VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

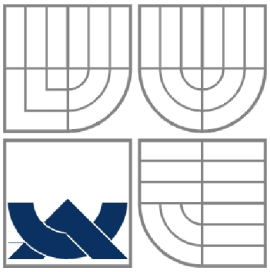
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

# NÁVRH TESTERU PAMĚTI RAM VE VHDL

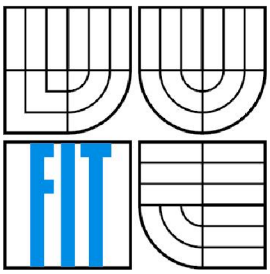
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. JIŘÍ CHARVÁT
AUTHOR

BRNO 2007



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

# NÁVRH TESTERU PAMĚTI RAM VE VHDL

RAM - TESTER DESIGN IN VHDL

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. JIŘÍ CHARVÁT

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. JOSEF STRNADEL, Ph.D.

BRNO 2007

*Zadání diplomové práce/6182/2006/xcharv10*

**Vysoké učení technické v Brně - Fakulta informačních technologií**

Ústav počítačových systémů Akademický rok 2006/2007

# Zadání diplomové práce

Řešitel: **Charvát Jiří, Bc.**
Obor: Počítačové systémy a sítě
Téma: **Návrh testeru paměti RAM ve VHDL**
Kategorie: Počítačová architektura

Pokyny:

1. Seznamte se se základními typy pamětí
2. Seznamte se se základními prostředky jazyka VHDL pro popis číslicových systémů
3. Seznamte se s diagnostikou pamětí, základními principy testování pamětí
4. Ve VHDL navrhněte zadaný typ RAM paměti a proveďte jeho simulaci, popř. i syntézu
5. Ve VHDL navrhněte tester pro zadaný typ RAM paměti a proveďte jeho simulaci, popř. i syntézu
6. Ověřte schopnost testeru detekovat poruchu v paměti RAM
7. Experimentujte s několika typy testů, shrňte a vyhodnoťte výsledky

Literatura:


- Abramovici, M., Breuer, M. A., Friedman, A. D. Digital Systems Testing and Testable Design. IEEE CS Press, Pistacaway, 1990. 670 s. ISBN 0-7803-1062-4.
- Mazumder, P., Chakraborty, K. Testing and Testable Design of High-Density Random-Access Memories. Kluwer Academic Publishers, Dordrecht, 1996. 386 s. ISBN 0-7923-9782-7.
- Adams, R. D. High Performance Memory Testing - Design Principles, Fault Modeling and Self-Test, Kluwer Academic Publishers, Dordrecht, 2003. 246 s. ISBN 1-4020-7255-4.
- Nadeau-Dostie, B. Design for At-Speed Test, Diagnosis and Measurement. Kluwer Academic Publishers, Dordrecht, 2000. 239 s. ISBN 0-7923-8669-8.
- Cohen, B. VHDL Coding Styles and Methodologies, Kluwer Academic Publishers, Dordrecht, 2001. 453 s. ISBN 0-7923-8474-1.


Při obhajobě semestrální části diplomového projektu je požadováno:

- Bez požadavků

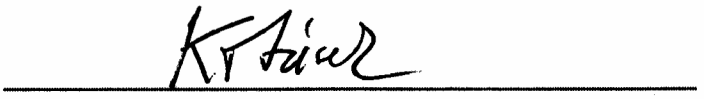
Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci ročníkového a semestrálního projektu (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVR-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Strnadel Josef, Ing., Ph.D.**, UPSY FIT VUT
Datum zadání: 15. června 2007
Datum odevzdání: 31. července 2007

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačových systémů a sítí
612 66 Brno, Božetěchova 2
L.S.



doc. Ing. Zdeněk Kotásek, CSc.
*vedoucí ústavu*

# LICENČNÍ SMLOUVA
# POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami

**1. Pan**

Jméno a příjmení: **Bc. Jiří Charvát**
Id studenta: 47047
Bytem: Řehořova 48, 618 00 Brno
Narozen: 04. 08. 1982, Brno
(dále jen "autor")

a

**2. Vysoké učení technické v Brně**

Fakulta informačních technologií
se sídlem Božetěchova 2/1, 612 66 Brno, IČO 00216305
jejímž jménem jedná na základě písemného pověření děkanem fakulty:

.....
(dále jen "nabyvatel")

## Článek 1
## Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
diplomová práce

Název VŠKP: Návrh testeru paměti RAM ve VHDL
Vedoucí/školitel VŠKP: Strnadel Josef, Ing., Ph.D.
Ústav: Ústav počítačových systémů
Datum obhajoby VŠKP: .....

VŠKP odevzdal autor nabyvateli v:
tištěné formě počet exemplářů: 1
elektronické formě počet exemplářů: 2 (1 ve skladu dokumentů, 1 na CD)

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

## Článek 2
## Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizování výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti:
   - ☒ ihned po uzavření této smlouvy
   - ☐ 1 rok po uzavření této smlouvy
   - ☐ 3 roky po uzavření této smlouvy
   - ☐ 5 let po uzavření této smlouvy
   - ☐ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

## Článek 3
## Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: .....

..... .....

Nabyvatel Autor

# Abstrakt

Tato práce popisuje problematiku hardwarového testování polovodičových pamětí. Popisuje princip fungování základních typů pamětí, způsob, jakým uchovávají data a způsob komunikace. Dále ukazuje typické poruchy, které v těchto pamětech mohou nastat. Součástí je také návrh a implementace modelu paměti a testeru v jazyce VHDL. Do paměti je možné zanést chyby a následně je připojeným testerem odhalit. Závěrem je nastíněno, jaká je úspěšnost při detekci různých druhů chyb použitím různých druhů testů. Zaměřuje se hlavně na detekci chyb pomocí march testu a jeho variant

# Klíčová slova

Test, paměť, RAM, paměťová buňka, SRAM, DRAM, statická paměť, dynamická paměť, chyby, March test, VHDL, dekodér adres.

# Abstract

This paper describes various approaches to hardware testing semiconductor memory. We describe the priciple of basic memory types, the way which each of them stores information and their comunication protocol. Following part deals with common failures which may occur in the memory. The section also describes the implementation of memory model and tester designed in VHDL language. It is possible to inject some errors into memory, which are later detected by the tester. The final section shows the response of tester to various error types according to used error detection method. The paper is especially focused on failure detection by variants of march test.

# Keywords

Test, memory, RAM, memory cell, SRAM, DRAM, static memory, dynamic memory, error, march test, address decoder.

# Citace

Jiří Charvát: Tester paměti RAM ve VHDL, Brno, FIT VUT v Brně, 2007

# Tester paměti RAM ve VHDL

## Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením
Ing. Josefa Strnadela, Ph.D.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jiří Charvát
31.7.2007

# Obsah

# 1 Úvod

Pro testování pamětí používaných v osobních počítačích již existuje mnoho nástrojů. Proč tedy vyvíjet specializované zařízení pro testování pamětí samostatně?

Testování pamětí softwarově uvnitř počítače má několik nevýhod. Jako nejpodstatnější se jeví nutnost použít rozhraní, které nabízí řadič paměti na základní desce. Chyby se tak sice podaří odhalit, ale někdy není možné odhalit o jakou závadu jde. Dále není možné testovat paměť celou, protože část zabírá samotný testovací program, popřípadě operační systém. Velkým problémem může být také vzájemná nekompatibilita některých součástí, například pamětí a základní desky. V tomhle případě se může fungující paměťový modul jevit jako vadný. To může nastat také při špatném nastavení časování nebo frekvence. Neposlední v řadě je také rychlost testování.

Softwarové testery jsou v zásadě dvou druhů. První z nich běží v prostředí operačního systému a pro komunikaci s pamětí využívá jeho služeb. Tento typ je, co se týče rychlosti, spolehlivosti detekce či zjištění typu závady, nejslabší, který můžeme použít. Implementací těchto testů je velmi mnoho a jejich kvalita se pohybuje v širokém rozpětí. O něco lépe je na tom test, který běží bez asistence operačního systému. Pro komunikaci s pamětí a periferiemi využívá vlastních nástrojů. Takový test zabere v paměti velmi málo místa a také komunikace může být optimalizována pro testování. Nicméně poruchy vzniklé nekompatibilitou hardware zůstávají. Tyto testy se obvykle spouštějí z bootovacího výměnného media, nebo bývají uloženy na pevném disku a bootloader je spouští místo operačního systému. Takové testery jsou tedy vhodné pro informaci typu, zda paměť pracuje správně spolu s ostatním vybavením. Pro testování většího množství modulů se příliš nehodí. Pokud test detekuje chybu, je nutné dále zjišťovat, která část hardwaru nepracuje správně.

Nejsilnějším nástrojem pro testování pamětí je vestavěná diagnostika, technologie známá pod názvem "*built-in self test*" (*BIST*) Takováto zařízení přímo od výrobce obsahují obvody, které za běhu kontrolují jeho chod a hledají závady. Výhody tohoto přístupu jsou zřejmé. Tester přímo na čipu nebo na desce spolu s testovanou pamětí může být velmi efektivní. Navíc výrobce testeru bývá většinou i výrobcem paměti, takže má přehled o struktuře testovaného obvodu a může testování lépe přizpůsobit. Přístup k místním sběrnicím také nabízí možnost masivního paralelizmu a tím další zvýšení rychlosti.

Další výhodou je, že testy probíhají v prostředí, ve kterém se dané zařízení provozuje. Je možné, že paměť, která při testování v průmyslovém testeru obstála, nebude v jiném zařízení v odlišných podmínkách (teplota, napájení) pracovat správně. Bývá velmi užitečné spojit BIST

s technologií "*bulit-in self repair*" (*BISR*). V této kombinaci např. po zjištění závady v některé paměťové buňce může dekodér adres přemapovat adresu této buňky do jiného místa redundantní pamětí, určenou pro tyto případy. Uživatel tak vůbec nemusí zaznamenat problém. Posledně zmiňovaná vlastnost může být také nevýhodou tohoto přístupu. Paměť může například pomalu degenerovat působením přehřívání. To by se dalo jednoduše odstranit, ale uživatel se o tomto problému dozví až při úplném selhání obvodu. Další podstatnou nevýhodou je nárůst složitosti obvodu, zvětšení plochy čipu, zvýšená spotřeba, ale v první řadě vyšší cena.

Specializované hardwarové testery eliminují některé nevýhody testování uvnitř počítače. Například můžeme ovlivnit komunikaci s pamětí na nejnižší možné úrovni. Samotná práce s pamětí také může být mnohem rychlejší. Zkouška paměti může být velice rychlá a přesná, ve většině případů také bývá odhalena příčina vzniku chyby, což může být velmi užitečné pro zkvalitňování postupu výroby. Neposlední v řadě je též konstrukce slotu pro paměťový modul. Výrobci základních desek nepočítají s velkou frekvencí výměny pamětí, tudíž sloty na základních deskách mnoho výměn nevydrží, a také samotná výměna je uživatelsky nepohodlná. Hardwarový tester může být vybaven mnohem lepším a také dražším konektorem. Nevýhodou tohoto řešení je jeho cena a nutnost zakoupení zařízení navíc.

V dnešní době je tendence umísťovat co nejvíce komponent systému (paměť, procesor, řadiče) do jednoho čipu. V tomhle případě není způsob testování pomocí externího zařízení vhodný, protože neexistuje přímý přístup k rozhraní paměti.

Takovéto testovací zařízení může najít uplatnění v mnoha oblastech. Například pokud je nutno kontrolovat větší množství paměťových modulů před zabudováním do počítačového systému, nebo jako výstupní kontrola při výrobě.

V této práci se zabývám převážně statickými a dynamickými polovodičovými paměťmi, jejich typickými poruchami a způsoby jejich nalezení.

# 2 Polovodičové paměti

O paměti v souvislosti s informačními technologiemi mluvíme jako o zařízení, které uchovává instrukce nebo data programů. Tyto informace mohou být uloženy mnoha způsoby, za použití různých technologií. Použitá technologie největší měrou ovlivňuje parametry výsledné paměti, podle kterých se vybírá vhodná paměť pro danou aplikaci. Parametry, které obecně u pamětí sledujeme, jsou zejména tyto:

- Kapacita - množství dat, které paměť dokáže najednou uchovat.
- Přístupová doba - délka časového úseku, který uplyne od okamžiku zaslání požadavku na přenos dat do okamžiku, kdy přenos dat započne.
- Přenosová rychlost - množství dat, které se do paměti zapíšou/přečtou za jednotku času.
- Způsob přístupu - může být náhodný nebo sekvenční. U náhodného se požadovaná operace provádí pouze nad těmi daty, nad kterými chceme (určenými adresou). U sekvenčního přístupu se obvykle musí přečíst všechna data, která jsou umístěna "před" blokem dat, nad kterým se operace provádí. Například převinutí magnetické pásky.
- Statičnost a dynamičnost - schopnost uchovat data bez zásahu zvenčí. Statická paměť uchová data dokud jsou připojené ke zdroji napájecího napětí. U dynamických pamětí tohle neplatí, informace se postupem času ztrácí a je nutné je pravidelně obnovovat.
- Energetická závislost - udává, zda je paměť schopna uchovat data i po odpojení od zdroje napájecího napětí. Energeticky závislé paměti se označují anglickým termínem "*volatile*" a nezávislé "*non-volatile*".
- Destruktivnost operací - tato vlastnost udává, zda daná operace provedená nad pamětí způsobí ztrátu dat. Například u dynamické paměti RAM je čtení destruktivní, tedy po přečtení dat je nutno ta samá data do paměti opět zapsat.
- Znovupoužitelnost - způsob, jakým se paměť připravuje na další použití. Tato vlastnost zahrnuje, zda se jednou nahraná paměť dá přepsat novými daty, eventuelně kolikrát je ji možné přepsat. Zda je možný přímý přepis, nebo se paměť musí před zapsáním nových dat připravit (smazat).
- Důležitými parametry jsou například cena za bit, spolehlivost, spotřeba elektrické energie, odolnost proti vnějším vlivům nebo fyzické rozměry. Pokud nahlédneme na paměť, jako na celý modul, může být dalším určujícím prckem přítomnost podpůrných funkcí, jako třeba kontrola parity ECC("error correcting code") nebo přítomnost registrů ("*buffered*" nebo též "*registred*" paměti).[1][2]

V mé práci jsem se zabýval paměťmi označovanými jako polovodičové. Jsou to takové paměti, v kterých je informace uložena pomocí logických hradel tvořených polovodičovými součástkami. Díky svým vlastnostem bývají používány v počítačové technice jako registry procesoru, rychlé

vyrovnávací paměti různých zařízení, nebo třeba operační paměť. Tyto paměti bývají obvykle souhrnně označovány zkratkou "*RAM*" (*Random Access Memory*). Tedy paměti s náhodným přístupem. Znamená to, že k datům uloženým v paměti tohoto typu se dá přistupovat přímo, ať jsou uloženy na kterémkoliv (náhodném) místě. Výsledkem je, že přístup do kteréhokoliv místa adresového prostoru trvá stejnou dobu a paměť mezitím neprovádí žádné další operace, jako je třeba vyhledávání dané adresy.

