárt.

Fontos hangsúlyozni azt is, hogy a hibatűrő és a megbízhatóság nem ekvivalens fogalmak, ugyanis az előbbi egy technika, mely alkalmazásával nem megbízható elemekből egy jobb paraméterekkel rendelkező rendszer állítható elő. Míg utóbbi egy tulajdonság, mellyel rendelkező alkatrészekből felépülő rendszer nem feltétlen lesz hibatűrő. [Johnson89]

---

[15] 0,9999999 valószínűséggel, ezt úgy is szokás mondani, hogy hét kilences, azaz $0,9_7$

1.4. Hihetőség (Creditability)

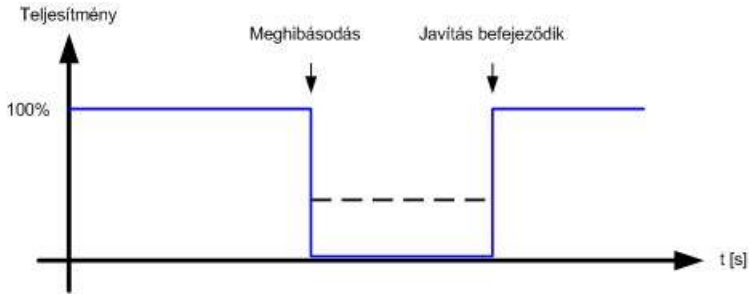
Annak a valószínűsége, hogy a rendszer pontosan végre hajtja a feladatát, vagy felfüggeszti a működését, de más lehetőség nem jöhet szóba. Azaz hibás működés során elhallgat, így a külvilágba hibás kimenet nem juthat és ezáltal más egységek működését nem lehetetlenít heti el. A hihetőség a fail-safe, fail-silent képesség mérőszáma. Ez a módszer azt biztosítja, hogy az eszközök determinált módon hibásodjanak meg. Ilyen lenne például egy hajszárító, mely vízbe esve tönkre menne ugyan, de az éppen ott fürdőzőt nem rázná meg.

A hihetőség különbözik a megbízhatóságtól, az utóbbinál vagy működik, vagy nem, de az előbbinél vagy helyesen működik, vagy oly módon hibásodik meg, hogy hibás kimenetet nem produkál. [Johnson89]

1.5. Teljesítmény (Performability)

Sok esetben lehetséges, hogy a rendszer egy-egy komponense (hardver vagy szoftver) meghibásodása esetén is folytassa a működését, esetleg csökkentett funkcionalitással. Esetleg csak a legfontosabb feladatokat hajtva végre, vagy nem olyan pontosan hajtva őket végre, de a hangsúly a működőképesség megőrzésén van. Ez már sokszor nagy segítséget jelent, mint ahogy az a Vega szondák esetében is volt. A navigációs rendszer a Haley-üstökös helyzetének meghatározására három szenzor állt rendelkezésre, egy 512x576 CCD, egy tartalék 512x288 CCD, míg a legtovább üzemképes szenzor a tartalék-tartaléka (azaz második tartalék) rendszer volt, mely mindössze két fotodiódából állt. [Szalai94]

A teljesítmény, angol szakszóval performability az a mérőszám, amely a rendszer teljesítőképességét fejezi ki az idő függvényében. Mint az látható az 1. ábra is, a hibatűrő technikák nélkül (folyamatos vonal) meghibásodás esetén 0%-ra esik vissza a teljesítmény, míg megfelelő technikával (szaggatott vonal) a teljesítmény esetleg csökken ugyan, de a működőképesség bizonyos foka továbbra is megmarad. Egy esetleges karbantartással az eredeti teljesítmény visszaállítható.



1. ábra: Teljesítmény alakulása hibatűrő technikával és anélkül

1.6. Belső hitelesség (Integrity)

„Annak biztosítása, hogy a rendszer taszkok a rendszer normális állapotaiban helyesen hajtódnak végre, és minden olyan normális állapotban jelzés keletkezik, amelyikből nem normális állapotba átmenet lehetséges.” [Selényi01]

A gyakorlatban számos példa van ilyen megoldásokra. Ilyen például, ha a fel nem használt program memória területét feltöltjük egy hibajelző címre ugró utasítással, vagy ha egy folyamat kiszámított értéknek – esetleg előre meghatározott vagy egy egyszerű és biztos módszerrel futásidőben becsült – tartományon belül kell lennie, ellenkező esetben hibajelzés és/vagy hibareakció következik.

1.7. Ellenálló képesség (Security)

„Annak a biztosítása, hogy a rendszer képes felismerni a jogosulatlan vagy hibás inputot és azt jelzi.” [Selényi01] Itt szintén használhatóak a fent említett hihetőségi tartományok értékekre és időtartamokra vetítve, illetve a bejövő minták átlagolása, szavaztatása. De egy egyszerű hardveres EMC szűrő is ellát megelőző feladatot – igaz jelzés nélkül teszi ezt –, ugyanis a rendszerben keletkező EM zajok, hibás bemenetet okoznának és ezek kiszűrésével a rendszer ellenállóbbá válik ellenük. A belső hiteleség (1.6 fejezet) a nevéből adódóan a belső folyamatokat, míg az ellenálló képesség a külső folyamatokból felügyeli.

1.8. Karbantarthatóság (Maintainability)

A legtöbb – a missziós rendszerek képeznek kivételt – készüléknél fontos szempont a javítási idő (Mean Time To Repair, MTTR) csökkentése. A javítási folyamat rövidítésével növelhető a rendelkezésre állási tényező. Gondoljunk csak egy bank

automatára egy forgalmas bevásárló utcában, ahol minden üzemen kívül töltött idő jelentős problémát jelent. E dimenzió javítható a rendszerbe gyárilag beépített diagnosztikai eszközökkel vagy menet közben cserélhető, esetleg amúgy is redundáns modulokkal. [Johnson89] A telefonközpontok, az internetet kiszolgáló szerverek többszörözött moduljai esetleg teljes egységek állnak rendelkezésre egy meghibásodás esetén, amelyek átveszik az egyébként kieső funkciókat. Ilyen alkarész lehet, a kiskereskedelmi forgalomban is kapható RAID-elt menet közben cserélhető (hot-swap) merevlemez is.

1.9. Tesztelhetőség (Testability)

A megtervezett rendszerek tesztelése a rendszer minőségének a záloga. A prototípus funkcióinak tesztelésétől, a lefektetett mutatók teljesítésének ellenőrzésén és a menet közbeni meghibásodások gyors felderítésén – a tesztelés során gyűjtött tapasztalat gyorsítja a karbantartást – át a folyamat, a megbízhatósági jellemzők növelésének egyik fontos módszere.

A rendszereknek célszerűen tartalmazniuk kell a mag funkciókon kívül, olyan további szolgáltatásokat (szoftveres és hardveres), melyek a belső állapotokat elérhetik (állapot megfigyelhető) és módosíthatják azokat (állapot vezérelhető). Az előbbi amely csak monitorozza a belső állapotokat – kívülről irányíthatatlan, de a működést jól leíró belső változók például –, az utóbbin keresztül pedig a belső állapot meg is változtatható – például szándékosan előidézett hibák kezelésének ellenőrzésére – . Persze a legjobb megoldás, ezek kombinációja, amely a beavatkozástól a megfigyelésig mindenre lehetőséget ad. E funkciók jelentősen csökkenthetik nemcsak a tesztelési, hanem a javítási időt is, így a karbantarthatósági (1.8 fejezet) mutató is javul .

2. Definíciók

2.1. Hiba ok (fault), hiba (error), hiba jelenség (failure)

„Három alapvető fogalom van a hibatűrés témakörében, ezek a hibaok (fault), hiba (error), és a meghibásodás (failure). Ezek ok-okozati összefüggésben állnak egymással. A hibaok az a hiba okozója, ami pedig a meghibásodás oka.” [Johnson89]

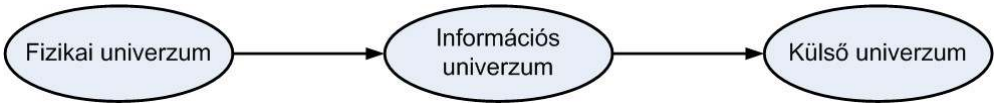
2. ábra: Hibaok – Hiba – Meghibásodás kapcsolata

A *hibaok* jelentése megegyezik azzal, amit az értelmező szótárban a hiba címszó alatt találunk, vagyis a későbbi hibák bizonyított vagy feltételezett oka. Hardveres komponens esetén tipikusan rövidzárlat, szakadás, paraméterváltozás sorolható ide, míg szoftveres komponensnél a nem várt bemeneti kombinációk, ciklusban ragadás, elcímzés, hogy csak párat említsek a lehetséges okok közül. A hibaokra (fault) egy példa, ha egy szennyeződés (mikró méretű porszemcse a félvezető szilíciumon) hatására megváltozott áramköri tulajdonságok miatt egy AND kapu bemenete ’1’ logikai szinten ragad.

A *hiba* a hibaok következményeként már megjelenik az eszköz belső állapotában is. Képzeljük el, hogy eddig a fenti AND kapura logikai ’1’ kapcsoltunk. Vagyis a hibaok nem tudott tovább terjedni, mert az eredeti bemeneti jel megegyezett a beragadt lábon lévő jellel. Előbb-utóbb ez a jel megváltozik és ’0’-t vesz fel. Ez a változás már nem jut el a kapu bemenetére – hiszen az logikai '1' állapotban ragadt –, így a szándékolttal ellentétes bemeneti jel hatására az AND kapu kimenete nem változik. Ezzel hiba (error) megjelenését okozza a rendszerben.

A *meghibásodás*, amikor a hiba kijut a rendszer kimenetére. Ez a fenti példával élve, azt jelenti, hogy eddig a logikai függvényben a kapu kimenete nem érvényesült, de ezután a kapu kimenete – áttételesen ugyan – meghatározza a rendszer kimeneti jeleit. Ezáltal a hiba a külvilágba kerül, és meghibásodássá (failure) válik.

A folyamat fenti definícióját magyarázza a 3. ábra, amelyik teljesen analóg az előzővel, de rávilágít, hogy a hibák három szintje három különböző közegben jelenik meg. A hibaok (fault) jellegéből fakadóan fizikai, a hiba (error) már a rendszer belső állapotát változtatja meg, azaz információs jellegű, míg a meghibásodás (failure) már alapvetően a külvilágot érinti. A 3. ábra [Johnson89] tree-universe[16] alapján.



3. ábra: Hibák szintjei

[16] Az angol szókapcsolat jelentése, három-univerzum. Ennél a modellnél léteznek azonban bővebb modellek is a szakirodalomban, amelyek a hibák keletkezésével és terjedésével foglalkoznak.

„A hibaok keletkezése számos dologra vezethető vissza. Külső zavaró hatásra vagy anélkül az elektromos alkatrészekben, illetve a rendszer vagy az alkatrész tervezése közben elkövetett hibák következményeként felléphetnek hiba okok. Ezért nagyon fontos, hogy megértsük az összes, lehetséges hibaok keletkezési módját. Végig követve egy tervezési folyamat lépéseit azonosíthatóak a hibaok keletkezési helyek.” [Johnson89]

2.2. Hiba okok

A tervezési folyamat a specifikációkészítéssel kezdődik, amennyiben az első lépések során nem sikerül a feladatot jól specifikálni, az egyben hibás koncepcióalkotást is eredményez, ami pedig a későbbiekben végig kíséri az egész tervezési folyamatot. A hiba még a késztermék megfelelőség vizsgálata során sem kerül a felszínre – a vizsgálat a kezdeti specifikáció alapján történik –, így nem derülnek ki a hibák, csak az alkalmazás éles bevetése közben. Amikor is már meghibásodás jelentkezik.

A *specifikációs tévedésben* nem csak rendszer elsődleges specifikáció során vétett hibák tartoznak bele, hanem annak mélyebb elemei is, úgymint a de komponált funkciók összehangolását végző időzítési, szintillesztési problémák, hogy csak néhányat említsek.

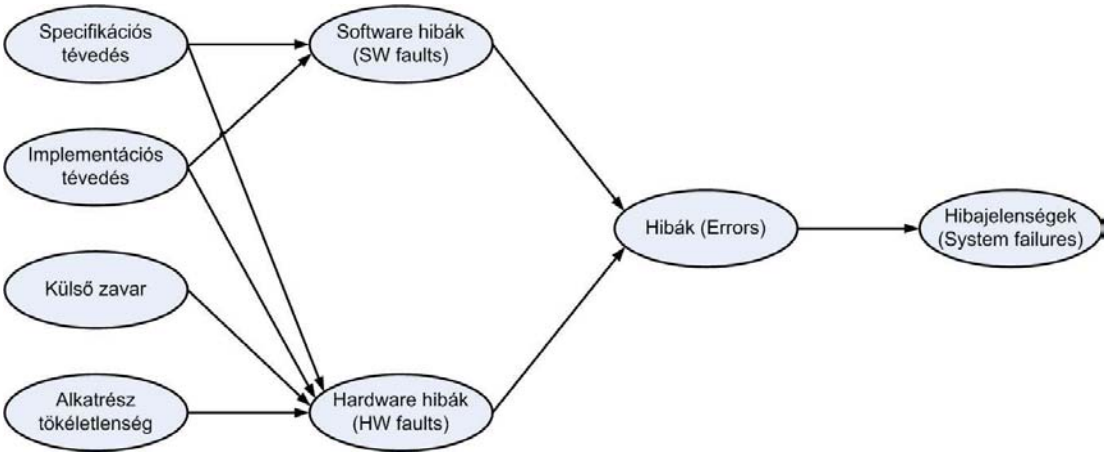
Az *implementációs tévedések* akkor fordulhatnak elő, amikor a specifikáció átültetésre kerül a gyakorlatba. Nem megfelelő, vagy rossz alkatrészválasztás, rossz tervezői döntés, vagy a kódolás során elkövetett hibák miatt. Például egy magas szinten (pl.: UML, szimulációs modell) tesztelt és hibátlannak minősített szoftver megvalósításában elkövettet kódolási hiba, amiknek okozója a programozó vagy a fordítóprogram is lehet. Bevezetés

Ugyanakkor *alkatrész tökéletlenség* a leggyakrabban előforduló jelenség. Hiszen két azonos típusú alkatrész sem egyezik meg teljesen, ugyanis paraméterei enyhén eltérhetnek és ezen eltérések adott tartományon kívül már jóval nagyobb értéket is felvehetnek. Ezt a jelenséget a tervezés során mindenképpen figyelembe kell venni. Azonban körültekintő tervezéssel vagy válogatással ez még könnyen orvosolható, nem úgy az alkatrészek véletlenszerű meghibásodásából adódó problémák. Ennek tipikus okai a mikroelektronikai alkatrészek esetén a tokozáson belüli kötések megszakadása, a fémkorrózió, vagy a elektronikai hordozó esetén a NYÁK-gyártási tökéletlenségek vagy a működési feltételek szélsőségei. Az alkatrész tökéletlenség meghibásodási típushoz

sorolható az alkatrészek öregedés miatti meghibásodása is, amikor is az alkatrészek veszítenek maximális teljesítőképességükből és folyamatosan csökkenő ütemben vehetőek csak igénybe.

A *külső zavaró hatások* elleni védekezésére épp oly hangsúlyt kell fektetni, mint az előző négy hibaforrásra. Hiszen hatásuk kiszámíthatatlan, de előrelátó tervezéssel az ellenük való védekezésre is mód nyílik. Külső zavaró hatások közé sorolhatóak az elektromágneses zavarásból, sugárzásból, operátor tévesztéséből, fizikai (harci) sérülésből vagy környezeti extrémitásokból (rezgés, hőmérséklet, por, nedvesség) adódó meghibásodások.

Az 4. ábra [Johnson89] összefoglalva láthatóak a fent felsorolt hibaok (fault) előidéző események. Az ábra jól illeszkedik az előző három univerzum (hibaok – hiba – meghibásodás) modellbe.

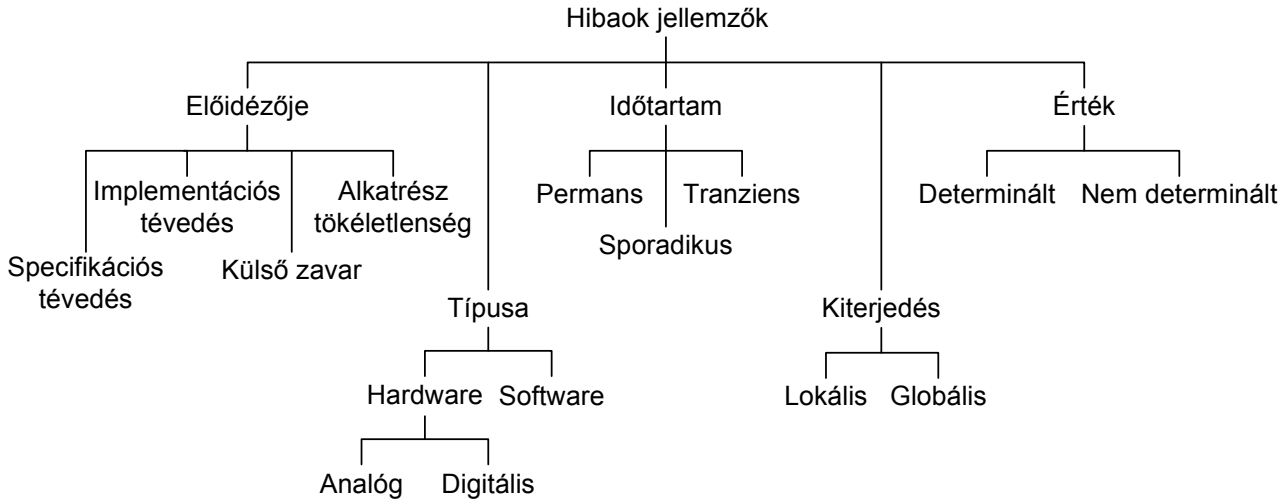


4. ábra: Hibák létrejöttének láncolata

2.3. Hiba tulajdonságok

Ha az előbbi útmutatást sikerül is betartani, az nem jelenthet tökéletes védelmet a hiba okok megjelenése ellen. Amennyiben sikerülne megérteni a hibaok keletkezésének mikéntjét, jobb eljárásokat lehetne kidolgozni a hibák kialakulásának kezelésére, felhasználóként és alkatrész gyártóként egyaránt. De addig is a hiba (error) megjelenése után is be kell avatkozni, hogy a rendszer működését fenntartsuk, de ekkor a hiba már megjelenik a rendszerben, ennek kiküszöbölésére el kell fedni azt.