Toto označení není příliš přesné, protože nic neříká o povaze paměti, nýbrž pouze o typu přístupu k ní. Paměti, kterými jsem se zabýval, patří primárně do skupiny označované jako "*RWM*" (*Read-Write Memory*), paměti určené pro čtení i zápis. Dělení podle typu přístupu je až podružné. Přesnější označení těchto pamětí by tedy bylo "polovodičové paměti RWM s náhodným přístupem".

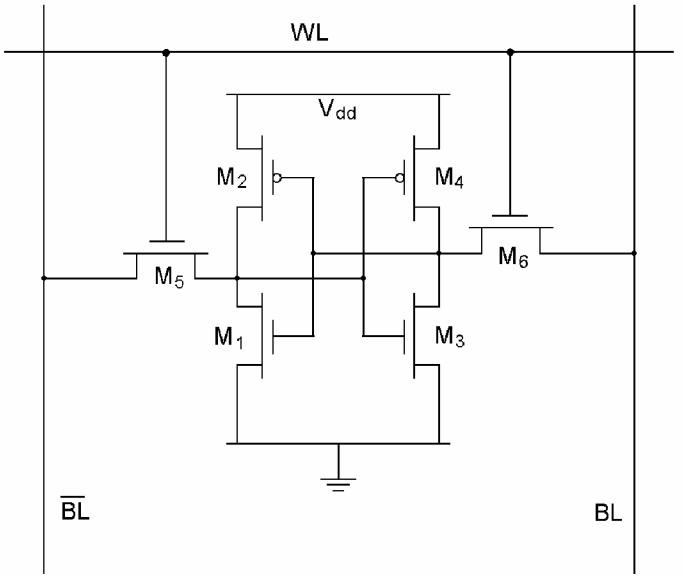
Rozlišujeme dva základní typy těchto polovodičových pamětí, běžně používaných ve výpočetní technice. Jsou to statické paměti, označované zkratkou "*SRAM*" (*Static Random Access Memory*), a dynamické paměti, známé jako "*DRAM*" (Dynamic *Random Access Memory*). Hlavní rozdíl mezi nimi je ve způsobu uchování dat.

Statické paměti udržují informaci v bistabilním klopném obvodu, její stav tedy zůstává nezměněn, dokud je buňka připojena k napájecímu napětí. Oproti tomu v dynamických pamětech reprezentuje hodnotu elektrický náboj udržovaný v kondenzátoru. Kondenzátor se ovšem časem pomalu vybíjí, zapsaná data je tedy nutné periodicky obnovovat.

## 2.1 Statické paměti

Paměti typu SRAM jsou realizovány jako bistabilní klopný obvod (flip-flop). Obvod se může nacházet v jednom ze dvou stabilních stavů a tím určuje, jaká hodnota "0" nebo "1" je v něm uložena. Této vlastnosti je dosaženo pomocí zpětné vazby, kdy tranzistor svým výstupem ovlivňuje sám sebe. Jedna paměťová buňka sestává z několika tranzistorů, jejich počet se pohybuje až kolem šesti v závislosti na použité technologii výroby.

Statické paměti se převážně vyrábějí s použitím technologie "*CMOS*" (*Complementary metal–oxide–semiconductor*), tedy pomocí polovodičů na bázi oxidu kovů nebo "*TTL*" (*Tranzistor-Tranzistor logic*) obvody s převládajícími tranzistory. Přepínání stavů se děje pomocí silného externího signálu. Struktura paměťové buňky je patrná z obrázku č.1.



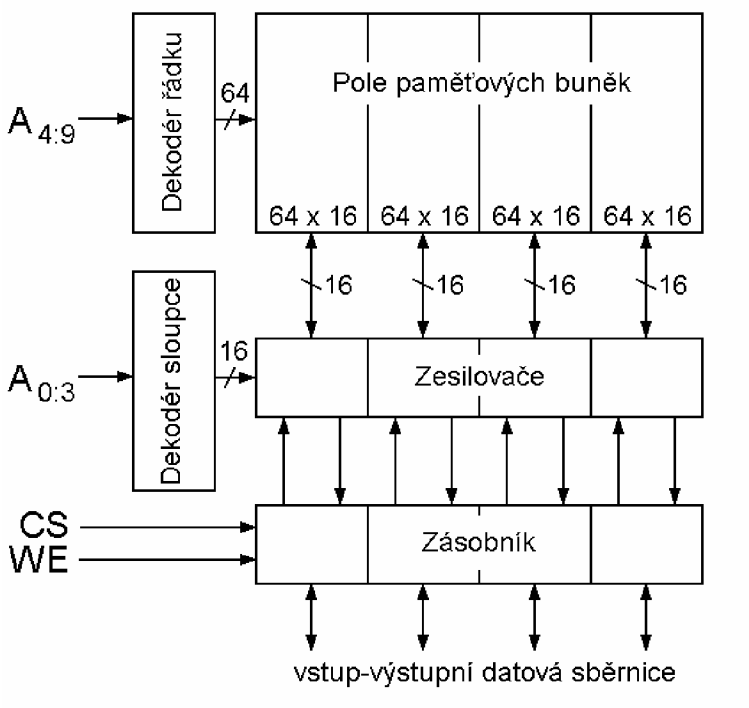

Obrázek 1. Paměťová buňka SRAM v technologii CMOS

Tranzistory M5 a M6 oddělují paměťovou buňku od datových vodičů, samotná informace je uchována v křížem zapojených tranzistorech M1 až M4, kde se uplatňuje zpětná vazba. Přítomnost druhého datového vodiče $\overline{BL}$ invertovaného BL není nezbytná, ale obvykle se používá, protože pomáhá redukovat šum a zvyšuje spolehlivost při čtení nebo zápisu.

Čtení dat probíhá následovně:

Na datové vodiče BL i $\overline{BL}$ se nastaví úroveň log.1. Následně se signálem WL otevřou přístupové tranzistory M5 a M6. V případ že v paměťové buňce byla uložena hodnota "1", vodič $\overline{BL}$ se přes tranzistory M1 a M5 uzemní a dostane se tak na něj hodnota "0", zatímco BL zůstane beze změny připojen na napájecí napětí a tudíž na něm bude hodnota 1.Pokud v paměťové buňce byla 0 bude situace opačná. Na BL se dostane hodnota "0" a na $\overline{BL}$ "1" . Je zřejmé, že po čtení zůstane uložená hodnota nezměněná, Čtení dat je tedy nedestruktivní.

Zápis probíhá podobně. Nejdříve se na datové vodiče se nastaví hodnota kterou si přejeme zapsat. Následně se po nastavení úrovně "1" na WL otevřou M5 a M6 a tím dojde k překlopení obvodu. K překlopení dochází, protože přicházející signály jsou "silnější" (vyšší úroveň) než stávající úroveň napětí mezi tranzistory. Je nutný pečlivý výběr parametrů tranzistorů aby k překlopení obvodu došlo. Příklad implementace modulu paměti je na obrázku 2.




Obrázek 2. Příklad implementace modulu paměti SRAM

V tomto případě jsou paměťové buňky uspořádány do 64 řádků a 16 sloupců. Délka slova jsou 4 bity. Adresa slova je desetibitová $A_{0\text{-}9}$, přičemž horní 4 bity $A_{0\text{-}3}$ slouží k adresování řádku a spodních 6 bitů $A_{0\text{-}3}$ určuje sloupec. Statické paměti jsou velmi rychlé, bohužel je jejich rychlost vykoupena velkou plochou na čipu, vysokou spotřebou a také cenou. Kvůli těmto vlastnostem se statické paměti vyrábějí téměř výhradně s malou kapacitou.

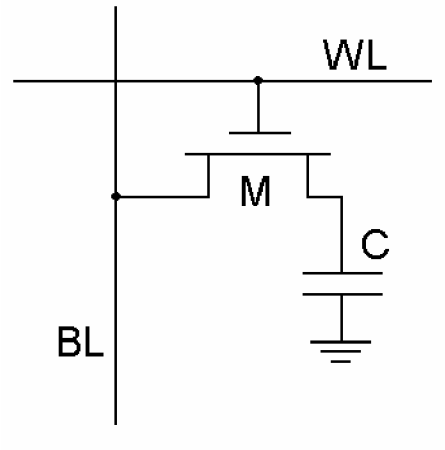
Díky nízké kapacitě není potřeba šetřit vývody pouzdra paměti a nepoužívá se multiplexování adresy na adresu řádku a sloupce, nýbrž se posílá celá adresa najednou. To vede k jednoduššímu časování, ale také k nárůstu počtu vývodů pouzdra paměti. Signály CS ("*Chip Select*") slouží k výběru modulu, se kterým chceme pracovat a WE ("*Write Enable*") k výběru, zda se bude z paměti číst, nebo naopak do ní zapisovat. Průběh signálů při čtení a zápisu ukazuje obrázek 3.


Obrázek 3. Průběh signálů při zápisu/čtení v SRAM

Řadič paměti musí zajistit správné načasování signálů. Při čtení musí vědět, od jaké doby jsou na výstupu platná data. Stejně tak při zápisu, aby během doby, kdy je aktivní CS, byla nastavená správná adresa a data. [3]

## 2.2 Dynamické paměti

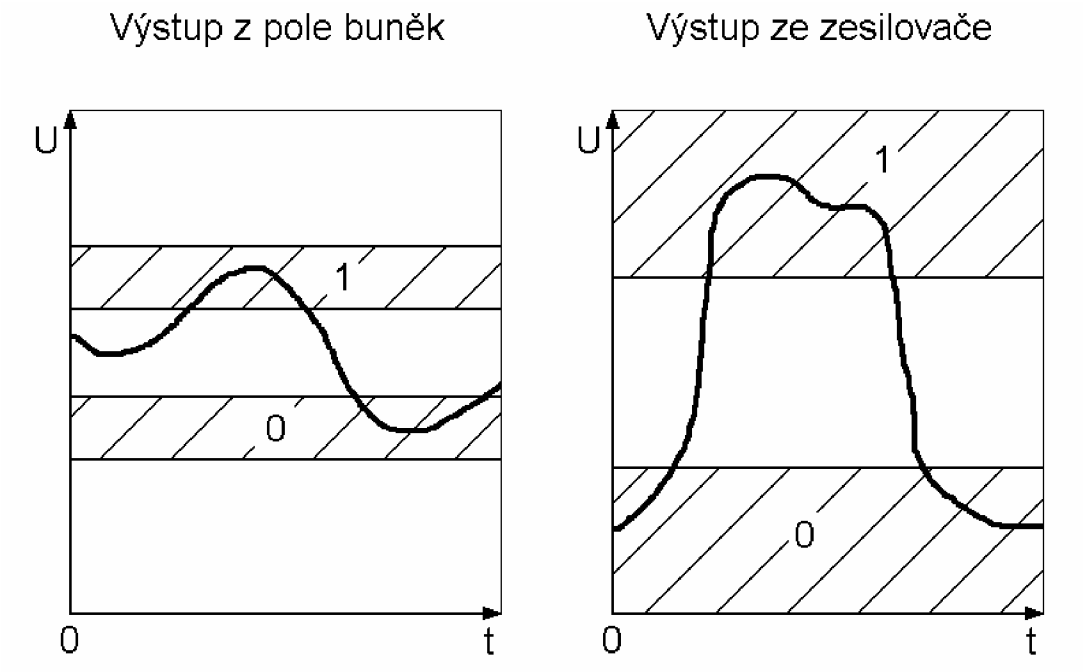
Dynamické paměti pracují na odlišném principu. Hodnota bitu je zde udržována pomocí nabíjení a vybíjení kondenzátoru. Paměťová buňka sestává z kondenzátoru C, který ukládá informaci v podobě elektrického náboje a z výběrového tranzistoru M, který slouží jako přepínač pro volbu kondenzátoru. Schéma paměťové buňky je na obrázku č.4




Obrázek 4. Paměťová buňka DRAM v technologii CMOS

Čtení probíhá po otevření tranzistoru M signálem WL. Tím se náboj z kondenzátoru dostane na výstup BL. Výsledná hodnota je čtena po zesílení zesilovačem. Při zápisu je po otevření tranzistoru M kondenzátor nabit nebo vybit podle zapisované hodnoty.

Zesilovač signálu je nutný u pamětí, kde pole paměťových buněk a okolní obvody používají jiné úrovně signálu pro logickou “0“ nebo “1“. Takový případ je znázorněn na obrázku č.5. Vyšrafované oblastí znázorňují, jaká úroveň signálu je buňkami, respektive okolní logikou, vyhodnocena jako “0“, a jaká úroveň jako “1“.



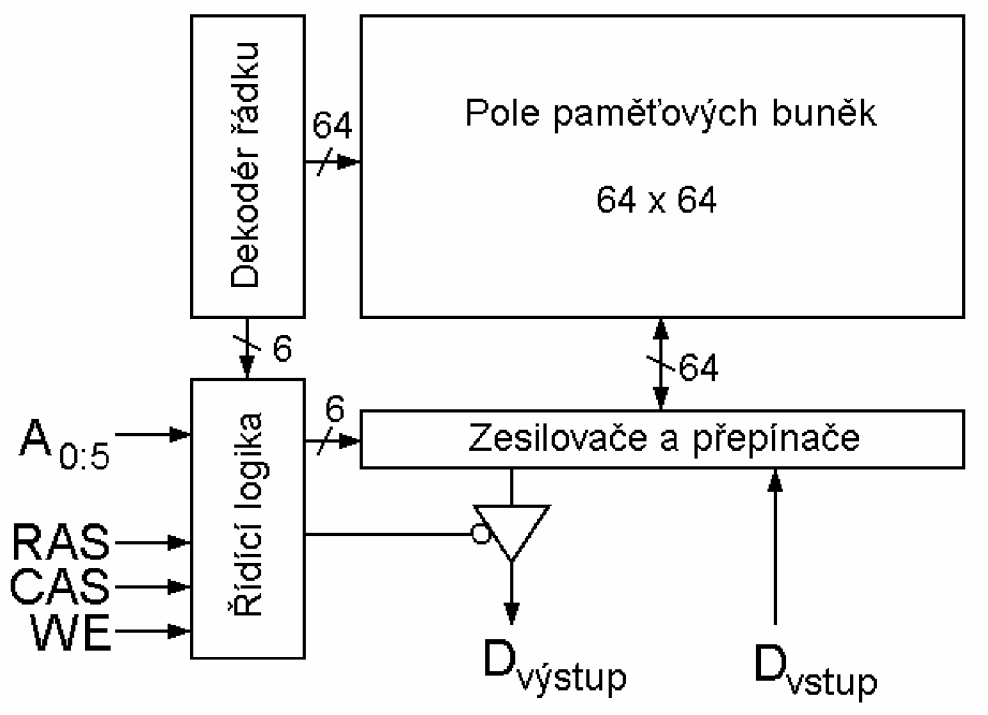

Obrázek 5. Průběh úrovně signálu před a po průchodu zesilovačem.

Zesilovače snižují dobu propagace signálu mezi přistupovanou paměťovou buňku a okolní logikou, která je na paměťovém modulu. Dále také sjednocují úrovně logické "0" a "1" mezi paměťovou buňkou a okolními obvody. [4] Používají se jak při čtení, tak při zápisu dat.

Uchování informací pomocí náboje v kondenzátoru má několik nevýhod. Mezi ně patří to, že kondenzátor je při čtení vždy vybit (čtení je destruktivní operace), takže po přečtení dat se musí hodnota opět zapsat. Čtení tedy trvá déle než zápis.