Első lépéseként, meg kell ismerni a hibaok (faults) fajtáit, ami az 5. ábra látható. [Johnson89]



5. ábra: Hibák csoportosítása jellemzőik alapján

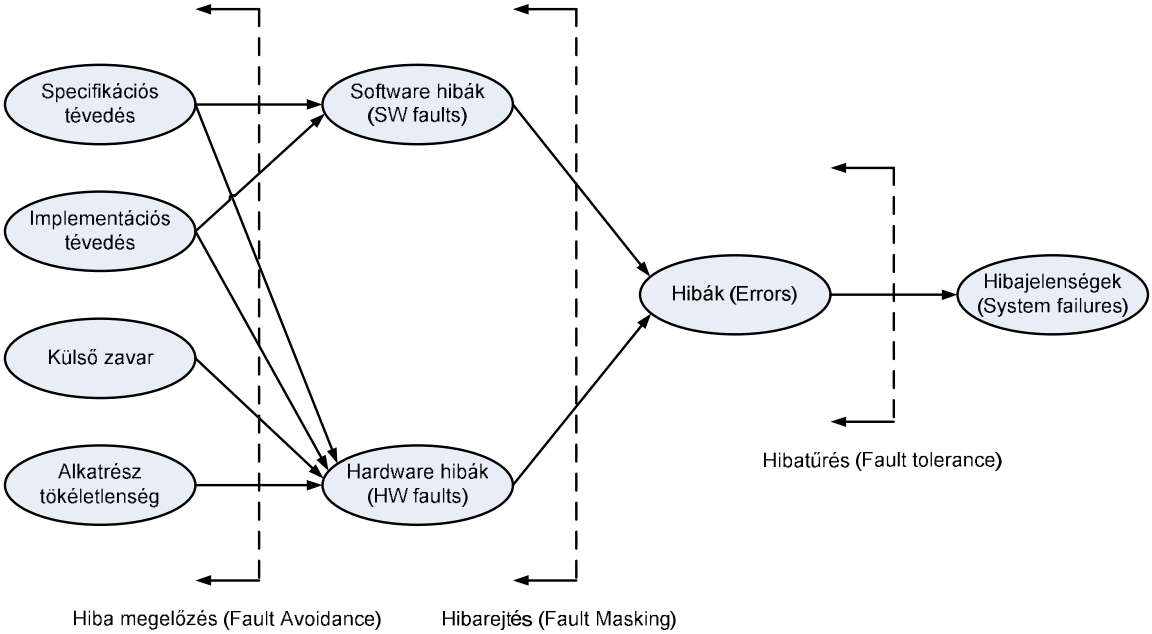
A fenti ábra öt csoportra és további alcsoportokba bontása kézenfekvő. Az jól látható, hogy rendkívül sok hibaok-kombináció képzelhető el a fenti besorolást alapul véve. Így az sem meglepő, hogy egy globális elmélettel várhatóan nem érhető el pontos előrejelzés. Ennek köszönhetően sok elméletet dolgoztak ki az egyes speciális hiba okok keletkezésének vizsgálatára.

A dolgozat kereteit meghaladja ezek tárgyalása, de az mindenképpen levonható tapasztalatként, hogy szinte minden elképzelhető és picit is reális hibára fel kell készülni, illetve a hiba lehetőségek feltárása után elemezni[17] kell az adott hiba előfordulási gyakoriságát, annak következményeit, és az elhárításra fordított erőforrásokat. [Kövesi04]

2.4. Hibaterjedés, hibák természete

Mint már eddig is szó esett róla, hogy nem a hibatűrés (fault tolerance) az egyetlen konstruktív technika, amivel a készülékek megbízhatósága növelhető. Emellett még két fő módszerről kell szót ejteni. A hiba megelőzés (fault avoidance, fault prevention) és a hibarejtésről (fault masking). A 6. ábra [Johnson89] mutatja az egyes technikák alkalmazásának területeit.

[17] Pl. ABC elemzéssel



6. ábra: Hibák terjedésének megakadályozása

A *hiba megelőzés* (fault avoidance, fault prevention) az első védelmi vonal a hibák keletkezésének kivédésében. Számos technika használható, amivel a felhasznált alkatrészek, technológiák minőségi paraméteri javíthatóak. Ennek a módszernek a jellegzetessége, hogy a rendellenességek még a kialakulásuk előtt kezelhetőek. Terv bírálat külső szakértővel, megbízhatóság növelő tervezési praktikák alkalmazása, alkatrész válogatás, túlméretezés, tesztelés, árnyékolás, egyéb minőségjavító eljárások mind-mind ebbe a kategóriába tartoznak. [Johnson89]

A *hibarejtés* (fault masking) technika a hibaok (fault) hibává "fejlődésétől"[18] óvja a rendszert, azaz amikor már bekövetkezett a "baj", annak hatásait próbálja eliminálni a rendszerből. Jellegzetes formája a fault-masking-nak a többségi szavazáson alapuló rendszerek, ahol több autonóm döntés születik és a végeredményt egy többségi szavazás adja meg. Ennél a példánál, ha az egyik résztvevő hibás végeredményt állít elő, az kiszűrhetővé válik, ehhez csak össze kell hasonlítani a többi résztvevő eredményeivel. Így a hibás eredmény nem okoz meghibásodást. [Johnson89]

A *hibatűrés* (fault tolerance) célja a meghibásodás elkerülése, ha már a hibaok hibává "fejlődött". Ennél alkalmazható a hiba maszkolás (fault-masking) és rekonfiguráció (reconfiguration). Az utóbbi módszer lényege, hogy detektálja a hibát,

[18] Nem engedi a fizikai univerzumból kilépni a hibaokot.

majd kideríti annak forrását és a hibás elemet eltávolítja a rendszerből, és esetlegesen újat állít működésbe. [Johnson89] A hiba maszkolással már az előzőekben már esett szó.

3. Hibatűrő megoldások

„Hibatűrés történelmének kezdetén, a redundancia alkalmazása mindig a fizikai hardveres megoldásokra korlátozott. A legelterjedtebb megoldás a hibatűrés megvalósítására a készüléken belüli fizikai egységeknek a többszörözése volt, de manapság már szofisztikáltabbak a megoldásaink.

A redundáns rendszer az egyszerű rendszerekhez képest a hozzáadott információ, erőforrás vagy időben dimenzióban különbözik a normál rendszer üzemelésétől. A redundanciáknak az alábbi fajtái lehetségesek:

Hardveres redundancia alatt extra hardver hozzáadását értjük, amit tipikusan hibadetektálásra vagy hibatűrés elérésére használnak.

Szoftver redundancia alatt extra szoftver hozzáadását értjük, ami azokat a kódrészek értendőek, amik az eszköz adott funkciójának elvégzésén felül tartalmaznak, hogy detektálja a hibákat és lehetőleg ki is javítsa azokat.

Információs redundancia alatt a felhasznált extra információ tartalmat értjük, ami az eszköz adott funkciójának ellátásához nem feltétlen szükséges, de a hiba detektálható és esetleg ki is javítható általuk, ilyen például a hibajavító bitek alkalmazása.

Idő redundancia alatt a rendszer működéséhez szükséges időn felüli működési időt értjük, amit hibadetektálás és hibatűrési funkciók használnak.” [Johnson89]

A hardveres redundanciára példaként hozható fel a fizikai blokkok többszörözése, a szoftveres redundanciára a várakozásokhoz rendelt timeout figyelés, az információs redundanciára a legegyszerűbb példa a paritás kód, míg az idő redundanciára az azonos számítások többszöri lefuttatása és a végeredmények koherenciájának ellenőrzése. Általában egyik redundancia sem létezik önmagában, hiszen egy timeout figyelés külön erőforrásokat igényel a processzortól, így esetlegesen szükséges lehet egy nagyobb processzor alkalmazására, ami pedig már hardveres redundanciát is igényelhet a szoftveres redundancia mellett.

Bármilyen redundanciát is használunk a költségek ilyen-olyan módon, de növekedni fognak. Ha hardveres redundanciát választunk extra alkatrészek szükségesek, a fogyasztás ezzel párhuzamosan nő, a modul méretei szintén, a járulékos mérnök

költségről nem is beszélve. Ha szoftveres redundanciát használunk, az szintén hatással van a hardveres redundanciára, hiszen több memóriára és gyorsabb processzorra van szükség, vagy a fejlesztési idő növekszik meg, és így tovább. „A fentiekből is látszik, hogy a redundancia mikéntje jelentős hatással van a rendszer teljesítményére, súlyára, energia felvételére és megbízhatóságára. Fontos megismerni az egyes módszereket, hogy kiértékeljük milyen lehetőségeket adódnak a megbízhatóság növelésére.” [Johnson89]

3.1. Hardveres redundancia

A legelterjedtebb formája a redundanciának. Ugyanis idővel az elektronikus alkatrészek egyre olcsóbbá és kisebbé válnak, így egyre elfogadhatóbb kompromisszum a hardveres redundancia alkalmazása. Ezen hardveres redundanciák három csoportba oszthatók.

A *passzív* kifejezés alatt, azokat az eljárásokat értjük, amely hiba esetén nem igényel beavatkozást az operátortól vagy a rendszer bármely részéről, hanem egyszerűen elfedi a hibát.

Az *aktív* megoldásoknál a megbízhatóság növekedését a hibadetektálás és hibaelhárítás útján érik el. Ez utóbbit gyakran rekonfigurációnak is nevezik, és rajta a hibás hardver eltávolítása, vagy adaptív módszerek alkalmazása értendő. Ez utóbbi nem más, mint a rendszer megpróbál a megváltozott körülményekhez alkalmazkodni, úgy, hogy olyan irányba változtatja a belső működését, hogy fent tartsa a rendszerműködést.

A *hibrid* módszer az előző kettő ötvözete, ami ötvözi a két rendszer előnyeit. Így a hiba elfedés (error masking), a hibadetektálás (error detection), a hiba lokalizáció (error location) és a hiba helyrehozás (error recovery) eszközeit közösen használja a hibrid rendszer a hibatűrési tulajdonságok (lásd 1 fejezet) javítására.

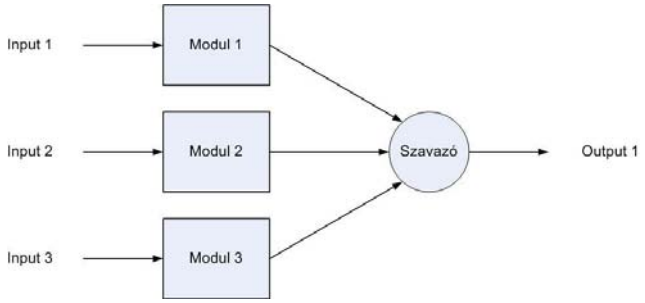
3.1.1. Passzív redundancia

3.1.1.1. Triple module redundancy (TMR)

A modulok háromszorozása, az előforduló hibák maszkolásán, elfedésén alapul. A legtöbb passzív redundancia fontos eleme a többségi szavazás (majority voting).

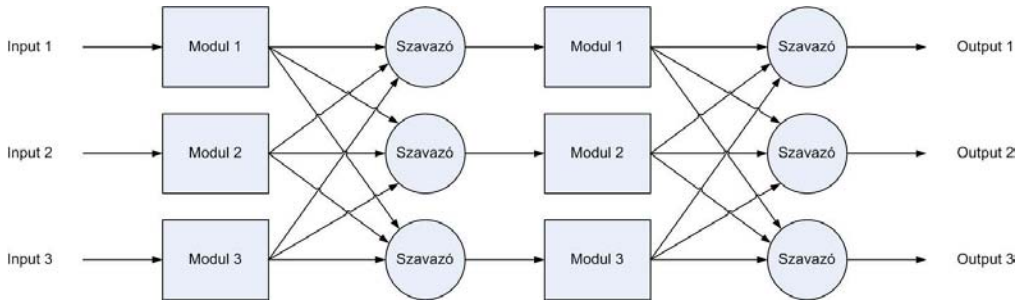
A megbízhatóság növekedését a működéshez szükséges egy helyett egyszerre három modul használatával érhető el. Lásd a 7. ábra. Majd ezen modulok eredményeinek összehasonlításával a többségi szavazón biztosítható a hibátlan működés mindaddig, míg

a három modulból kettő hibátlanul működik. Vagyis a módszer egy egység meghibásodását tudja csak tolerálni, amint egynél több modul hibásodik meg a működése kiszámíthatatlan lesz. Ez az eljárás természetesen szoftver esetén is használható. Ott általában három különböző szoftver példány fut, aminek az eredményeit szavaztatja a rendszer.



7. ábra: TMR

A rendszer gyenge pontja[19] a szavazó, hiszen amennyiben az nem működik helyesen, úgy az egész rendszer működése ellehetetlenül, hiába sokszoroztuk meg a modulokat. Vagyis a TMR megbízhatósága nem lehet jobb a szavazó megbízhatóságánál. Mivel a szavazó szoftveres és hardveres esetben is egyszerű, így meglehetősen nagy megbízhatósággal bír a többi rendszerhez képest, de amennyiben ez mégsem lenne elegendő a rendszer megbízhatósága szempontjából, a szavazókat is háromszorozni lehet. Ahogy ez a 8. ábra is szerepel. [Johnson89]



8. ábra: TMR szekvenciális felbontással

Ahogy ez a fenti ábrán is látható, a funkció szekvenciális lépéssorozattá alakításával és az egyes szintek közé szavazó iktatásával a TMR megbízhatósága jelentősen nő, hiszen a hibaok (fault) keletkezési helyéhez közel sikerült a hibát (error) megállítani, így az nem terjed ki a rendszer többi részére. A hiba az adott szinten

[19] Szakirodalomban: „single point of failure”

kiküszöbölődik, és nem okozza az egész funkció sorozat kiesését. Ennek köszönhetően egyszerre több hibát is tolerálni tud a rendszer, feltéve, ha azok különböző szinteken fordulnak elő.

3.1.1.2. N-modular redundancy (NMR)

Az NMR a triplikálás általános esete, véges számúszor többszörözött egységekre alkalmazva. A megoldás ugyanúgy a többségi döntés elvén alapul, mint amikor három rendszer működik párhuzamosan. Itt a rendszerek száma (N) háromnál nagyobb is lehet, azonban célszerű páratlanra megválasztani, mert ellenkező esetben szavazategyenlőség léphet fel. A gyakorlatban még az 5MR és 7MR rendszert alkalmazzák. Az 5MR és a 7MR alkalmazásának előnyei a TMR-hez képes a nagyobb megbízhatóságban van, hiszen ez nem egy, hanem két, illetve három meghibásodást képes kezelni. Ugyanakkor N értékét nem célszerű túl nagyra sem választani, mert azzal drasztikusan nőnek a költségek, fogyasztás, súly és a hely igények is. [Johnson89]

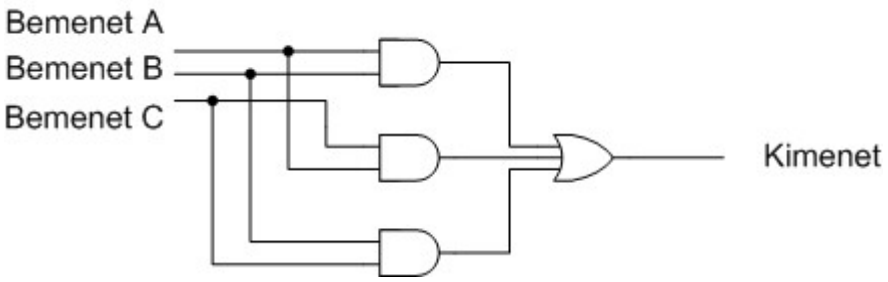
3.1.1.3. Többségi döntők

A többségi szavazók az NMR rendszerek kulcskérdései, sok esetben kritikus pontjai a megbízhatóságnak, ugyanis sok esetben[20] nincs lehetőség többszörözött szavazókat alkalmazni a kimeneteken, hiszen általában egy azonos kimeneti jelcsoportra van szükség, nem pedig több azonos csoportra. Ilyenkor a szavazó a rendszer kritikus pontjává válik, ahol egyszeri hiba is a meghibásodását vonhatja maga után. Ekkor az elérhető legjobb megbízhatóságot kell garantálni a szavazónak. Ez nem egyszerű feladat, de erre mutatnak be példákat a következők.

Az *egyszerű szavazás* a legkézenfekvőbb szavazó áramkör, mindössze négy kapuból kialakítható, mint az látható a 9. ábra. A szavazónak többségi döntést kell hoznia, azaz három résztvevő esetén a kettő vagy több “szavazattal” rendelkező eredmény mellett kell döntenie. Ez a mechanizmus olvasható ki a lenti áramkör működéséből is, ahol bármelyik két bemenet magas szintje esetén magas lesz a kimenet és bármely két bemenet alacsony szintje esetén, a harmadik bemenettől függetlenül, alacsony kimeneti szint jelenik meg.

---

[20] Még, olyan nagy megbízhatóságú rendszereknél, mint a vadászgépek, többségében csak egy aktuátor van. [Johnson89]



9. ábra: Kapus szavazó

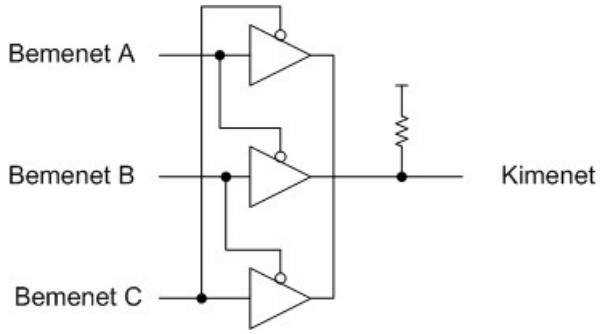
A fenti ábrán az egybites szavazó rajza látható, de azonos, párhuzamosan elhelyezett áramkörökkel a szavazó adott szószélességre bővíthető. A szavazó szempontjából nem indifferens, hogy a bemenetén mikor érkeznek meg a jelek, ugyanis a bemeneti jelek elcsúszása rossz kimeneti eredményt okozhat. Ez kiküszöbölhető a bemeneten és a kimeneten egyaránt elhelyezett master-slave elrendezésű flip-flop-kal. [Johnson89]

Mint az megtehető a kapus szavazóval, megtehető a többféle egy bites szavazóval is. Ezek az egy bites szavazók ugyanazt a logikai kapcsolatot valósítják meg, mint az előbb bemutatott digitális kapus áramkör. Néha célszerűbb más szavazási eljárást használni. Például magas háttér sugárzás esetén egyszerű *kapcsolóelemekből* (tranzisztor, MOSFET) felépülő áramkör célra vezető lehet, ahol az egyes kapcsoló elemek a sugárzással szemben jobban ellenálló technológiával[21] készülnek, de célszerű lehet ennek a kapcsolásnak az alkalmazása optocsatolóval leválasztott kimenetek esetén is, hiszen ott adódik a lehetőség a kapcsoló tranzisztorok használatára, ezen keresztül pedig az áramkör egyszerűsítésére, ami általában pozitívan befolyásolja a megbízhatóságot.