Paměťové moduly jsou obvykle organizovány jako dvourozměrná matice buněk a čtení i zápis probíhá po řádcích - celý řádek najednou, je tedy nutné zapsat zpět celý řádek. Mimoto se kondenzátory také časem samovolně vybíjejí a za určitý čas tak uloženou informaci ztratí. Je nutno tedy periodicky obnovovat i data, se kterými se nepracuje. Maximální doba, po kterou paměť data udrží bez obnovení, se pohybuje až kolem 64 ms v závislosti na konkrétním provedení. Struktura modulu je podobná jako u SRAM, příklad implementace na obrázku č.6.

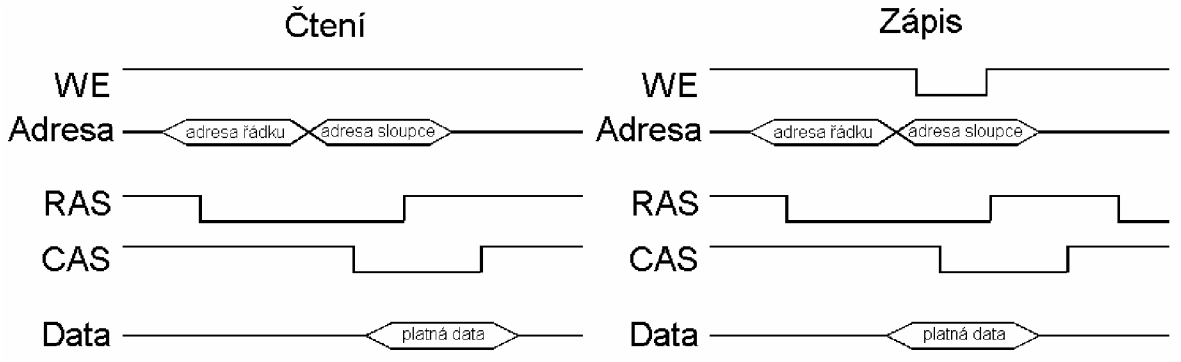
Zde vlastní paměť tvoří pole buněk o velikosti 64x64 bitů. Oproti SRAM byl signál CS nahrazen signály RAS ("*Row Address Strobe*") a ("*Column Address Strobe*"). Tyto signály určují typ adresy, která se zrovna nachází na adresové sběrnici $A_{0\text{-}6}$. Pokud je aktivní CAS, znamená to, že na $A_{0\text{-}6}$ je adresa sloupce. Pokud je aktivní RAS, logika očekává přítomnost adresy řádku.




Obrázek 6. příklad implementace modulu paměti DRAM

Přenos adresy tedy probíhá ve dvou krocích a celá paměť je adresována 12-ti bity. V důsledku toho je možno přistupovat ke každému bitu v paměti samostatně, což ale není obvyklé. Rozdělení (multiplexování) adresy na dvě části (RAS a CAS) je z toho důvodu, že dynamické paměti mívají obvykle velké kapacity a k jejich adresování je potřeba velká šířka adresy.

Pokud by se tato adresa přenášela najednou, bylo by pouzdro s pamětí příliš velké. Navíc by musel být k dispozici velký adresový buffer. Dalším rozdílem je oddělení vstupu a výstupu. Časování signálů při čtení je na obrázku č.7.




Obrázek 7. Průběh signálů při zápisu/čtení v DRAM

Z průběhů na obrázku 6 je vidět, jak probíhá přenos adresy buňky. Nastavením signálu RAS na “0“ indikuje, že na adresní sběrnici je platná adresa řádku. Hodnota je dočasně uložena v řídící

logice. Po definované době řadič paměti vymění adresu řádku za adresu sloupce a vynuluje i signál CAS. V tomto okamžiku paměť nastaví požadovaná data na datovou sběrnici. Protože však čtení je destruktivní operace, musí se zajistit zpětný zápis. Ten nastává ve chvíli, kdy se RAS vrací zpět do hodnoty "1". (na datové sběrnici jsou stejná data, která byla čtena). Po ukončení zpětného zápisu je CAS nastaven na "1". Prodleva mezi signály RAS a CAS (tzv. RAS-CAS delay) a také jejich délka je pevně daná parametry paměti.

Zápis probíhá obdobně, jenom je vynechána fáze čtení a signálem WE je indikován zápis nových dat. Vzhledem k složitějšímu ovládání DRAM je nutné, aby řadič přesně načasoval průběh signálů.

K eliminaci ztráty dat samovolným vybíjením kondenzátorů v dynamické paměti slouží periodická obnova dat. Existuje několik způsobů, jak ji provádět.

- Blokové obnovení - čítač sleduje čas, po který data nebyla obnovována (například může počítat hodinové signály) a po dosažení nastavené hranice provede obnovu celé paměti řádek po řádku. Nevýhodou tohoto přístupu je, že paměť je během této doby nepřístupná.
- Distribuované obnovení - doba, kterou paměť udrží data bez obnovy, se vydělí počtem řádků. Získané číslo je potom perioda, se kterou probíhá obnova dalšího řádku. Oproti blokovému obnovování má tato metoda tu výhodu, že doba, po kterou je paměť nepřístupná, se rozdělí na více menších úseků a pokud je s ní třeba komunikovat, nestane se, že bychom museli čekat, než se celá paměť obnoví.
- RAS only refresh - nejjednodušší typ obnovení paměťové buňky. Spočívá v tom, že je proveden prázdný čtecí cyklus, při němž je sice aktivován signál RAS a do paměti je přivedena adresa řádku (refresh adresa), signál CAS ale zůstává neaktivní. Paměť pak svojí vnitřní logikou načte řádek a načtená data zesílí. Při neaktivním signálu CAS nejsou ale přivedena na výstup do výstupního bufferu, nýbrž zpět zapsána. Dojde tak k obnově. Tento druh obnovení je řízen řadičem paměti.
- CBR (CAS Before RAS refresh) - pokud je signál CAS vyvolán před RAS. Adresa se určuje podle vnitřního obnovovacího čítače adres, který se při obnovení řádky inkrementuje. Díky němu také tato metoda snižuje nároky na spotřebu energie. Používá se nejvíce.
- Hidden refresh - obnovovací cyklus je skryt za cyklus čtení. Signál CAS je udržován na nízké úrovni a mění se pouze RAS. [5]

Maximální doba, po kterou paměť udrží bez obnovení, se pohybuje až kolem 64 ms, záleží ale na provedení. Samotné obnovování nepředstavuje nijak výrazné zpomalení, zabere méně než 1% celkového času. [6]

# 3 Poruchy v polovodičových pamětech

Závada v paměti může vzniknout z mnoha příčin. Tyto příčiny jsou silně závislé na typu paměti a na technologii výroby, což způsobuje, že jeden typ pamětí je proti některým faktorům způsobujícím poruchy odolnější než jiný.

## 3.1 Příčiny vzniku poruch

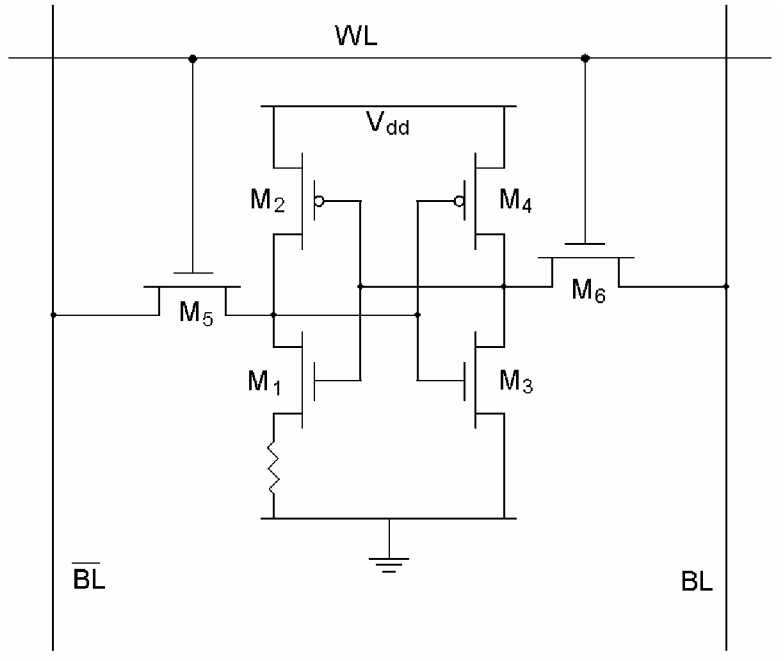
Poruchy, které vznikají při výrobě, bývají zpravidla způsobeny nečistotami v použitých materiálech. Dále potom mohou vzniknout například působením statické elektřiny nebo špatnou úrovní napájecího napětí. Mezi další faktory, které mohou způsobit poruchu, patří extrémní teploty, ale třeba i magnetické pole či ionizující záření.[7]

Tyto a další negativní faktory mohou zapříčinit některé z následujících jevů, které způsobí, že paměť nebude pracovat správně. Tento výčet zdaleka není úplný, závisí na typu paměti a použité technologii výroby. Může dojít například k proražení kondenzátoru, zkratu sběrnice, přerušení sběrnice nebo kteréhokoliv jiného vodiče, zkratu tranzistoru., propojení tranzistoru na zem a k mnoha jiným jevům.

Při výskytu některé z těchto poruch se paměť obvykle začne chovat jinak, než se od ní očekává. V konečném důsledku dochází k tomu, že data čtená z paměti se neshodují s těmi, která do ní byla uložena, dochází tedy k chybě dat v důsledku fyzické poruchy v paměti.

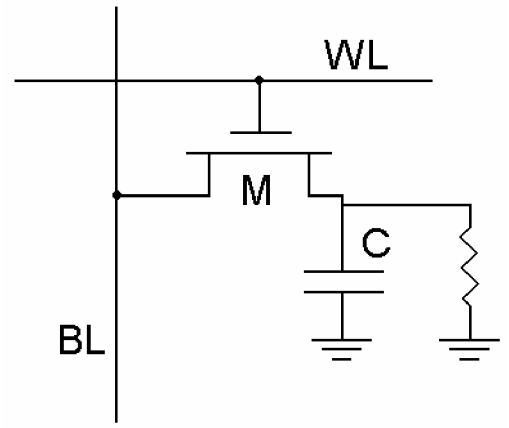
Pro ilustraci uvádím několik poruch, které se mohou vyskytnout.

- Destruktivní čtení v paměti, kde čtení není destruktivní operace. Například v statických pamětech není čtení destruktivní operací, za určitých podmínek však může tuto vlastnost ztratit. Pokud například v statické paměťové buňce dojde k poruše, jak je znázorněno na obrázku č.8, kdy dojde k přerušení spojení tranzistoru $M_1$ s uzemněním, dochází při čtení z buňky k překlopení a tudíž k inverzi hodnoty, kterou buňka obsahuje. Přerušení vodiče je zde znázorněno jako odpor.




Obrázek č.8 Statická paměťová buňka s přerušeným uzemněním tranzistoru $M_1$

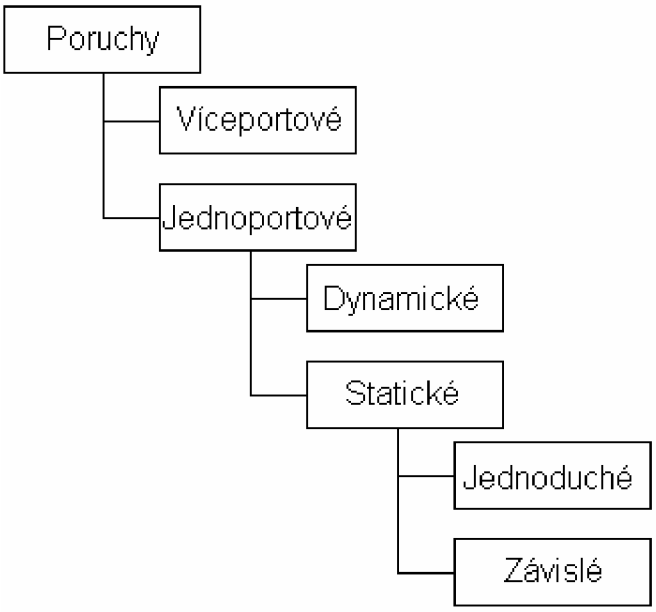
- Porucha přednabíjení hodnot. Při čtení hodnot z některých typů pamětí bývá využíváno efektu, kdy se na některý vodič nastaví vysoká úroveň signálu a posléze se tato hodnota po připojení k paměťové buňce buďto změní na nízkou úroveň uzemněním, nebo zůstane vysoká. Pokud tenhle mechanizmus nefunguje správně, může se místo vysoké hodnoty přečíst nízká. K tomuhle obvykle dochází, pokud po sobě těsně následují nejdříve operace zápis a potom čtení, kdy se nestačí dostatečně rychle nastavit příslušné hodnoty. [8]
- Ztráta informace. U této poruchy dochází k narušení funkce paměťové buňky takovým způsobem, že buňka ztratí schopnost uchovat informaci. U statických pamětí může být tato závada způsobena přerušením uzemňovacího vodiče, jak je naznačeno na obrázku č.7, nebo naopak přerušením přívodu napájení k tranzistoru $M_2$ nebo $M_4$. V takovém případě nedochází k překlápění klopného obvodu při zápisu dat. U dynamických pamětí může být tahle vada způsobena uzemněním pólu kondenzátoru, který je připojen na tranzistor, jak je na obrázku č.9. Podle velikosti odporu tohoto zkratu tak dochází k rychlejšímu vybíjení kondenzátoru než obvykle a buňka tak může ztratit informaci dříve než dojde k obnovení.




Obrázek č.9 Dynamická paměťová buňka se zkratem na kondenzátoru

## 3.2 Klasifikace poruch

Podle toho, jakým způsobem se závada projevuje, rozdělujeme poruchy do několika kategorií, jak je znázorněno na obrázku č.10. Na obrázku pro zjednodušení není zakreslen celý strom možných projevů poruch, ale pouze jedna větev. Strom je symetrický, tedy i poruchy ve víceportových pamětech mohou být dynamické a statické, a dynamické mohou být jednoduché a závislé.




Obrázek 10. rozdělení poruch podle vnějších projevů

### 3.2.1 Poruchy jednoportové a víceportové

Podle typu paměti se v první řadě chyby dělí na poruchy jednoportových a víceportových pamětí. Rozdíl mezi nimi tkví v počtu současně prováděných operací. U jednoportových pamětí je to maximálně jedna operace. Tedy v jednom okamžiku probíhá zápis nebo čtení na jednu paměťovou buňku, pokud jde o bitově orientovanou paměť, nebo na skupinu buněk v případě slovně orientovaných pamětí. Jednoportové jsou například operační paměti v osobních počítačích.

U víceportových pamětí může být počet současně prováděných operací vyšší. Z takové paměti můžeme například číst z jedné adresy a zároveň zapisovat na jinou. Používají se například v grafických akcelerátorech, nebo jako rychlé vyrovnávací paměti.

### 3.2.2 Poruchy statické a dynamické

Další dělení je podle počtu operací nutných k tomu, aby se porucha projevila. K vzniku statické chyby postačí pouze jediná operace. Pro ilustraci zápis libovolné hodnoty do paměťové buňky nemá vliv na stav této buňky. U dynamických chyb je k výskytu chyby nutné provést operací několik. Třeba zápis stejné hodnoty dvakrát po sobě a následné čtení vrátí nesprávnou hodnotu.

### 3.2.3 Poruchy jednoduché a závislé

Poruchy se dále dělí na jednoduché a závislé. Jednoduchá porucha není ovlivněna chováním jiné poruchy, kdy může dojít k vzájemnému ovlivňování vnějších projevů. Naopak u závislé poruchy k ovlivňování dochází Například pokud díky přerušení adresového vodiče dojde k deaktivaci celého sloupce v matici paměťových buněk, zamaskují se poruchy, které vznikají díky poruchám buněk v tomto sloupci. [9]

## 3.3 Typické projevy poruch

V této kapitole jsou popsány nejčastější chyby, které vznikají při operacích s pamětí jako důsledek poruch. U každé je v závorce uveden anglický výraz, kterým se daná chyba označuje v různé literatuře. Dále uvádím zažité zkratky, které se pro jednotlivé chyby používají.

- Izolované chyby, postihující pouze jednu paměťovou buňku nezávisle na okolí
    - Stálá hodnota (“*State Fault*“) (SF) - v buňce je trvale hodnota “1“ nebo “0“ bez ohledu na to, jaké operce se s pamětí provádí.
    - Chyba inverze (“*Transition fault*“) (TF) - po prvním zápisu hodnoty do buňky nejde již tato hodnota změnit, nezdaří se invertující zápis.

- Destruktivní zápis (“*Write disturb fault*“) (WDF) - zápis stejné hodnoty do buňky vede k inverzi. Například buňka obsahovala “1“, následně do ní byla zapsána “1“, výsledný obsah je “0“.
- Destruktivní čtení (“*Read destructive fault*“) (RDF) - operace čtení vrací invertovanou hodnotu. Obsah buňky se též změní na opačnou hodnotu. Například buňka obsahovala “1“, přečte se “0“, výsledná hodnota v buňce bude “0“.
- Maskované destruktivní čtení (“*Deceptive read destructive fault*“) (DRDF) - operace čtení vrátí správnou hodnotu, ale obsah buňky je změněn na opačný. Například buňka obsahovala “0“, přečte se “0“ výsledná hodnota v buňce je “1“.
- Chybné čtení (“*Incorrect read fault*“) (IRF) - při čtení vrátí invertovanou hodnotu, obsah buňky však nezmění. Například v buňce je hodnota “0“, přečte se “1“ ale hodnota v buňce zůstane “0“.

- Závislé chyby - vznikají interakcí několika buněk. Buňka, jejíž hodnota nějakým způsobem ovlivňuje jinou, se označuje jako řídící, buňka, která je ovlivňována, jako závislá.
  - Závislá hodnota (“*State coupling fault*“) (CFst) - hodnota jedné buňky přímo závisí na hodnotě jiné buňky. Například pokud je v řídící buňce “0“, potom je v závislé také “0“ a naopak.
  - Závislá destruktivní operace (“*Disturb coupling fault*“) (CFds) - operace nad jednou buňkou (zápis/čtení) změní hodnotu v jiné buňce.
  - Závislá inverze (“*Transition coupling fault*“) (CFtr) – bude-li v jedné buňce určitá hodnota, nezdaří se invertující zápis do jiné buňky. Například pokud bude v řídící buňce “1“ a v závislé “0“, po zápisu “1“ do závislé bude její hodnota stále “0“.
  - Závislý destruktivní zápis (“*Write destructive coupling fault*“) (CFwd) - bude li v jedné buňce určitá hodnota, selže neinvertující zápis do jiné buňky. Tedy hodnota bude změněna. Například pokud v řídící buňce bude “0“ a v závislé “1“, potom po zápisu “1“ do závislé se její hodnota změní na “0“.
  - Závislé destruktivní čtení (“*Read destructive coupling fault*“) (CFrd) – bude-li v jedné buňce určitá hodnota, pak čtení v jiné buňce invertuje její obsah a tuto hodnotu vrátí. Například bude-li v řídící buňce “0“ a v závislé také “0“, potom čtení ze závislé buňky vrátí “1“ a její obsah změní na “1“.
  - Maskované závislé destruktivní čtení (“*Deceptive read destructive coupling fault*“) (CFdrd) – bude-li v jedné buňce určitá hodnota, pak při čtení jiné získáme správnou hodnotu, ale zároveň dojde k inverzi obsahu. Například pokud bude v řídící buňce “1“ a v závislé “1“, potom čtení ze závislé vrátí “1“, ale její obsah invertuje na “0“.
  - Chybné závislé čtení (“*Incorrect coupling fault*“) (CFir) – bude-li v jedné buňce určitě hodnota, potom čtení v jiné sice nezmění její hodnotu, ale vrátí inverzní. Například pokud

bude v řídící buňce “0“ a v závislé “1“, potom čtení ze závislé vrátí “0“, ale její hodnota zůstane “1“.

Mimo těchto závad, které se vyskytují nezávisle na operacích, které byly s pamětí prováděny předtím, než závada nastala, existuje ještě skupina takzvaných dynamických chyb. Tyto se objevují pouze po nějaké sekvenci operací. Vznikají při existenci parazitní kapacity mezi datovými vodiči, tato kapacita se může postupem času nabíjet a tím následně ovlivnit čtená nebo zapisovaná data.
Pokud tyto chyby postihují pouze jednu buňku, mohou nastat následující situace.

- Destruktivní dynamické čtení (“*Dynamic read destructive fault*“) (dRDF) - po libovolné operaci (čtení/zápis) následuje čtení, při kterém je invertována hodnota buňky, tato je potom vrácena.
- Maskované dynamické destruktivní čtení (“*Dynamic deceptive read destructive fault*“) (dDRDF) - po libovolné operaci (čtení/zápis) následuje čtení, které vrátí správnou hodnotu, ale následně obsah buňky invertuje.
- Chybné dynamické čtení (“*Dynamic incorrect read fault*“) (dIRF) - po libovolné operaci (čtení/zápis) následuje čtení, které vrátí invertovanou hodnotu, obsah buňky zůstává zachován.
- Dynamická inverze (“*Dynamic transition fault*“) (dTF) - po libovolné operaci (čtení/zápis) následuje invertující zápis, v buňce však zůstane původní hodnota.
- Dynamický destruktivní zápis (“*Dynamic write disturb fault*“) (dWDF) - po libovolné operaci (čtení/zápis) následuje neinvertující zápis, v buňce však zůstane původní hodnota

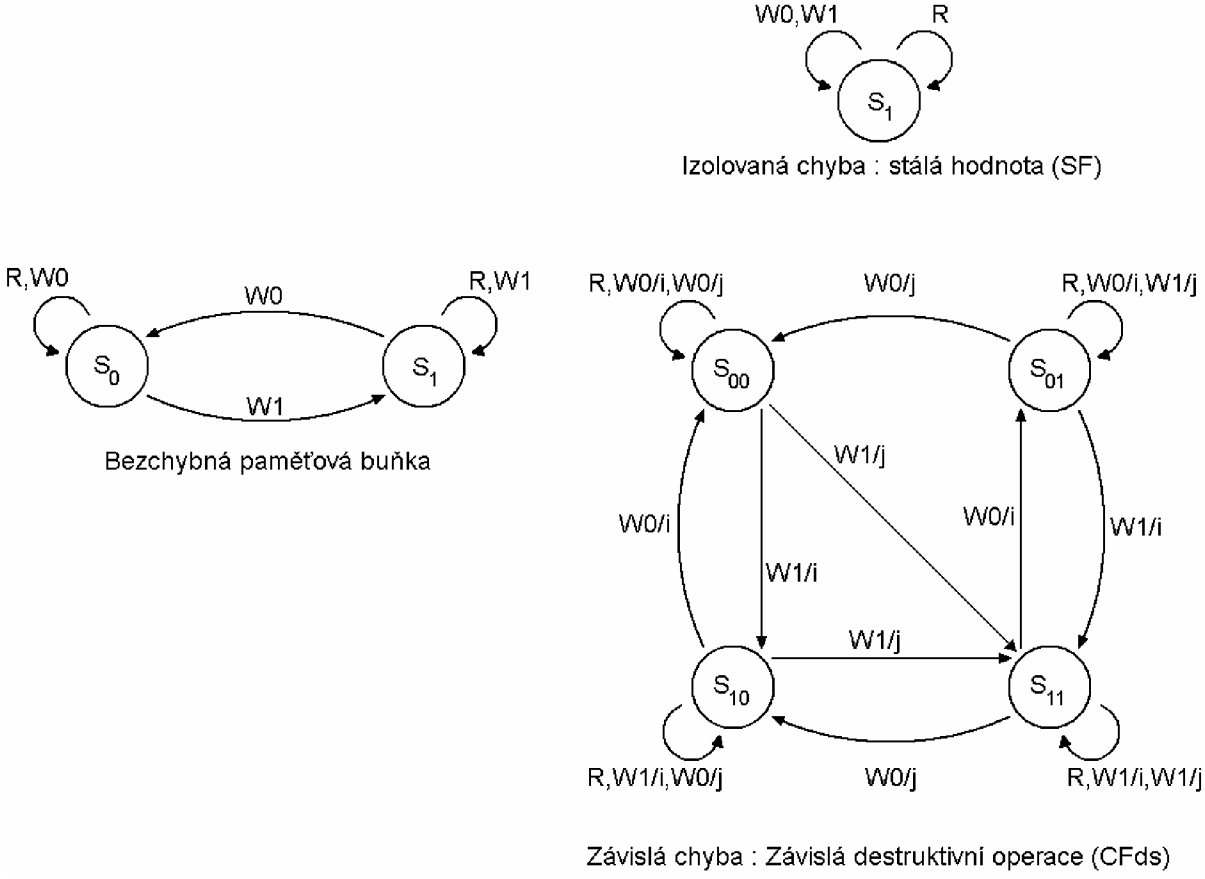
V případě, že dynamická porucha postihuje více než jednu buňku, je situace složitější. Kombinací, které se musí prověřit, je velmi mnoho a jejich detekce je velmi obtížná. Lze je souhrnně rozdělit do následujících kategorií.

- Několik operací za sebou nad jednou buňkou změní hodnotu sousední buňky.
- Jedna buňka je v určitém stavu. Několik operací za sebou na sousední buňce má za následek chybu v čtení nebo zapisování druhé buňky. Pouze však v případě, že první buňka je v určitém stavu.
- Operace v jedné buňce následovaná operací v druhé buňce má za následek poruchu v druhé buňce.
- Operace v jedné buňce následovaná operací v druhé má za následek poruchu v první buňce.

Chování chyb se dá dobře popsat pomocí Markovských diagramů. Ukázka jejich použití je na obrázku č. 11. Hodnotu uloženou v paměťové buňce reprezentují stavy $S_0$ a $S_1$.V $S_0$ je v buňce uložena “0“ v $S_1$ obsahuje “1“. Operace W0 a W1 značí zápis “0“ respektive “1“, operace R je čtení

z buňky. Diagram na obrázku vlevo ukazuje činnost bezchybné paměťové buňky, diagramy vpravo potom činnost buněk, u kterých vznikají chyby v důsledku nějaké poruchy. [10]

U izolované chyby je v buňce stálá hodnota "1"- je tedy ve stavu $S_1$ bez ohledu na prováděné operace. V případě závislé chyby máme dvě buňky, řídící a závislou, označené i a j. Každá z nich může uchovávat buď "0", nebo "1". Existují tedy celkem 4 kombinace jejich stavů, s nimi korespondují stavy $S_{00}$ až $S_{11}$. V tomhle příkladě se jedná o chybu, kdy při zápisu "1" do buňky j dojde zároveň k zápisu "1" do buňky i. V digramu se používá toto značení : Operace W0/i značí zápis "0" do buňky i, ostatní operace zápisu analogicky. Přestože je možné číst z každé buňky samostatně, v tomto příkladu je lhostejné z které buňky se čte, proto je zde operace čtení označena pouze písmenem R.




Obrázek 11. Model chyb pomocí markovských diagramů.

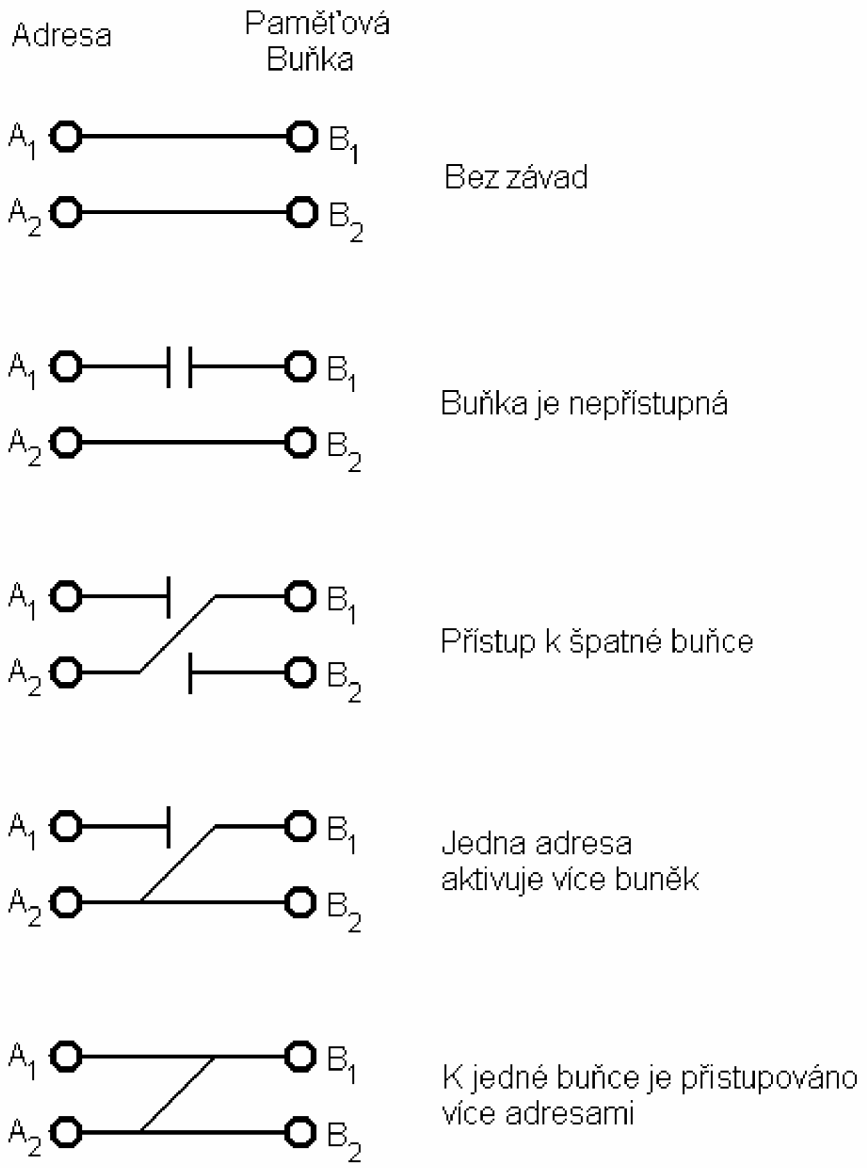
# 3.4 Poruchy adresového dekodéru

Aby bylo možno adresovat data v paměti, musí existovat dekodér, který každé fyzické paměťové buňce, resp. skupině buněk, přiřadí jednoznačnou adresu, pod kterou se dá k datům přistupovat. Pokud je možno ke každé paměťové buňce obsahující jeden bit přistupovat samostatně, má každá taková buňka jedinečnou adresu. O takové paměti mluvíme jako o bitově orientované. V případě, že

se pod jednou adresou skrývá skupina buněk, operace se provádějí nad větším množstvím bitů. Jedná se o slovně orientovanou paměť.