A *buffer-es* szavazó áramkör fő alkalmazási területe az SRAM alapú FPGA-ban [Xilinx06] van, ahol a kozmikus sugárzás miatt bekövetkező single event upset (SEU) hatására az SRAM tartalma átíródhat, így az áramkör működése megváltozik. Emiatt nehezen készíthető megbízható szavazó, hiszen a szavazó is ugyanolyan sugárzásnak van kitéve – így az is ugyan olyan megbízhatatlan –, mint a többi áramkör. Az FPGA-ban fellelhető meghajtók a 10. ábra szerinti összekapcsolásával azonban létrehozható egy relatíve megbízható egybites megoldás, hiszen ez kevés SRAM cellát használ az

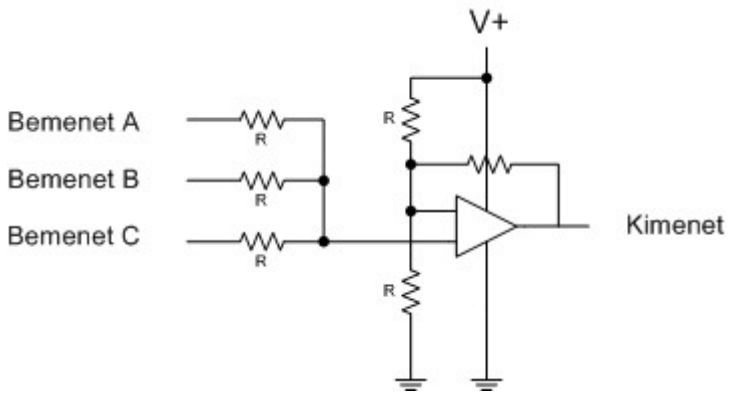
[21] Szakirodalomban RadHard, azaz „Radiation Hardened” eljárásnak nevezik.

összekapcsolási kényszerek tárolására, így kisebb a lehetősége egy SEU-nek. [Samudrala04]



10. ábra: Buffer-es szavazó

A *műveleti erősítős* (threshold gate) egy másik lehetőség a többségi szavazás áramköri megvalósítására. A 11. ábra látható az áramkör, amely egy tápfeszültségről működik, hiszterézises komparátor üzemmódban. Egyik bemenetén a tápfeszültség 1/3 és 2/3 értéke közé van beállítva, míg a másik bemenetén lévő feszültségosztó határozza meg a komparált feszültség szintet. Ugyanis, attól függően, hogy az ellenállások plusz tápra (logikai egyes) vagy földre kapcsolódnak (logikai nulla) változik a bemeneten a feszültség, vagyis a szavazás végeredménye.



11. ábra: Műveleti erősítős

Ha mindegyik bemenet ’1’ akkor a feszültség megegyezik a tápfeszültség értékével, ha az egyik ellenálláson ’0’ van, de a többin ’1’ marad, akkor a feszültség a tápfeszültség 2/3-ával egyezik meg, míg ha két ’0’ és egy ’1’ szintű bemeneti jel van, a kimeneti feszültség 1/3-ra esik vissza, amikor minden bemenet ’0’, akkor értelemszerűen nulla lesz az osztó feszültsége is. Így egy többségi szavazóként működhet az áramkör, ha a referencia feszültség 1/3 és 2/3 közé van beállítva. A feszültségosztót egyaránt

meghajthatja digitális kapu, valamint analóg áramköri elem is, csak az a lényeg, hogy meghajtóként üzemeljen mindkét feszültségnél (föld és táp).

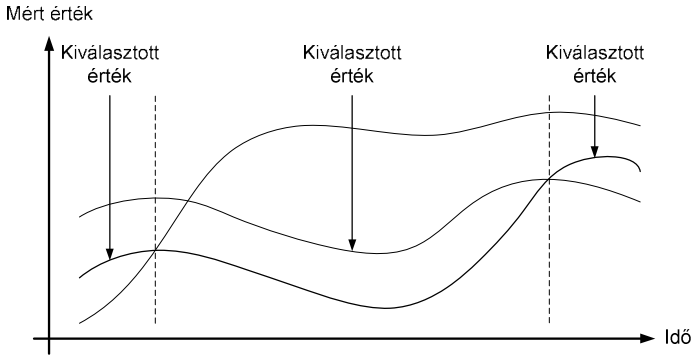
Az áramkör egyszerűsége folytán könnyű alkalmazni, azonban mégsem elterjedt. Ennek oka, hogy a műveleti erősítők alkalmazása nem célszerű digitális áramkörökben, mert más célra fejlesztik őket, így lassúsak és gerjedhetnek is. Olyan esetekben azonban, ahol nincs digitális áramkör, de több érzékelő jeléről kell szavazással dönteni, ott jól alkalmazható ez a megoldás is.

Természetesen a szavazást nem csak hardveresen, hanem *szoftveres szavazással* is meg lehet megvalósítani. Ebben az esetben biztosítani kell az egyes résztvevők kommunikációját, hogy a szavazandó értékeket meg tudják osztani egymással. Ez a megosztás lehetséges több portos memóriák alkalmazásával, pont-pont, pont-multipont vagy multipont-multipont kommunikációval, de ügyelni kell arra, hogy a kommunikáció is többszörözött legyen, azaz egy közös kommunikációs busz alkalmazása a legtöbb esetben nem elegendő, hiszen az single point of failure-t okozhat.

A szoftveres megoldás előnye a hardvereshez képest a rugalmassága, kisebb alkatrész igénye (gondoljuk csak egy 32 bit szóhosszúságú szavazóra), ami kisebb fogyasztást és költséget eredményez. Ugyan az algoritmus csak kis számítási kapacitást igényel, de a dedikált hardvereshez képest lassabb, és emellett a független rendszerek (nincs közös órajel) esetén szinkronizációs problémákat is meg kell oldani, ami pedig megbízhatóság szempontjából nem feltétlen előnyös.

Ugyanakkor sokszor (például szenzorok kimenete) esetén előfordul, hogy helyesek az értékek, de nem egyeznek meg teljesen, hanem a pontosságukon belül eltérnek egymástól. Ilyenkor egy normális szavazó hibát jelezne, de a szoftveres megoldással könnyen megoldható, hogy a helyes értékeknek, adott Δ tartományon belül kell elhelyezkedniük egymáshoz, azaz meg van engedve egy Δ eltérés, amin belüli eltérések még nem számít hibának. Ez megoldja a problémát, de több párhuzamos szavazó esetén biztosítani kell, hogy a mindegyik szavazó kimenet teljesen (bitre pontosan) azonos legyen, különben ez a későbbiekben nagy problémát jelenthet. [Johnson89]

Ha a Δ tűrést kettő hatványai szerint választjuk meg, akkor ez a módszer hardveres szavazó esetén is alkalmazható az LSB bitek elhagyásával elvégzett összehasonlításra.

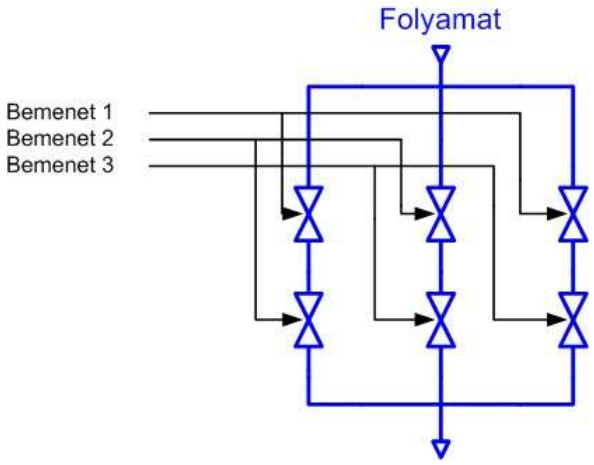


12. ábra: Középérték szavazó, kiválasztási eljárás

Az előbbi problémára ad, egy kicsit más megoldást a *középérték szavazó*, ami több, akár egymástól nagy mértékben eltérő bemenet esetén is a legjobb megoldást adja. Ugyanis, mint azt a 12 ábra is mutatja, a kimenetére az értékben középen elhelyezkedő jelet választja. Amennyiben a három jelből kettő helyes, akkor könnyen belátható, hogy mindig a helyes jelet választja. Az elv minden páratlan számú bemenetű szavazó esetén is alkalmazható. [Johnson89]

Bizonyos esetekben van értelme a kimeneti értéket a bemeneti értékek függvényeként meghatározni. Ha például nem a középső értéket választjuk ki, mint középérték szavazó esetében, hanem a bemenő jelek alapján számítjuk ki a kimeneti értéket. Lásd még [Parhami94], [Latif-Shabgahi04], [Krstic05]

Ha csak egy beavatkozót használunk, nem tudjuk a szavazókat többszörözni, akkor kritikus pontokat (szavazó, beavatkozó) viszünk be a rendszerbe. Éppen ezért ha lehetséges, és a folyamat jellege megengedi, több (2-3) beavatkozót és hozzá megfelelő számú szavazót kell alkalmazni. Amikor több beavatkozó használatára is lehetőség nyílik, jobb megoldás, ha a szavazó áramköröket elhagyva, több beavatkozó olyan összekapcsolt rendszerét használjuk, amely a folyamatirányításon kívül elvégzi a szavazás feladatát is. Ezt nevezik *technológiai szavazásnak.* Erre mutat példát a 13. ábra, ahol hat szelep soros párhuzamos kapcsolatával – csak úgy, mint a tranzisztoros szavazónál – történik a szavazás. Ugyanis a technológiai szavazás nem csak a szavazó áramkör hiányából adódó megbízhatóság növekedésének előnyeivel rendelkezik, de a beavatkozók többszörözése következtében, az azokban bekövetkező hibák kezelésére is módot ad.