Dekodér tedy určuje, které adrese budou příslušet konkrétní paměťová místa. Této vlastnosti se využívá při výrobě samotné paměti. Během výrobního procesu se stává, že se v paměťovém čipu objeví několik vadných míst. Bylo by neekonomické vyřadit již téměř hotový paměťový modul. Proto se paměti vyrábějí s jistou redundancí. Vyrobený čip má větší kapacitu, než jaká je jmenovitá kapacita čipu a tato paměť "navíc" se využije jako náhrada za vadnou část, pokud se ve zbytku čipu objeví chyby. Paměťový dekodér jednoduše místo na chybný sektor ukazuje do bezchybné oblasti.[11]

Závady v dekodéru se obvykle snadno odhalí. Jednak se projevují stále a jednak se obvykle projevují stejně při čtení i zápisu. V adresovém dekodéru se mohou objevit závady, které jsou graficky znázorněny na obrázku č.12.




Obrázek 12. Poruchy v adresovém dekodéru.

Jednotlivé obrázky v pořadí odshora dolů, zobrazují následující poruchy.

- Bezchybný dekodér.
- Buňka je nepřístupná. Vybráním adresy buňky nedojde k její aktivaci, zápis ani čtení se neprovede. Při výskytu této poruchy daná adresa neukazuje na žádnou paměťovou buňku a zároveň k paměťové buňce nepřísluší žádná adresa.
- Přístup ke špatné buňce. V tomhle případě adresa aktivuje jinou buňku než bylo původně zamýšleno. Navenek se může projevovat jako dvojnásobná chyba předchozího typu. V případě, že se paměťová buňka, na kterou nově ukazuje adresa, nachází v redundantní oblasti, nemusí se nezbytně jednat o závadu, nýbrž o přemapování vadné paměťové buňky.
- Jedna adresa aktivuje více buněk. Přístup pomocí jedné adresy aktivuje více buněk, tedy při zapisovaná hodnota je uložena ve více buňkách. Při čtení posléze je tato hodnota vrácena.
- K jedné buňce je přistupováno více adresami. Porucha se projevuje podobně jako v předchozím případě, rozdíl je zde v tom, že může nastat situace, kdy v obou buňkách přistupovaných pod stejnou adresou mohou být odlišné hodnoty. Čtení z buněk v tomto případě může vracet náhodné hodnoty.

# 4 Mechanismy detekce poruch v polovodičových pamětech

Při testování pamětí a logických obvodů obecně se používá několik přístupů. Každý z těchto přístupů pracuje s jinými informacemi a odhaluje jiné typy poruch, které se v daném zařízení mohou vyskytnout.

Nejobecnější z těchto přístupů je testování na úrovni logických členů, takzvané funkcionální, nebo logické testování. Pracuje s logickými vstupy a výstupy testovaného zařízení. Výstupy se vyhodnocují jako logické hodnoty, to znamená že úroveň napětí nad danou hodnotu se vyhodnotí jako logická "1" a úroveň pod jinou hranici jako logická "0" a s těmito binárními daty se pracuje. Testy probíhají stylem, že na vstupy zařízení posíláme řídící signály a na výstupech kontrolujeme zda je odezva zařízení taková jakou očekáváme. Informace získané z testů tohoto typu obvykle slouží pro informaci, zda je zařízení vadné nebo ne. Dále může poskytnout informace která část selhala, a jakým způsobem se tato závada projevuje. Obvykle se ale nedá zjistit nic o tom, které logické členy selhaly, či z jaké příčiny závada vznikla.

Detailnější informace o závadě se dají získat pomocí testů, které pracují na nižší úrovní hardwaru. Například statické parametrické tety, které získávají informace přímo z úrovně napětí na výstupech a vstupech testovaného obvodu. Nebo dynamické parametrické testy, ty pracují s průběhem úrovně napětí na výstupech zařízení. Pomocí těchto zkoušek se dá odhalit porucha v obvodu, ale i odhadnout dispozice obvodu k selhání.

Dynamické testy zase zkoumají časování vstupních a výstupních signálů. Jako například zpoždění náběhu signálu z "0" na "1" a podobně. Další cenné informace může poskytnout takzvaný IDDQ test. Tento test je založen na měření proudu protékající obvodem v klidovém režimu, tedy v nějakém ustáleném stavu.

Dalším testem pro zkoušení elektronických zařízení může být i takzvané zahořování ("*burn-in test*"). V téhle zkoušce nejde o testování v pravém slovy smyslu. Podle vaničkové teorie poruch se předpokládá, že během doby životnosti se poruchy nejčastěji vyskytují těsně po okamžiku výroby, kdy se objevují výrobní vady a ke konci životnosti, kdy se na zařízení projevuje stárnutí. Snahou je co nejvíce zkrátit dobu výskytu výrobních vad, tedy zařízení se nechá pracovat za zvýšených teplot a napájené vyšším napětím, aby se všechny nedostatky projevily v nejkratším možném čase.[12]

V téhle práci jsem se zabýval pouze funkcionálním testováním, tedy testováním, které spočívá v zasílání příkazů danému obvodu, v tomto případě paměti, a čtením dat, které se objevují na jeho výstupu.

Pro výběr algoritmu zajišťujícího funkcionální testování paměti je nutno sledovat mnoho hledisek. Především zda se jedná o bitově či slovně orientovanou paměť. Dále také typ testované

paměti, který napovídá jaké závady v ní můžeme očekávat. V neposlední řadě je důležitá doba, kterou test zabere, popřípadě jakého stupně spolehlivosti při odhalení závad chceme dosáhnout. Obecně platí, že čím větší paměť a čím větší jistotu při odhalování chyb požadujeme, tím delší dobu průběh testu trvá. Pro detekci různých typů poruch je vhodný jiný algoritmus a je vhodné použít několik typů testů.

Test pro odhalení poruch se dá zařadit do některé ze tří skupin. Může se jednat o "*walk test*" - postupující jednička/nula, "*march test*" - pochodující jedničky/nuly a nebo o "*galop test*" - ubíhající jednička/nula.[13]

Test typu postupující jednička nebo postupující nula funguje tím způsobem, že cela paměť je naplněna jednou hodnotou (například nulami) a jediná paměťová buňka, která má jinou hodnotu je ta právě testovaná. O testování jednotlivých paměťových buněk mluvíme pouze u bitově orientovaných pamětí, u slovně orientovaných jednotlivé buňky zastupuje shluk několika buněk které se nedají samostatně adresovat (slovo).

Takový test může pracovat například následovně: celá paměť je vynulována, následně se z prvního paměťového místa přečte nula, zapíše jednička, přečte jednička a opět zapíše nula. Potom se adresa zvýší o jedničku a celý proces se opakuje dokud není otestována celá paměť. Pokud některá z operací čtení nevrátí očekávanou hodnotu detekuje se chyba. Tento typ testů může detekovat většinu jednoduchých chyb, ale selhává ve většině případů chyb závislých. Protože se celá paměť prochází pouze jednou a doba testování jedné buňky je konstantní, má časová složitost testu lineární charakter.

Pochodující jedničky nebo nuly pracují obdobně. Rozdíl je v tom, že paměťové místo je po otestování vyplněno opačnou hodnotu než v ní byla původně uložena. Například ve vynulované paměti se přes všechny buňky provádí postupně tyto operace: čtení nuly, zápis jedničky, čtení jedničky. Původně vynulovaná paměť se tak z jedné strany plní jedničkami. Při vhodné kombinaci prováděných operací a několika průchody pamětí je možno detekovat velké procento jednoduchých i závislých chyb. Časová složitost je opět lineární, protože doba trvání testu je násobkem počtu míst, které se samostatně testují. Testy jsou náročnější v tom, že pro dosažení spolehlivosti při detekci závislých chyb je nutno projít celou paměť několikrát.

Testy typu ubíhající jedničky nebo nuly jsou co do detekčních schopností nejlepší, ale za tuto výhodu platí velkou časovou náročností. Jejich princip spočívá v tom, že při testování každé buňky (při změně její hodnoty) otestuje ještě znovu celou okolní paměť. Tedy je přistupováno ke všem buňkám pokaždé když se inkrementuje adresa právě testované buňky.

Průběh testu pak může vypadat například takto. Na začátku je paměť vynulována, adresa testované buňky se nastaví na začátek paměti. Do testované buňky se zapíše jednička a následně je celá okolní paměť prověřena, že obsahuje pouze nuly. Do testované buňky se zapíše opět nula, inkrementuje se adresa testované buňky a celý proces se opakuje.

Je zřejmé, že se ve vzorci pro dobu trvání testu vyskytne druhá mocnina velikosti paměti, z tohoto důvodu je časová složitost kvadratická. To může být u větších pamětí nepřijatelné, proto se většinou používají různé optimalizace. Jedna z možností urychlení je netestovat při každém posunu adresy testované buňky celou okolní paměť ale pouze řádek a sloupec na kterém se testovaná buňka nachází. Nevýhodou tohoto přístupu je nutnost znát rozmístění paměťových buněk na čipu. Další z možností se jeví prověřit pouze bezprostřední okolí právě testované buňky, tedy 8 bezprostředně sousedících buněk a nebo lépe ještě k tomu 16 buněk sousedících "ob jednu buňku". Ve spojení s prvním způsobem, testováním řádků a sloupců, může být tato metoda velice efektivní i pro závislé chyby při zachování akceptovatelné časové náročnosti.

## 4.1 Detekce poruch vycházející z march testu

Nejznámější skupinou testů pro odhalování chyb v pamětech je march test a jeho varianty. Testování spočívá v konečných sekvencí operací prováděných nad každou buňkou. Tento sled se potom v závislosti na typu testu může opakovat pro všechny buňky v určeném pořadí.

### 4.1.1 Notace pro popis činnosti testu

Pro popis jednotlivých variant testu je zavedena notace. [1][6][8]

r - čtení z paměti

w - zápis do paměti

r0,r1 - čtení 0 resp. 1 z paměti

w0,w1 - zápis 0 resp.1 do paměti

↑ - zápis 1 do buňky kde byla 0

↓ - zápis 0 do buňky kde byla 1

↕ - inverze hodnoty

⇧ - provádění operace na buňky se zvyšující se adresou

⇩ - provádění operace na buňky s klesající adresou.

⇳ - na směru procházení nezáleží

→ - zápis 0 do buňky obsahující 0

🡪 - zápis 1 do buňky obsahující 1

### 4.1.2 Příklady variant testu

Například zápis části varianty testu march SS je:

⇳ (r0,r0,w0,r0,w1) ⇳ (r1,r1,w1,r1,w0)

Tento test z buňky přečte 0, potom opět přečte 0, následně zapíše 0, opět přečte 0 a nakonec zapíše 1. Po provedení této sekvence přejde na buňku s o jedna vyšší adresou a vše se opakuje. Potom provede znovu to samé, ale s inverzními hodnotami.. Předpokládá se, že paměť je před spuštěním testu vynulována. Pokud kterákoliv z operací selže, například místo očekávané 0 je přečtena 1, je to signálem, aby se buňka označila jako vadná, případně byla dále testována.

Tento test odhalí téměř všechny chyby izolované a také některé závislé. Například detekuje tyto chyby :

izolované: state fault, transition fault, read destructive fault, deceptive read destructive fault, incorrect read fault. Které chyby je tento test schopen odhalit.

závislé: Disturb coupling fault, write destructive coupling fault, write destructive coupling fault aj.

Některé další varianty march testu:

U každého testu uvádím dobu trvání, první vzorec značí skutečnou dobu trvání celého testu. Proměnná $t_r$ v něm zastupuje dobu čtení z paměťového místa, $t_w$ potom dobu zápisu. Druhý vzorec pro zjednodušení předpokládá, že čtení i zápis trvají stejnou dobu (čas t). Proměnná n zastupuje počet paměťových míst, které se testují. U bitově orientované paměti je to počet paměťových buněk. Do časů zápisu a čtení je nutno započítat prodlevu mezi dvěma operacemi.

MARCH SS: {⇳(w0); ⇳(r0,r0,w0,r0,w1); ⇳(r1,r1,w1,r1,w0 ); ⇩(r0,r0,w0,r0,w1);
⇩(r1,r1,w1,r1,w0);⇳ (r0)}

$9nt_w+13nt_r$ , 22nt

MATS+: {⇳(w0); ⇳(r0,w1); ⇩(r1,w0)}

$3nt_w+2nt_r$ , 5nt

March C-:{⇳(w0);⇳(r0,w1);⇳(r1,w0);⇩(r0,w1);⇩(r1,w0);⇳(r0)}

$5nt_w+5nt_r$ , 10nt

March B : {⇳(w0) ;⇳(r0,w1,r1,w0,r0,w1);⇳(r1,w0,w1); ⇩(r1,w0,w1,w0);⇩(r0,w1,w0)}

$11nt_w+6nt_r$ , 17nt

Pmovi: {⇩(w0);⇳(r0,w1,r1);⇳(r1,w0,r0);⇩(r0,w1,r1);⇩(r1,w0,r0)}

$5nt_w+8nt_r$ , 13nt

March U :{⇳(w0); ⇳ (r0,w1,r1,w0); ⇳(r0,w1);⇩(r1,w0,r0,w1);⇩(r1,w0)}

$7nt_w+6nt_r$ , 13nt

March LR { ⇳(w0); ⇩(r0,w1);⇧(r1,w0,r0,w1);⇧(r1,w0);⇧(r0,w1,r1,w0);⇧(r0)}

$7nt_w+7nt_r$ , 14nt

March SR {⇩(w0);⇧(r0,w1,r1,w0);⇧(r0,r0);⇧(w1);⇩(r1,w0,r0,w1);⇩(r1,r1);}

$6nt_w+8nt_r$ , 14nt

March G{⇳(w0); ⇧(r0,w1,r1,w0,r0,w1); ⇧(r1,w0,w1); ⇩(r1,w0,w1,w0); ⇩(r0,w1,w0); ⇧(r0,w1,r1); ⇧(r1,w0,r0); }

$13nt_w+10nt_r$ , 23nt

March Y {⇳(w0); ⇧(r0,w1,r1); ⇩(r1,w0,r0); ⇳(r0); }

$3nt_w+5nt_r$ , 8nt

March RAW {⇳(w0); ⇧(r0,w0,r0,w1,r1,w1); ⇧(r1,w1,r1,r1,w0,w0) ;⇩(r0,w0,r0,r0,w1,r1); ⇩(r1,w1,r1,r1,w0,r0); ⇳(r0);}

$7nt_w+7nt_r$ , 14nt

### 4.1.3 Problém detekovatelnosti poruch

Detekční schopnosti různých testů jsou odlišné. Test, který dobře detekuje jednu chybu, může u jiné naprosto selhat. Proto je vhodné při testování pamětí použít několik různých testů tak, aby pokud možno byla pokryta celá škála možných chyb. Pro výběr kombinace testů, který bude výsledný tester provádět, je důležité vědět, které chyby je ten který test schopen odhalit. V následujícím výčtu se pokusím nastínit vlastnosti, které musí tester splňovat, aby danou chybu dokázal rozpoznat.

| Chyba | způsob detekce |
|---|---|

Izolované chyby

| | |
|---|---|
| Stálá hodnota (SF) | Velmi jednoduchá chyba, dokáží ji detekovat testy, které čtou obě hodnoty “0“ a “1“ (r0,r1) |
| Chyba inverze (TF) | V test musí být invertující zápis (w1w0r0,w0w1r1) |
| Destruktivní zápis (WDF) | Je nutný neinvertující zápis (w1w1r1,w0w0r0) |
| Destruktivní čtení (RDF) | Stejně jako SF každé čtení vrací jinou hodnotu (r0,r1). |
| Maskované destruktivní čtení (DRDF) | Detekce vyžaduje dvě operace čtení za sebou (r0r0,r1r1). |
| Chybné čtení (IRF) | Stejně jako SF postačí jedna operace čtení (r0,r1). |

Závislé chyby

| | |
|---|---|
| závislá hodnota (CFst) | Test by měl obsahovat kombinace (w0,r0,w1r1) a projít paměť oběma směry. |
| Závislá destruktivní operace (CFds) | Po provedení zápisu/čtení na všech buňkách zkontrolovat celou paměť (r0,r1). |
| Závislá inverze (CFtr) | Test musí provést invertující zápis. Průchod oběma směry s různou zapisovanou hodnotou. |
| Závislý destruktivní zápis (CFwd) | Obdobně jako CFtr. Průchod oběma směry zaručí, že řídící buňka nabude hodnoty "0" i "1". |
| Závislé destruktivní čtení (CFrd) | Detekce snadná, čtení ve dvou průchodech, při každém průchodu, jiný stav okolních buněk. |
| Maskované závislé destruktivní čtení (CFdrd) | Test musí obsahovat dvě operace čtení za sebou, opět nutné dva průchody pro různé hodnoty řídící buňky. |
| Chybné závislé čtení (CFir) | Snadná detekce, (r0,r1). |

Tabulka 1. Schopnosti testů detekovat chybu daného typu

Při detekci závislých chyb je důležité, aby byl test symetrický. Při průchodu pamětí je důležité jednak je-li adresa řídící buňky nižší nebo vyšší než adresa buňky závislé, a jednak jestli k chybě dochází pokud řídící buňka obsahuje hodnotu "0" nebo "1". Testy tedy musí zohlednit všechny čtyři možné kombinace. Takovou symetrii splňují testy march C- a march SS, nesymetrické testy mohou některé chyby opomenout. Příkladem nesymetrie je test Mats+. Pokud test splňuje tuto symetrii, pak záleží na sledu operací v march modulech, které chyby budou detekovány a které ne.

V následujících kapitolách se budu věnovat detekčním vlastnostem jednotlivých testů podle tabulky č.1

#### 4.1.3.1 Statické nezávislé poruchy

- Stálá hodnota (SF) - Nejprůhlednější chyba, detekují ji všechny testy.
- Chyba inverze (TF) - Detekují prakticky všechny testy (selhává Mats+, protože mu chybí na konci testu modul čtení r0, kterým by chybu 1w0 detekoval. Ovšem pouze v případě, kdy nejde přepsat pouze určitá hodnota, např. 0w1 se zdaří ale 1w0 ne. Pokud selhává 0w1 a 1w0, Mats+ odhalí všechny takové chyby
- Destruktivní zápis (WDF) - Detekuje march SS (obsahuje neinvertující zápis).
- Destruktivní čtení (RDF) - Obdobně jako State fault, všechny testy ji detekují.
- Maskované destruktivní (DRDF) - Obdobně jako destruktivní zápis, detekuje ji march SS.
- Chybné čtení (IRF) - Obdobně jako stálá hodnota nebo destruktivní čtení, detekují ji všechny testy.

#### 4.1.3.2 Statické závislé poruchy

- Závislá hodnota (CFst) - Detekce je jednoduchá, přesto ji některé testy nezvládají. mats+ nedetekuje chybu v případě, když je při chybě 0 řídící adresa nižší a při chybě 1 vyšší. Detekuje tak pouze polovinu těchto chyb.
- Závislá destruktivní operace (CFds) - march C- a march SS zachytí všechny kombinace typu 1w0, 0w1, r0, r1. March C nezvládne 1w1 a 0w0, nesymetrický mats+ selhává úplně.
- Závislá inverze (CFtr) - mats+ zachytí v modulu 2 pouze 2 z 8 kombinací, další dva testy zachytí vše.
- Závislý destruktivní zápis (CFwd) - 0w0 a 1w1 zvládne pouze march SS, ostatní 2 testy jsou neúspěšné.
- Závislé destruktivní čtení (CFrd) - march SS i march C- detekují vše, mats+ detekuje jen polovinu kombinací.
- Maskované závislé destruktivní čtení (CFdrd) - tuto chybu detekuje pouze march SS.
- Chybné závislé čtení (CFir) - Mats+ zachytí jen polovinu kombinací. Ostatní dva testy ji detekují.

#### 4.1.3.3 Poruchy adresového dekodéru

Tyto chyby se většinou projevují velmi výrazně. Pro detekci všech chyb je nutné aby test prošel paměť v obou směrech s rozdílnou zapisovanou hodnotou. V případě že by jedna adresa ukazovala na dvě buňky nebo opačně jedna buňka měla dvě adresy, bude očekávaná hodnota jiná. Toto splňuje například march SS.

#### 4.1.3.4 Poruchy paměťové sběrnice

- Datová sběrnice - pokud bude vodič přerušen, bude řadič číst hodnotu vysoké impedance. Chová se tedy jako State fault a chyba je snadno detekovatelná. Naopak v případě spojení několika datových vodičů budeme ze všech číst nejnižší hodnotu kteréhokoliv z nich. Tedy pokud na všech bude vysoká úroveň (H), čteme H, ale pokud na kterékoliv bude nízká hodnota (L), všechny spojené vodiče budou přes ni uzemněny a čteme L. Chyba se dá nalézt testem, který zapisuje nějaký vzor, například pravidelné střídání 0 a 1. Dále může nastat, že některý bit je přímo uzemněn, popř. připojen na napájecí napětí. V tom případě se chová jako state fault.

- Adresová sběrnice - mohou nastat stejné případy jako u datové sběrnice, tedy přerušení bitu, spojení, a nebo spojení na zem nebo napájecí napětí. Chyby jsou většinou typu state fault a dají se snadno odhalit.

#### 4.1.3.5 Poruchy řídících signálů

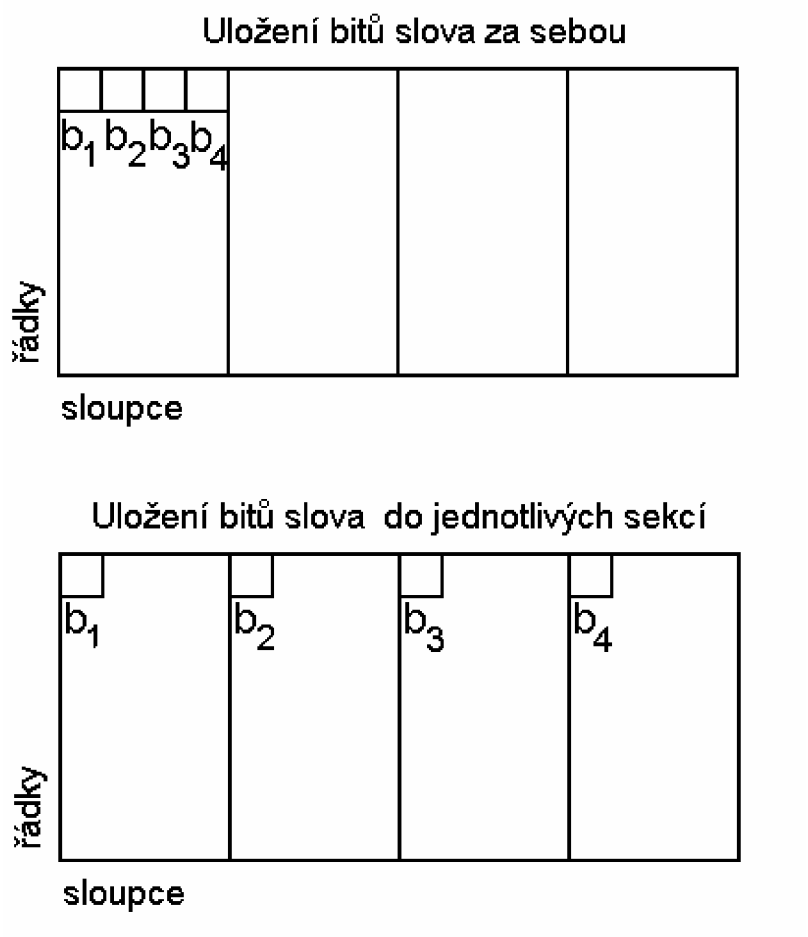
Pokud dochází ke špatnému přenosu řídících signálů, například v důsledku přerušení vodiče, nebo špatného časování, paměť většinou vůbec nereaguje, nebo se chová "dostatečně podivně", aby byla detekována chyba. Obzvlášť dynamické paměti jsou na časování signálů RAS a CAS velmi citlivé. Některé paměti mají pro tyhle stavy speciální chybový signál.

Všechny dosavadní úvahy spočívaly na předpokladu, že můžeme přistupovat k paměťovým buňkám samostatně a do nich zapisovat nebo číst bitové hodnoty. Pracovali jsme s bitově orientovanou pamětí.

#### 4.1.3.6 Poruchy ve slovně orientovaných pamětech

V případě, že je možno přenášet pouze slova o více bitech (slovně orientovaná paměť), je situace složitější. Závislé chyby se totiž mohou vyskytovat nejen v rámci jednoho slova (obě buňky závislé buňky náleží jednomu slovu), ale také každá v jiném slově. Tady záleží na tom, jak je slovo v paměti uloženo. Jednotlivé bity v slově spolu mohou, nebo také nemusí fyzicky sousedit, jak je znázorněno na obrázku č.13. Podle toho je nutné testovat jak chyby postihující dvě a více slov ("*inter word*"), tak i ty, které se omezí pouze na jedno slovo ("*intra word*"). [14]

V případě, že slova v paměti leží za sebou, chyby se testují march testem, který prověří z každého slova vždy jen dva bity a to ty, kterými spolu slova sousedí na řádku.




Obrázek 13. Rozložení jednotlivých bitů slova v paměti

Poruchy uvnitř slova(intra word) se dělí do čtyřech kategorií.

- Neomezené poruchy uvnitř slov ("*Unrestricted intra word faults*") - hodnota jedné buňky (buňky závislé) je závislá na jiné buňce (buňce řídící). Poloha této buňky ve slově je obecná, nemusí ležet vedle sebe.
- Omezené poruchy uvnitř slov ("*Restricted intra word faults*") - řídící buňka leží těsně vedle buňky závislé.
- Souběžné omezené poruchy uvnitř slov ("*Concurrent restricted intra word faults*") - v tomhle případě jsou 2 řídící buňky a leží vedle závislé buňky.
- Souběžné chyby uvnitř slov ("*Concurrent intra word faults*") - větší množství řídicích buněk na obecných pozicích uvnitř datového slova. Aby vznikla chyba, musí se změnit stav všech řídících buněk stejným směrem. například z "0" na "1".

Každá skupina se dá detekovat pomocí speciální sady vzorů (slov, které se zapisují a čtou na jednotlivé pozice), pomocí nichž lze chybu detekovat. Jedna z možností je také prověřit všechny možné kombinace bitů, což je ale velmi zdlouhavé a neefektivní, protože se mnoho posloupností opakuje. Algoritmus by mohl fungovat například takhle:

1. Celá paměť se zaplní jedním vzorem dat, např. pro délku slova 8 bitů vzor "01010101"
2. provádí se cyklus přes celou paměť od nejnižší adresy po nejvyšší
   - kontrola, zda se Byte nezměnil (je v něm uložen vzor)
   - zapíše do Bytu inverzní vzor
3. provádí se cyklus přes celou paměť v opačném směru od nejvyšší adresy po nejnižší
   - kontrola, zda se Byte nezměnil (je v něm uložena negace vzoru)
   - zapíšu původní vzor (inverzi inverze vzoru)
   -

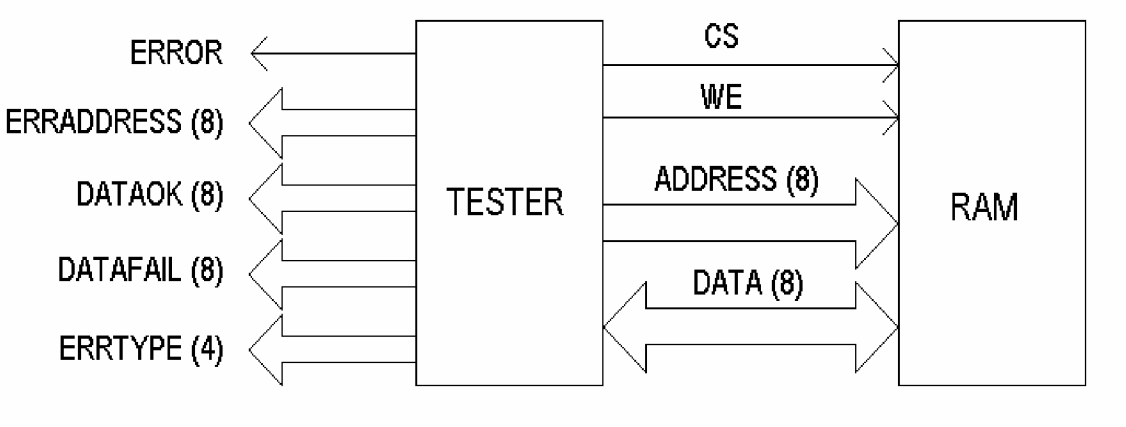
Existuje též specializovaný test, který lze algoritmem upravit pro danou paměť podle délky slova. Tento test je přímo určen k detekci intra word chyb.

# 5 Implementace

Celý systém sestává ze dvou částí, a to z vlastního testeru a paměti RAM, která je testována. Propojení komponent se realizuje pomocí vnějších portů. Tester generuje řídící signály pro paměť a na základě odezvy vyhodnocuje, zda je paměť vadná či ne. Testování probíhá porovnáním čtených dat s jejich očekávanou hodnotou.

## 5.1 Návrh testeru

Schéma propojení testeru s pamětí je na obrázku č.14. Na něm jsou také znázorněny signály, které slouží k propojení obou komponent. Slovem "signál" je zde myšlen propojovací vodič, definovaný v jazyce VHDL jako "signal". Názvy signálů zde použitých se shodují s těmi, které jsou požity v implementaci.




Obrázek 14. Schéma řadiče testu s pamětí

Zdrojový kód této komponenty je umístěn na přiloženém CD v adresáři "testy". Tento adresář obsahuje několik podadresářů, v každém z nich je zdrojový kód testeru, který implementuje test jednoho typu. Například tester který prověřuje paměť testem march B je v adresáři march_b.