13. ábra: Technológiai szavazás

Az előzőekhez hasonlóan képzeljünk el egy motort, melynek a tekercselése három részre van osztva. Ehhez kialakítható olyan vezérlés, amely visszacsatolt jelen keresztül változtatni tudja a kimeneti jelének nagyságát. Egy közös armatúrán elhelyezve a tekercseket, azok fluxusai összegződnek, vagyis egy olyan összegzőt kapunk, aminek három bemenetén akkora jelerősség érkezhet, hogy egy bemeneti jel önmagában is képes előállítani a megfelelő kimeneti szintet. Ami hibátlan bemenet esetén (1/3+1/3+1/3=1) és egy hiba esetén is működőképes (1+1-1=1). Ezt nevezik *fluxus összegzőnek* (flux-suming).

Ez az elv nem szavazás ugyan, de azzal megegyező eredménye van. Ezt a módszert nem csak motorok esetén, hanem egyéb áramkörök esetén is alkalmazzák. Csupán egy megfelelő transzformátor és meghajtó áramkört szükséges, amit ugyan a kimeneti jel visszacsatolása bonyolít kissé. Ugyanakkor a fluxus összegző egyszerű (nagy megbízhatóság) és robosztus (külső zavarokkal szembeni rendkívüli ellenálló képesség), kivitele miatt előnyös. [Johnson89]

A *szó hosszúságú szavazás* (word-voter) az eddigiektől eltérően nem bitenként dönt az eredményről, hiszen a valóságban több bitből álló szók az egyes modulok kimenetei, amik nem függetlenek egymástól. Így célszerű azokról az egész szó összehasonlításával dönteni. Ennek belátására nézzük a következő példát. Legyen a három bemeneti kombináció ’01’, ’11’ és a hibamentes harmadik ’10’, de a bitenkénti szavazás eredménye nem jelez hibát, és az eredmény ’11’ lesz, ami pedig nem helyes eredmény, de egy egész szóhosszúságot figyelő szavazóval a hiba felismerhető lett volna.

Vannak olyan esetek (például kerekítési hibák), amikor a kimeneti értékek nem egyeznek meg teljesen. Ilyenkor a hibákat számolni célszerű, vagy a kimenet csak egy részére alkalmazni a szavas szavazót, míg a fennmaradó részen továbbra is bites szavazó marad. [Mitra&McCluskey00]

Ugyanakkor a szavas szavazó miden fajta előnye mellett, hátránya a bites szavazókhoz képesti bonyolultságában rejlik. Ezt az áramkört végképp nem célszerű diszkrét elemekből felépíteni, de az FPGA-k területen van létjogosultsága a módszernek.

3.1.2. Aktív redundancia

Alapvetően más úton jár az aktív redundancia, mint a passzív. Hiszen itt nem a hiba elrejtése (fault masking) a cél, hanem éppen ellenkezőleg, fel szeretnénk tárni (error detection) azt. Tisztában szeretnénk lenni vele, hogy melyik modulban keletkezett (error location) és ha van rá lehetőség, a rendszer struktúráját olyan irányba szeretnénk módosítani, hogy az negligálja (error recovery, reconfiguration) a hiba jelenlétét.

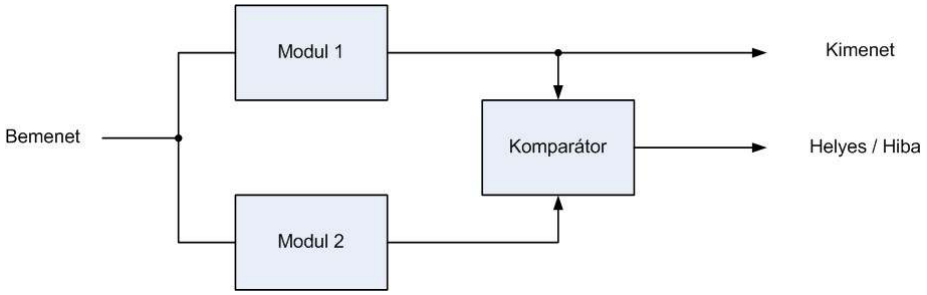
3.1.2.1. Duplikáció összehasonlítással

Ennek egyik legegyszerűbb módszere, amikor azonos tevékenységet kétszer[22] hajtunk végre. Ilyenkor lehetőség nyílik az eredmény összehasonlításon alapuló hibadetektálásra. Vagyis a két végeredmény különbsége esetén tudjuk, hogy legalább egy egység hibázott. A TMR-rel ellentétben, azt viszont nem tudjuk megmondani, hogy melyik egységnek van igaza, hiszen csak két vélemény áll szemben egymással, melyek közül egyik sem megbízhatóbb a másikénál. Ennek a módszernek pont ez a lényege, azaz hogy derüljön ki az eredmény helyes vagy helytelen volta (Hihetőség (Creditability)). Azt ugyanakkor fontos hangsúlyozni, hogy sem itt, sem az eddig megismert TMR rendszereknél, a felsorolt módszerek nem védenek azon hibák (common mode failure) ellen, amelyeket "beleterveztünk"[23] a redundáns modulokba.

---

[22] Hardveres redundancia esetén ez párhuzamos végrehajtást jelent, idő redundancia esetén ez egymás utáni végrehajtást.

[23] A megvalósult rendszer specifikációja tartalmazza.

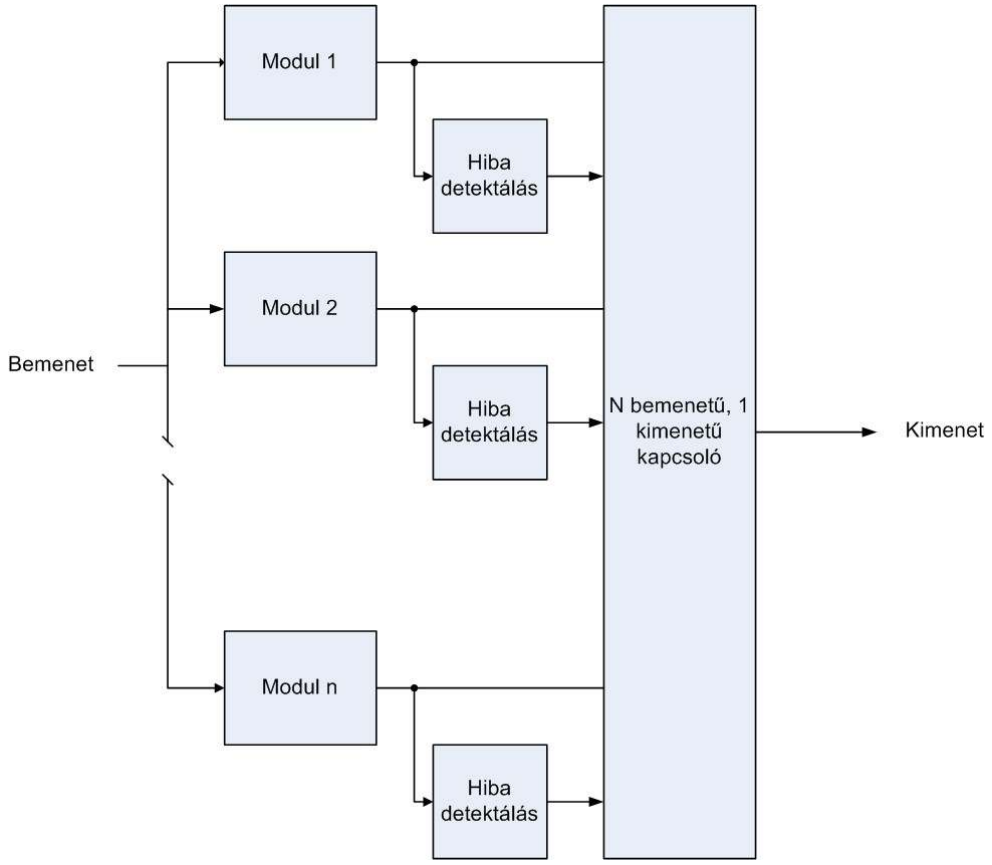


14. ábra: Duplikáció összehasonlítással

A 14. ábra látható egy általános, duplikált egység blokkvázlata, ahol a komparátor kimenete jelzi, hogy az eredmény helyes vagy sem. Ebből már látszik is a struktúra nagy hibája, vagyis hogy a TMR szavazójához hasonlóan, a megbízhatóság kritikus eleme az összehasonlító egység. Szerencsére, az egybites összehasonlító egy darab XOR kapuból áll, ami természetesen korlátlanul bővíthető és így biztonság szempontjából kevéssé kritikus egység van szó, csak úgy mint bites szavazók esetén (3.1.1.3 fejezet). Ugyanakkor ennél a módszernél is hasonló problémák (nem teljesen azonos kimenet, késleltetési kérdések) jelentkezhetnek, mint a korábban megismert szavazóknál, de a megoldások (pl.: szoftveres szavazó) is hasonlóak lehetnek.

3.1.2.2. Tartalékos rendszerek

A másik aktív redundáns megoldás a tartalékos rendszerek, melynek általános rajza 15. ábra látható. Különböző hibadetektálási (error detection) elvek mentén hibásnak (error location) beazonosított modulok cserélhetőek le (error recovery) tartalék egységekre.



15. ábra: Tartalékos rendszerek

Koncepcionálisan a lecserélés egy kapcsoló útján képzelhető el. A hibadetektálás sok módja ismert, ezek közül párat a következő fejezetben mutatok majd be. Amennyiben mindegyik modul hibátlannak bizonyult, prioritási elven hozható döntés a kimenetre kapcsolt egységről, ellenkező esetben a hibásnak bizonyult modul ideiglenesen, vagy véglegesen kizárásra kerül a rendszerből. Hiba esetén a tartalékos rendszerek vissza tudnak térni (rekonfiguráció) a tökéletesen üzemelő állapotba, de az "átkapcsolás" időbe telik, amely idő alatt a funkció részben vagy egészben kieshet.