Vzhledem k tomu, že se testery připojují k jedné paměti, mají jednotné rozhraní. Jakou ukázku implementace uvádím rozhraní testeru, připojení paměti k testeru a část kódu, který popisuje testování paměti.

Popis rozhraní, jednotlivé položky jsou vysvětleny v komentářích.

```
entity test is
port (
      error : out STD_LOGIC;
            -- indikuje vznik chyby, pokud je tato hodnota "0" tak test probíhá. V případě
            -- výskytu chyby se tato hodnota nastaví na "1".
```

```
        erraddress : out STD_LOGIC_VECTOR (0 to 7);
              -- adresa, na které vznikl nesoulad mezi čtenými a očekávanými daty (chyba).
        dataok : out STD_LOGIC_VECTOR (0 to 7);
              -- očekávaná data, jsou zde data, která by byla čtena pokud by byla paměť
              -- v pořádku.
        datafail : out STD_LOGIC_VECTOR (0 to 7);
              -- data, která byla přečtena a byla vyhodnoceny jako chybná.
        errtype: out STD_LOGIC_VECTOR (0 to 3)
              -- kód vzniklé chyby.
        );
  end test ;
  architecture arch_test of test is
```

Popis rozhraní komponenty ram, blíže v kapitole 5.2 Návrh paměti.

```
  component ram
    port(
        address : in STD_LOGIC_VECTOR (0 to 7);
              -- adresa paměti do které se přistupuje.
        we,cs : in std_logic;
              -- signály pro ovládání paměti určuje zda je povolen zápis. "1" - zápis "0"
              -- čtení. Signál cs  udává zda je daný paměťový modul vybrán.
        data : inout std_logic_vector (0 to 7)
              -- vstup-výstupní datová sběrnice, slouží k přenosu dat z a do paměti.
        );
    end component;
  begin
```

Vytvoření instance paměti, připojení signálů.

```
  ramI : ram port map (
            address => address,
            we => we,
            cs => cs,
            data => data
            );
  process
  begin
```

Inicializace řídících signálů, vynuluje výstupy a paměti sdělí že je vybrána a bude se do ní zapisovat.

```
error <='0';
erraddress <= "00000000";
errtype <= "0000";
cs <= '1';
we <= '1';
```

Počátek testování, v následujícím bloku se paměť vyplní nulami se vzrůstající hodnotou adresy 0 až 255. Čekání 5 ns je pro účely simulace. V zavedené notaci se jedná o operaci ⇧(w0).

```
data <= "00000000";
for i in 0 to 255 loop
        --cyklus přes celou paměť
    data <= "00000000";
    wait for 5 ns;
    address <= (conv_std_logic_vector(i,8));
        -- nastavení adresy, převod desítkového čísla na jeho binární reprezentaci
    wait for 5 ns;
end loop;
```

Test, zda byla předchozí operace (vyplnění paměti nulami) úspěšná. V případě že je detekována chyba jsou nastaveny výstupy error, erraddress a errtype. V zavedené notaci se jedná o operaci ⇧(r0).

```
for i in 0 to 255 loop
        --cyklus přes celou paměť, proměnná "i" se konvertuje do binárního tvaru.
    address <= (conv_std_logic_vector(i,8));
    wait for 10 ns;
    if (data/="00000000") then
        -- porovnání čtených dat a očekávaných ("00000000"). Pokud nesouhlasí,
        -- nastaví se výstupy
        error <='1';
        dataok <= "00000000";
        datafail <= data;
        erraddress <=(conv_std_logic_vector(i,8));
        errtype <= "0000";
        --v případě že nastala chyba (čtená data se liší od "00000000") je signálem
        -- error indikován vznik chyby, signálem dataok předána správná data,
        -- signálem datafail předána přečtená chybná data a pomocí errtype indikován
        -- typ chyby (v tomto případě "0000").
```

```
        else error <='0';
    end if;
```

Kód pokračuje dalšími operacemi nad pamětí, z hlediska principu testu již nepřináší nové informace, proto uvádím jenom tuto část.

Jednotlivé signály mají následující význam:

- Rozhraní testeru pro komunikaci s uživatelem
    - ERROR (1 bit) - informace o chybě. Pokud je aktivní, značí to, že tester právě zaznamenal závadu v paměti. V té době jsou také platná data na ostatních výstupních signálech testeru poskytující informace o vzniklé závadě. Hodnota error se nastavuje na "1" v případě, že se čtená data neshodují s očekávanými, tedy s těmi, které byly do paměti uloženy. Pokud je hodnota "0" test probíhá bez závad.
    - ERRADDRESS (8 bitů) - adresa paměti, na které byla nalezena chyba. V tomto příkladu se jedná o slovně orientovanou paměť, tudíž se jedná o adresu osmibitového slova. Adresování paměti je podrobněji popsáno dále v sekci o modelu paměti.
    - DATAOK (8 bitů) - hodnota, která byla do paměti původně zapsána, tedy taková která by se přečetla, pokud by paměť byla v pořádku. Pracuje se najednou s celými Byty.
    - DATAFAIL (8 bitů) - hodnota, která byla z paměti skutečně přečtena. Může se lišit od očekávané hodnoty v jednom až osmi bitech.
    - ERRTYPE (4 bity) - informace o operaci, při které byla přečtena chybná data. Kódy jednotlivých chyb jsou uvedeny v komentářích zdrojových souborů s testy. Liší se podle druhu testu. Pomocí této hodnoty a hodnotách dataok a datafail bývá obvykle možné určit, o jakou závadu paměti se jedná. Dekodér, který by podle těchto informací přesně určil množinu poruch které mohly tuto chybu zapříčinit, nebyl implementován a je předmětem dalšího vývoje.
- Rozhraní testeru pro komunikaci s pamětí
    - CS (1 bit) "*chip select*" - signál slouží pro výběr aktivní paměti. Pokud není "1" tak paměť nereaguje na žádné ostatní příkazy. V tomto případě, kdy se pracuje s jediným paměťovým modulem, je trvale nastavena 1.
    - WE (1 bit) "*write enable*" - povolení zápisu. Pokud je hodnota 1, je možno do paměti zapsat data. Naopak pokud we není nastaven, probíhá čtení.
    - ADDRESS (8 bitů) - Adresa Bytu, který se bude z paměti číst, nebo toho, do kterého se bude zapisovat.

- DATA (8 bitů) - Tento signál slouží jako obousměrná sběrnice pro přenos zapisovaných dat do paměti a čtených dat do testeru. Sběrnice nemá žádné řídící signály, směr přenosu se určuje pomocí signálu we.

Implementovaný tester nemá žádné vstupy pro ovládání uživatelem. Po zapnutí napájení začne probíhat test na připojené paměti. Při výskytu závady jsou nastavovány příslušné výstupní signály. Po doběhnutí testu začíná celý proces znovu od začátku.

Výběr druhu testu je realizován pomocí připojení patřičného testeru. Každý implementovaný test má vlastní komponentu (soubor s kódem ve VHDL). Propojení správného testeru s pamětí je záležitost simulátoru, ve kterém se modely simulují. Tester obsahuje paměť jako svoji komponentu a vytváří její instanci.

Při implementaci bylo nutné zejména důsledně ošetřovat práci s datovou sběrnicí. Signály na ní jsou řízeny ze dvou komponent a mohlo by tak dojít ke kolizím. Tester funguje zároveň jako řadič paměti a zajišťuje tak správné časování signálů.

Toto časování existuje pouze z důvodu porovnání doby trvání jednotlivých testů. Chyby, které vznikají jako důsledek nesprávného časování, nebyly uvažovány, tedy při návrhu paměti nebylo časování implementováno a paměť na řídící signály reaguje okamžitě. Časování, které tester používá, bylo zvoleno tak, aby řádově odpovídalo typu testované paměti.

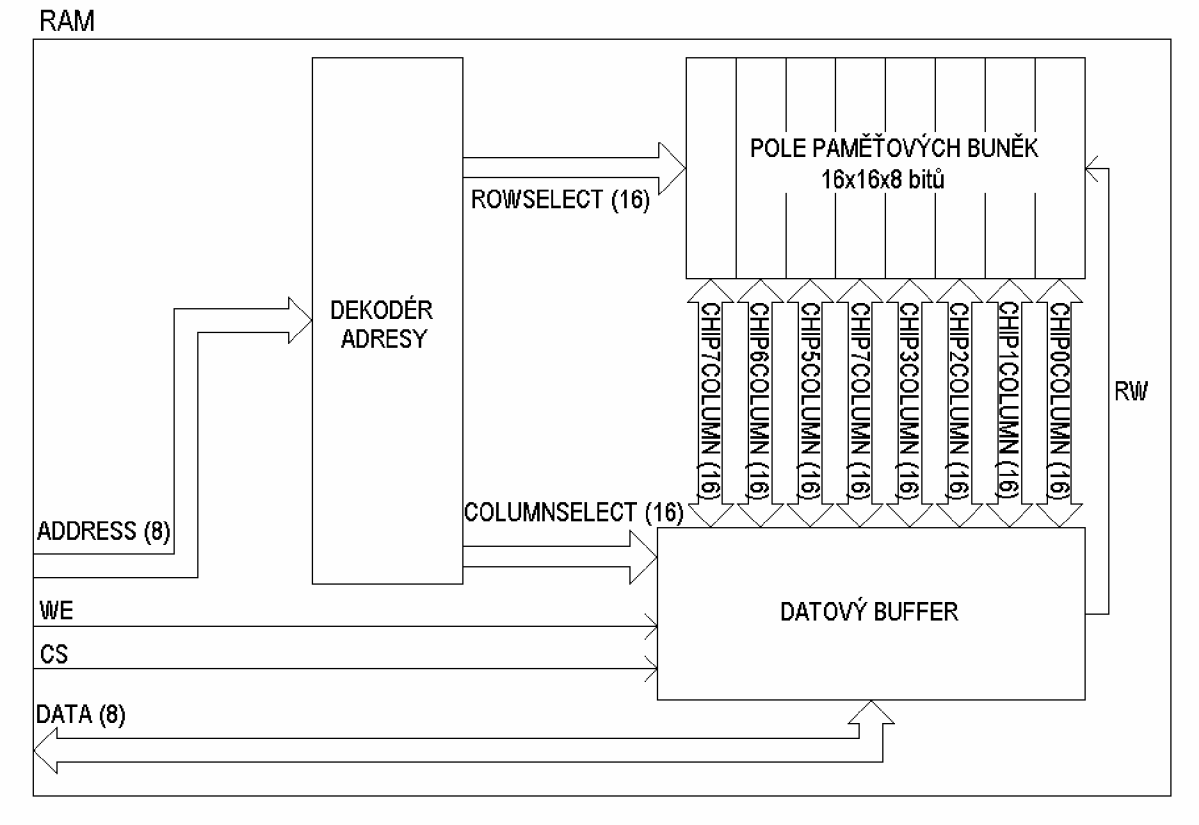
Díky tomu, že použitá paměť je statická a asynchronní, není nutné ani generovat hodinový signál. Dále pak není třeba řešit pravidelné obnovování zapsaných dat jako u dynamických pamětí. Do budoucna by bylo rozšíření o možnost připojení dynamických pamětí jistě zajímavé. Otevřelo by se mnoho možností testování chyb při špatném časování řídících signálů, protože dynamické paměti mají řízení podstatně složitější a obecně jsou na tyto typy chyb více náchylné.

## 5.2 Návrh paměti

K ověření funkce testeru a hlavně k určení schopnosti odhalit různé typy poruch bylo nutné vytvořit paměť, jejíž funkčnost by tester ověřoval. Tato paměť též musí být schopna poruchy simulovat.

Pro tuto úlohu jsem zvolil paměť statickou, asynchronní, slovně orientovanou. Její kapacita je 256 Byte. Tato velikost je pro testování detekce chyb dostačující a díky nevelkému počtu signálů je možné při simulaci detailně sledovat její činnost. Simulace je také přehlednější a průběh testu přes celou paměť netrvá příliš dlouho.

Záznamová oblast je organizována do polí paměťových buněk se šestnácti řádky a šestnácti sloupci. Délku uloženého slova jsem zvolil 8 bitů. Protože každý bit slova je uložen v jiném paměťovém poli, je těchto polí celkem 8, každé o velikosti 16 x 16 bitů. Paměťový modul se skládá z několika komponent, jejich uspořádání je patrné z obrázku č.15.




Obrázek 15. Struktura paměti

Samotná paměť je představována entitou RAM. Ke komunikaci s okolím jí slouží porty ADDRESS, WE, CS a DATA, které se uvnitř paměti mapují na stejnojmenné signály. Toto rozhraní již bylo dostatečně popsáno v kapitole o implementaci testeru.

Zdrojový kód paměti je na přiloženém CD v souboru ram.vhd. Tato komponenta plní v celém systému následující funkce. Poskytuje rozhraní pro připojení testeru, vytváří instance jednotlivých podkomponent paměti a propojuje jednotlivé vnitřní komponenty mezi sebou.

Uvádím zde ukázku zdrojového kódu popisující rozhraní paměti.

```
entity ram is
 port(
      address : in STD_LOGIC_VECTOR (0 to 7);     -- adresa
      we,cs : in std_logic;                       -- řídící signály
      data : inout std_logic_vector (0 to 7)      -- data
      );
end ram;
```

Proměnná “address“ složí k přenosu adresy slova s kterým se bude pracovat. Port s názvem “we“ určuje zda je do paměti povolen zápis, přepíná mezi režimem čtení a zápisu. Port “cs“ udává

zda bude paměť aktivní. A nakonec proměnná "data" reprezentuje obousměrnou datovou sběrnici přes níž se přenáší čtená a zapisovaná data.

Komponenta "ram"se skládá z několika vnořených podkomponent. Každá z nich je popsaná v samostatném vhd souboru.

**Pole paměťových buněk**

V této části probíhá samotné ukládání dat. Pole je tvořeno signály, na které se při zápisu připojuje log. "1" nebo "0". podle hodnoty, která se zapisuje. Při čtení se na základě těchto signálů nastaví odpovídající hodnota na výstupní sběrnici. Do komponenty vstupuje signál RW z datového střadače, který určuje, zda operace bude čtení nebo zápis. Adresování buněk v této komponentě probíhá pouze na základě signálu rowselect. Tedy se čte nebo zapisuje celý řádek (128 bitů) najednou.

Čtení probíhá v jednom kroku podle toho, na kterém bitu signálu rowselect je "1" se nastaví aktivní jeden řádek matice. Obsah tohoto řádku se potom nastaví na výstupní sběrnici realizovanou signály chip0column až chip7column.

Vzhledem k tomu, že není možné vybrat si z řádku pouze jedno slovo (8 bitů), musí zápis probíhat ve dvou krocích. V prvním kroku se přečte celý řádek, na kterém se nachází Byte, který se ukládá. Tento řádek se uloží do datového bufferu. V druhém kroku se v bufferu přepíšou ty bity, které pokrývá zapisovaný Byte novou hodnotou. Tuto operaci může buffer provést, protože zná adresu zapisovaného sloupce. Sdělí mu ji dekodér adresy pomocí signálu columnselect. Nakonec se změněný řádek zapíše do paměťové matice.