A tartalék egységek két fő típusa létezik, a hideg és a meleg tartalék. A különbség a kettő között, hogy a modul energizált állapotban (meleg tartalék) vagy kikapcsolt állapotban (hideg tartalék) várja, hogy a kapcsoló kiválassza. A meleg tartalék gyakorlatilag szinkron üzemel a kiválasztott modullal, éppen ezért gyorsan át is veszi annak szerepét, de a folyamatos üzem miatt többet fogyaszt és esetlegesen gyorsabban meg is hibásodhat. Míg a hideg tartalék kevesebbet fogyaszt, és a kikapcsolt eszköz megbízhatósága sem romlik olyan mértékben az idővel, mint bekapcsolt állapotában, de a hibás modul kiesése utáni elindulás sok időt vesz igénybe. Mindegyik tartalékolás esetén

van a működésben egy kieső idő, ami bizonyos rendszerekben nem engedhető meg, ilyen egy vegyi üzem folyamatait felügyelő rendszer számítógépe. Egy telekommunikációs műhold esetében néhány másodpercnyi kiesés még megengedhető, a fogyasztás terén tett engedményekért és az összességben a nagyobb megbízhatóságért cserében.

Nagy előnye a tartalékos rendszereknek, hogy 'n' számú azonos funkciójú modul esetén is közel 'n' hibát tud tolerálni. Csak összehasonlításként egy 7MR rendszerrel, ahol n=7 és csak k=3 egység meghibásodást tudja csak tolerálni. Az egységek redundanciáját százalékban kifejezve (k/n)*100% módon lehet. Ez az érték a tartalékos rendszerek esetén közel 100% (ha 'n' végtelenbe tart), az NMR rendszereknél ez maximálisan 50% (ha 'n' végtelenbe tart) körüli teljesítményéhez hasonlítva, nem is beszélve arról, hogy ez nem jár iszonyatos fogyasztás többlettel. Ugyanakkor persze az újrakonfigurálás ideje alatt a tartalékos rendszer esetleg nem rendelkezik teljes funkcionalitással.

A *hibadetektálás eszközeiről* eddig nem esett szó, de a tartalékos rendszereknél, ez kulcskérdés, hiszen ha ez az algoritmus vagy eljárás nem működik helyesen, akkor az egész rendszer működésképtelen.

Az egyik legegyszerűbb ilyen módszer, ha egy operátor érzékeli a rendszer meghibásodását, és elindítja a tartalék eszközt. Ez persze, csak abban az esetben megoldható, ha a rendszer környezete megengedi az operátori beavatkozás lassúságát. Ha a beavatkozás nem igényel jelenlétet, az akár távolról is megtehető, ahogy azt számos műhold esetén is teszik. Az átkapcsoló elektronika egy váltókapcsolótól, egy harci eszköz redundáns átkapcsoló áramkörének a bonyolultságáig terjedhet. Az operátori döntést, bonyolultabb esetekben diagnosztikai eredményekkel célszerű segíteni, ugyanis segítség nélkül, az operátor legtöbbször már nem tud hibadiagnosztikát végezni (error detection, error location), csak a meghibásodást (failure) tárhatja fel, azaz már csak a meghibásodás bekövetkezte után tud dönteni.

Az öndiagnosztika egyik legegyszerűbb esete a watchdog időzítő. Nem igényel jelentős hardver ráfordítást, és a modern processzorok integráltan tartalmazzák is. A watchdog nagy előnye, hogy nem csak a hardveres hibák, de a szoftveres hibák ellen is sikeresen alkalmazható.

A watchdog időzítő rendszerint sokkal egyszerűbb annál az egységnél, amit monitoroz és éppen ezért sokkal megbízhatóbb is. Alap esetben egy egyszerű lefelé

számlálóról van szó, amely elkezdi a számlálást egy előre beállított értékről, és ha az eszköz, amit monitoroz nem állítja vissza időnként a számláló érétkét újra és újra, akkor előbb-utóbb[24] az eléri a nulla értéket, és beállít egy kivétel flag-t, vagy egyszerűen újraindítja a megfigyelt egységet, ezzel detektálva annak abnormális működését.

A már többször említett timeout is hasonlóképpen működik, amikor egy esemény bekövetkezése normál esetben jól definiált időn belül meg kell történjen, erre az időre beállított számlálót elindítva, ha az esemény nem következik be, hibát jelez a számláló.

„A watchdog számláló nem detektálja az összes problémát, csak megerősíti az egység ”életképességét”. Éppen ezért további megoldásokkal kombinálva alkalmazzák.” [Parhami06]

A watchdog alkalmazása során fontos, a számlálót alaphelyzetbe állító utasítás átgondolt elhelyezése. Célszerűen a főciklusban kell elhelyezni. Valamint a számláló periódus idejét is mindig az adott alkalmazás tulajdonságaihoz (reagálási idő, taszkok futási ideje) kell igazítani.

Létezik egy busz forgalmat figyelő watchdog struktúra is, ahol a végrehajtás jellegzetességeit megfigyeli a control-flow watchdog, a fordító által csak erre célra előkészített program jellegzetességek[25] alapján. [Parhami06]

Ilyen egyszerű hibadetektálásra használható egy bemeneti érték adott tartományon belül tartózkodásának a megkövetelése is. Például egy szenzor kimeneti értékét mérve, a szenzor fizikai (adatlapján feltűntetett) határain kívül eső érték adódik, ilyenkor egyértelmű a szenzor vagy a mérőáramkör meghibásodása. De amennyiben további hihetőségi tartományok (általában szűkebb és az előbbi tartományon belül esik) is felállíthatóak egy rendszernél, ahol a vezérelt folyamat adottságaiból adódóan sohasem vehet fel értéket. Így nem csak a szenzor alapvető elsődleges meghibásodására lehet következtetést levonni, hanem további hibák is felderíthetők. Ezekben a hihetőség vizsgálatoknál nem csak a értéktartománnyal lehet operálni, hanem egyéb folyamat jellemző tulajdonságokkal, mint például a jel változási sebessége.

Öntesztelő algoritmusokat is széles körben használják hibadetektálásra. Az algoritmusok nehézségük éppen abban áll, hogy egy hibásan működő rendszernek kell

---

[24] Általában a megfigyelt eszköztől független, önálló órajellel rendelkezik, így végső soron egy visszaszámláló időzítőt valósít meg.

[25] Érvényes memória címek, elágazások, függvényhívások

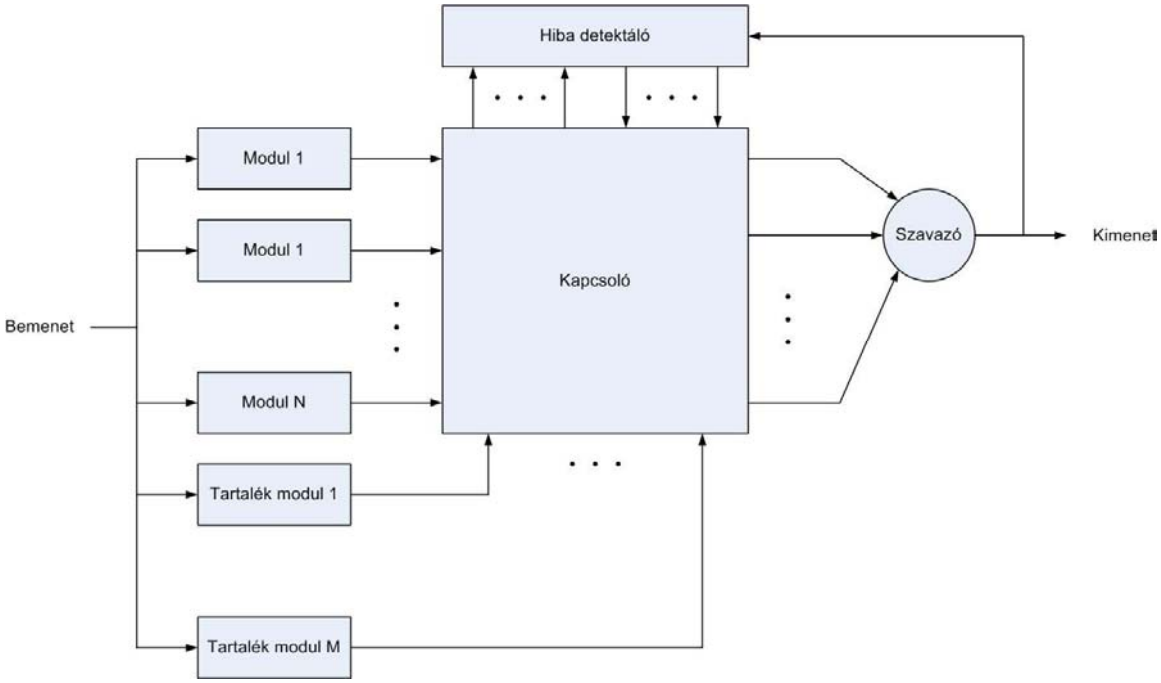
kiderítenie magáról a működő képtelenségét. De ilyen esetben is elképzelhető az elsődlegesen kiszámított értékek közelítő becslése alapján a működőképességet meghatározni. Ezen megoldások mindig az alkalmazás specifikus implementációtól függenek.

3.1.3. Hibrid redundancia

A hibrid megoldások ötvözik az aktív és passzív megoldások előnyeit, így a legjobb tulajdonságokkal rendelkező rendszerek kaphatóak alkalmazásukkal, ugyanakkor persze ezek a leg erőforrás igényesebbek is.

3.1.3.1. Tartalékolt NMR

A tartalékolt NMR rendszerek kis overhead-jük ellenére nagy hátrányuk, hogy működésük nem folyamatos. Hiba esetén ugyanis a rekonfiguráció (error recovery) a tartalék típusától (meleg vagy hideg) függően, de igényel átállási időt és a hibadetektálás is relatíve bonyolult és kevésbé robosztus, mint az NMR rendszerek esetén a szavazás. Egy NMR, és egy tartalékolt rendszerrel ötvözni lehet a két rendszer előnyeit. Vagyis a tartalékolt rendszereknél megszokott kis mértékű redundanciát, és hiba esetén folytonos működést biztosító struktúrát kapunk.



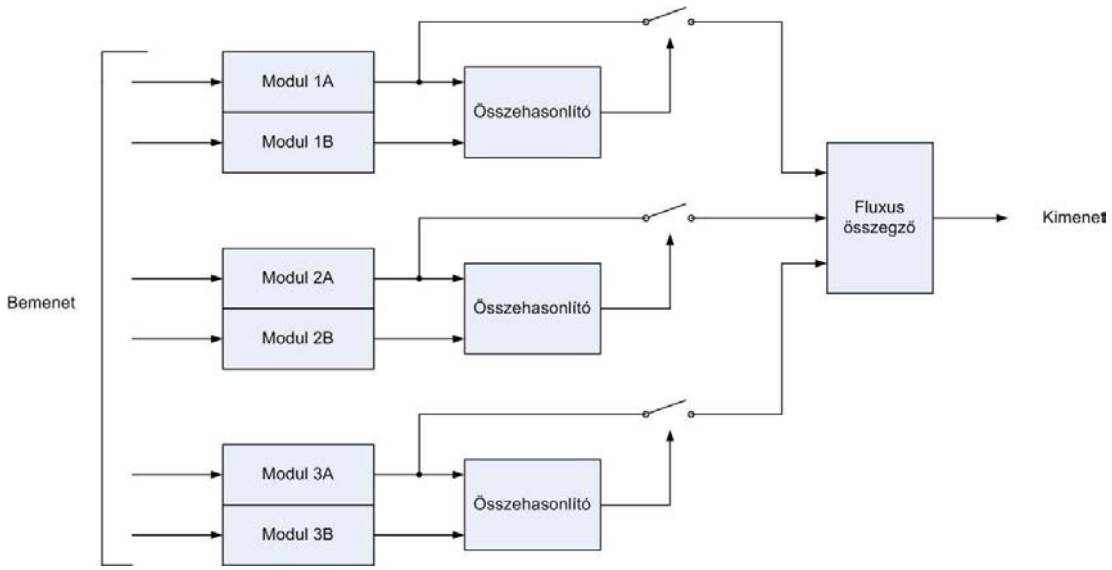
16. ábra: Tartalékolt NMR

A 16. ábra látható a rendszer általános blokk vázlata. Az első ’N’ darab modul kimeneti jelei kapcsolódnak a szavazóra, a többi modul tartalékként van jelen a rendszerben. A hibás modul jelenlétét a rendszerben a megszavazott, és az alapmodulokból érkező jelek összehasonlítása jelzi. A hibás modul egy tartalék egységre cserélhető le mindaddig, amíg van még működőképes tartalék a rendszerben. A fenti elrendezés hét modul és háromszorozott redundancia, N=3 (TMR) esetén k=5 modul meghibásodását tudja tolerálni, azaz k/n*100% = 71%.

Itt említeném meg, hogy léteznek a fentihez meglehetősen hasonló elrendezések (self-purging redundancy, sift-out modul redundancy), de ezek tárgyalása meghaladja a dolgozat kereteit, további információ a [Johnson89] található.

3.1.3.2. Triplex-duplex

A 17. ábra látható triplex-duplex elrendezés. Ez szintén egy “öszvér” megoldás, amely több duplex – ami fail-silent, azaz hiba esetén elhallgat – rendszer TMR-hez hasonló rendszerbe szervezése révén jön létre. Azért nem teljesen TMR az elv, mert a szavazó flux-summing elven működik. Így egy duplex pár együtt nem értése esetén az elhallgat, de a maradék két pár TMR üzemmódból duplex üzemmódba vált, ahogyan egy normális szavazó is tenné. Azonban, ha még egy páros tagjai közti eltérés miatt azok is lekapcsolják magukat, akkor csupán egy duplex pár marad, amely egyszerű duplex rendszerként végzi tovább a feladatát. Ilyen módon használja ki ez a módszer a rendelkezésre álló modulokat a hosszantartó működés zavartalansága érdekében.



17. ábra: Triplex-duplex elrendezés

Köszönetnyilvánítás

Szeretném megköszönni tanáraim segítségét és önzetlen munkáját, hogy megszerették velem – e szakmát és sok ismeretet gyűjthettem általuk. Kiemelném a diplomamunkám elkészítésben külön is segítséget nyújtó tanáraimat: Benesóczky Zoltán, Görgényi András, Hermann Imre, Rácz Gábor, Selényi Endre, Zoltán István.

Valamint köszönöm a családom szüntelen támogatását, nélkülük a tudomány eme fokát nem hódíthattam volna meg.

A munkám néhai édesapámnak ajánlom, aki legalább annyira szerette a szakmáját, mint fia.

Irodalomjegyzék

[Caldwell98] Douglas Wyche Caldwell, „*Minimalist Fault-Tolerance Techniques for Mitigating Single-Event Effects in Non-Radiation-Hardened Microcontrollers*”, UCLA, 1998., http://www.cs.ucla.edu/~rennels/dougdiss.pdf

[Johnson89] Barry W. Johson, „*Design and Analysis of Fault-Tolerant Digital System*”, 1989., Addison-Wesley Publishing

[Kövesi04] Dr. Kövesi János, „*Kockázat és megbízhatóság*”, egyetemi jegyzet (BME-MVT), 2004.

[Krstic05] M.D. Krstic, M.K. Stojcev, G. Lj. Djordjevic, I.D. Andrejic, „*A mid-value select voter*”, Microelectronics Reliability 45 (2005.), pp. 733–738

[Latif-Shabgahi04] G.R. Latif-Shabgahi „*A novel algorithm for weighted average voting used in fault tolerant computing systems*”, Microprocessors and Microsystems 28 (2004.), pp. 357–361

[Marx00] Marx György, „*A marslakók érkezése*”, Akadémiai kiadó, 2000

[MISRA] MISRA Consortium, http://www.misra-c2.com/

[Mitra&McCluskey00] Subhasish Mitra, Edward J. McCluskey, „*WORD VOTER: A New Voter Design for Triple Modular Redundant Systems*" VLSI Test Symposium, pp. 465-470, 2000.

[Nelson&Carol87] Victor P. Nelson, Bill D. Caroll „*Tutorial: Fault-Tolerant Computing*” IEEE Computer Society Press, Washington DC. 1987.

[Neumann56] J. von Neumann „*Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components*” Automata Studies, C. E: Shannon and J. McCarthy, eds. Princton University Press 1956., pp. 43-98

[Parhami94] Behrooz Parhami „*Voting Algorithms*”, IEEE Transactions on Reliability, Vol. 43., No. 4., 1994. December

[Parhami06] Behrooz Parhami, „*Hardware Design Methodes*”, Fault-Tolerant Computing UCSB egyetemi jegyzet, 2006., http://www.ece.ucsb.edu/Faculty/Parhami/pres_folder/f33-ft-computing-lec13-hardw.pdf

[Rennels84] David A. Rennels „*Fault-tolerant Computing – Concepts and Examples*”, IEEE Transactions on Computers, December 1984., pp. 1116-1129

[Rennels99] David A. Rennels, „*Fault-Tolerant Computing*”, Encyclopedia of Computer Science, ed., Anthony Ralston, Edwin Reilly, and

David Hemmendinger, 1999. International Thomson Publishing - Europe, http://www.cs.ucla.edu/~rennels/article98.pdf

[Samudrala04] Praveen Kumar Samudrala, Jeremy Ramos, and Srinivas Katkoori, „*Selective Triple Modular Redundancy (STMR) Based Single-Event Upset (SEU) Tolerant Synthesis for FPGAs*", IEEE Transactions on Nuclear Science, Volume: 51, October 2004.

[Selényi01] Dr. Selény Endre, *„Megbízhatóságelmélet alapjai"*, BME-MIT, 2001., http://home.mit.bme.hu/~selenyi/ftp/szganalizis/megbanal1.doc

[Serlin84] Omri Serlin, „*Fault-Tolerant Systems in Commercial Applications*", IEEE Computer, August 1984., pp. 19-30

[Szalai94] Balázs András, Hernyes István, Horváth István, Pálos István, Spányi Péter, dr. Szalai Sándor, Várhalmi László, „*Megbízhatóságot növelő hardver és szoftver megoldások*", XXX. Ipari Elektronikus Mérés és Szabályozás Szimpózium, 1994. szeptember

[Xilinx06] Xilinx Corporation Application Notes, *„Triple Module Redundancy Design Techniques for Virtex FPGAs"*, 2006., http://www.xilinx.com/bvdocs/appnotes/xapp197.pdf

[Yeh96] Y. C. Yeh, *„Triple-Triple Redundant 777 Primary Flight Computer"*, Aerospace Applications Conference, 1996. IEEE, Volume: 1 , pp. 293 -307