Pole buněk komunikuje s datovým bufferem a dekodérem adresy přes následující rozhraní.

```
entity mem_matrix is
 port (
       rw:in STD_LOGIC;                              -- přepínání zápis/čtení
       rowselect: in STD_LOGIC_VECTOR (0 to 15);
                                                     -- výběr řádku
       chip0column: inout STD_LOGIC_VECTOR (0 to 15);
       chip1column: inout STD_LOGIC_VECTOR (0 to 15);
       .
       .
       chip7column: inout STD_LOGIC_VECTOR (0 to 15)
                                                     -- data
       );
end mem_matrix;
```

Signálem “rw“ paměť rozlišuje jestli právě probíhá operace čtení či zápisu na řádek určený pomocí “rowselect“. Řádek je vybírán na základě informace z dekodéru adresy, pokud pracuje tento dekodér správně, tak je v signálu “rowselect“ aktivní právě jeden bit. Signály chip0column až chip7column představují sběrnice k jednotlivým oddílům pole buněk.

**Dekodér adresy**

Dekodér adresy slouží k rozdělení adresy Bytu na adresu sloupce a řádku, v kterém se požadovaná data nacházejí. V tomto případě, kdy je matice čtvercová, se adresa rozdělí rovnoměrně: spodní 4 bity - adresa sloupce, vrchní 4 bity - adresa řádku. Z dekodéru vede signál ke každému řádku matice, nastavením jednoho bitu tohoto signálu na 1 se příslušný řádek matice stane aktivním. Stejně tak se děje s adresou sloupce, jenomže tato adresa se přivádí do datového bufferu. Ve skutečnosti signál výběru sloupce nastaví jako aktivní osm sloupců v poli paměťových buněk, protože délka slova je osm bitů. Toto rozdělení se děje až v datovém bufferu.

Pro ilustraci uvádím popis rozhraní dekodéru. Vstupní adresa “address“ je brána jako dvě bezznaménková čtyřbitová binární čísla (vrchní a spodní 4 bity). Rozsah každého z nich je tedy 0 až 15. Při správné funkci dekodéru bude podle tohoto čísla na výstupu aktivní vždy pouze jeden bit, kterým bude určen jeden ze šestnácti řádků a sloupců v poli paměťových buněk.

```
entity address_decoder is
port (
       address: in STD_LOGIC_VECTOR (0 to 7);
              --vstup dresy, binární číslo
       rowselect: out STD_LOGIC_VECTOR (0 to 15);
              -- výběr aktivního řádku v poli buněk, pouze jeden bit je nastaven na “1“
       columnselect: out STD_LOGIC_VECTOR (0 to 15)
              -- výběr aktivního sloupce v poli buněk, pouze jeden bit je nastaven na “1“
      );
end address_decoder;
```

**Datový buffer**

V této části se dočasně uchovává hodnota čtených nebo zapisovaných dat. V případě zápisu zde navíc probíhá proces řízení zápisu, jak je nastíněno v popisu pole paměťových buněk.V případě skutečné paměti by zde byly také zesilovače signálu. Funkce tohoto modulu spočívá ve zprostředkování komunikace mezi dekodérem adresy, polem paměťových buněk, a vnějším rozhraním celého modulu. Z bufferu vede do pole buněk 128 datových vodičů, protože šířka úseku paměťového pole v kterém je uložen jeden bit slova je 16 a slovo má 8 bitů, tedy 16x8 bitů. Úkolem

bufferu je z jednoho řádku paměti (18 bitů) vybrat 8 bitů. které vytvoří slovo. Toto slovo se, podle toho zda se z paměti čte či zapisuje, buďto nastaví na výstupní sběrnici, nebo zapíše do pole paměťových buněk.

Činnost bufferu při zápisu probíhá následovně.

```
if we='1'then
```

Nejdříve je z paměti vyčten celý řádek, tedy pole buněk se nastaví na čtení.

```
rw<='0';
```

Do dočasného zásobníku jsou uložena přečtená data, pro zjednodušení uvádím komunikaci pouze s jedním oddílem pole paměťových buněk.

```
chip0buf:=chip0column;
```

Příslušný bit v zásobníku je přepsán hodnotou, kterou chceme zapsat do paměti, následně se pomocí signálu "rw" nastaví pole buněk do režimu pro zápis a data se zapíší.

```
chip0buf(n):=data(i);
chip0column <= chip0buf;
rw <= '1';
```

Samostatné čtení z paměti již bylo ukázáno jako součást procesu zapisování, nemá tedy cenu ho zde znovu uvádět.

### 5.2.1 Simulace chyb

Aby bylo možno ověřit funkci testeru, musí paměť simulovat závady, které v reálu mohou nastat.

Pomocí modifikace zdrojového kódu je možné vytvořit libovolnou chybu. V současnosti, jsou na přiložením CD zdrojové kódy pamětí v kterých jsou tyto chyby:

- Chyby adresového dekodéru
  - Více adres ukazuje na stejnou buňku
  - Některé adresy neukazují nikam (žádné paměťové buňky se nenastaví jako aktivní)

- Izolované chyby v poli paměťových buněk
  - Stálá hodnota ("*State Fault*") (SF)
  - Chyba inverze ("*Transition fault*") (TF)
  - Destruktivní zápis ("*Write disturb fault*") (WDF)
  - Destruktivní čtení ("*Read destructive fault*") (RDF)
  - Maskované destruktivní čtení ("*Deceptive read destructive fault*") (DRDF)

    - Chybné čtení (“*Incorrect read fault*“) (IRF)
- Závislé chyby v paměťových buněk
    - Závislá hodnota (“*State coupling fault*“) (CFst)
    - Závislá destruktivní operace (“*Disturb coupling fault*“) (CFds)
    - Závislá inverze (“*Transition coupling fault*“) (CFtr)
    - Závislý destruktivní zápis (“*Write destructive coupling fault*“) (CFwd)
    - Maskované závislé destruktivní čtení (“*Deceptive read destructive coupling fault*“) (CFdrd)
    - Chybné závislé čtení (“*Incorrect coupling fault*“) (CFir)

Zavedení chyb do paměti se provádí většinou v poli paměťových buněk, kde při čtení nebo zápisu se testuje adresa, na hodnotu adresy chyby a v případě shody se provede příslušná akce, v závislosti na typu chyby. Vytváření chyb probíhá přímo ve zdrojovém kódu. Místa, kde se tak děje, jsou okomentována. Na přiloženém CD jsou 4 soubory s vadným paměťovým polem. každé takové pole obsahuje závady tří různých typů.

Pro ilustraci uvádím zdrojové kódy několika chyb.

Vytváření izolovaných chyb je nejjednodušší, nemusí se testovat stav ostatních buněk. Například u chyby stálá hodnota (SF) stačí při čtení z postižené buňky vložit na výstupní sběrnici místo obsahu buňky stálou hodnotu, v tomhle příkladu “0“.

```
if (i=x) and (j=y) then chip0column(j)<='0'; end if;
```

Proměnné i a j představují čísla řádku a sloupce nad kterými se v paměti provádí operace, v x a y jsou potom uloženy souřadnice poruchy. Pokud se tyto hodnoty shodují, čte se z vadné buňky, připojí se na výstup chip0column “0“ bez ohledu na obsah buňky.

Generování chyby Chybné čtení (IRF) je také snadné. Změna spočívá pouze v tom, že na výstup se přiřadí opačná hodnota než je v paměťové buňce. V následujícím kódu vzniká chyba na sedmém bitu slova(chip7column) na řádku x a sloupci y. Blok kódu se provádí při čtení.

```
if (i=x) and (j=y) then
        -- pokud se aktivní řádek a sloupec (i, j) shodují s adresou chyby (x,y) je vrácena
        -- inverzní hodnota
        if (ram7(x)(y)='1') then
                -- ram7(x)(y) je sedmý bit slova na adrese x,y.
                chip7column(j)<='0';
                -- na výstup se zapisuje chybná hodnota
```

```
        else
              chip7column(j)<='1';
        end if;
```

Při vytváření závislých chyb je situace o něco složitější, je nutné kontrolovat nejen obsah závislé buňky u které vzniká chyba při čtení, ale i obsah buňky řídící. U chyby závislá hodnoty se při každé operaci s řídící buňkou, má adresu x,y provede následující kód.

```
if (ram1(x)(y)='1') then ram1(xz)(yz)<='1';
                    else ram1(xz)(yz)<='0'; end if;
```

Tím se zajistí, že závislá buňka, která má souřadnice xz,yz, bude mít vždy opačnou hodnotu než buňka řídící.

U chyby typu závislá inverze (CFtr), kdy dochází v závislosti na stavu řídící buňky k selhání invertujícího zápisu do buňky závislé, je nutné kvůli určení zda se jedná o invertující či neinvertující zápis uložit obsah závislé buňky před zápisem do pomocné proměnné. Vytvoření chyby pak vypadá takto.

```
variable tc_fault STD_LOGIC;
tc_fault:= ram7(xz)(yz);
             -- zavedení pomocné proměnné,  uloží se do ní původní hodnota závislé
             -- buňky (xz,yz) sedmý bit (ram7)
(.......)
             -- v tomhle místě se provede zápis dat do paměti
if (i=xz) and (j=yz) then
             -- porovnání jestli se zapisuje do vadné závislé buňky
              if (ram4(x)(y)='1') then
             -- pokud je v řídící buňce "1" invertující zápis se nezdaří, zůstane původní
             -- hodnota
                          ram7(xz)(yz)<=tc_fault;
             end if;
 end if;
```

Při zanášení chyb do adresového dekodéru existuje mnoho možností jak poruchy kombinovat. Je možné vytvářet poruchy pouze pro jednotlivé buňky, či pro celé řádky nebo sloupce. Poruchy, postihující celé řádky či sloupce jsou však obvyklejší. Implementace chyby, která při přístupu k jistému řádku tento řádek neaktivuje je následující.

V původním bezchybném dekodéru se nastavuje jeden bit výstupu na jedničku v závislosti na čísle přistupovaného sloupce prostým přiřazením.

```
columnselect(col_i) <= '1';
```

Proměnná "columnselect" je výstupem dekodéru má šířku 16 bitů. jednička se nastavuje na bitu, který je určen proměnnou "col_i". Pokud by však sloupec určený proměnnou x neměl být přístupný, musí se přiřazení podmínit.

```
if (col_i/=x) then columnselect(col_i)<='1'; end if;
```

V tomhle případě se jednička do výstupu nastaví pouze pokud není adresovaný sloupec roven x.

Obdobně se implementuje chyba, kdy dvě adresy řádku aktivují stejný řádek.

kód bezchybného přiřazení:

```
 rowselect(row_i) <= '1';
```

řádek s číslem x je adresován pomocí adresy x i y:

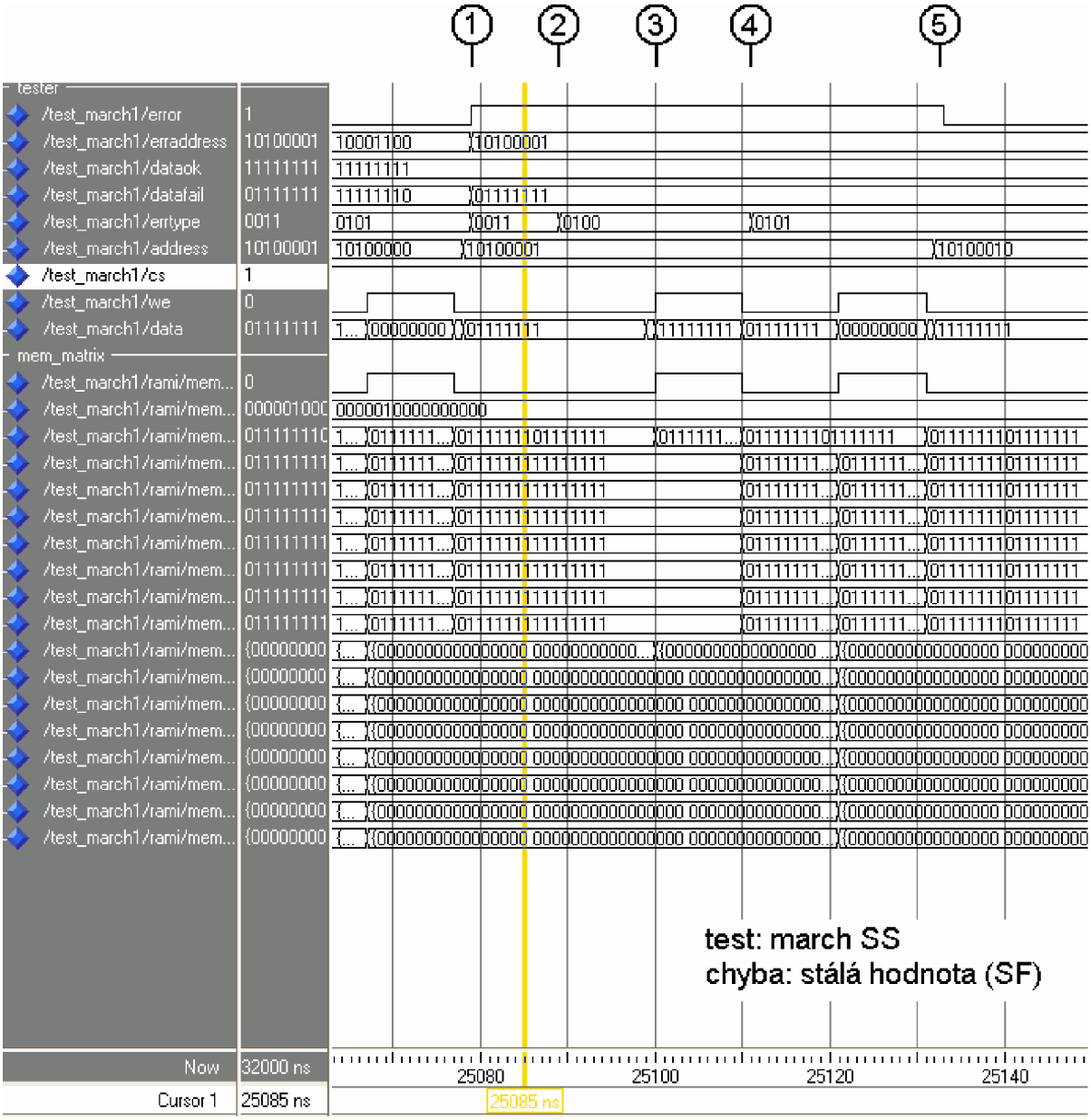
```
if (row_i=y) then row_i:=x; end if;
rowselect(row_i)<='1';
```

Proměnná "row_i" obsahuje číslo řádku a podle ní se nastavuje jednička v patřičném bitu výstupu "rowselect".

# 6 Ověření funkčnosti řešení

Ověření funkčnosti systému proběhlo nejdříve pomocí testovacích modulů. pomocí kterých byla prověřena funkčnost jednotlivých komponent, a následně již jako celek pomocí simulace celého systému. Jako ukázku ze simulací uvádím průběh signálů při testování pomocí testu march SS. Do testované paměti byla zanesena porucha stálá hodnota. Tato porucha zapříčiní že v prvním bitu slova na adrese "10100001" bude stále "0". Průběh signálů a detekce chyby je patrná na obrázku č.16. Ve výřezu simulace je tato chyba detekována pomocí této části testu march SS: ⇧(r1,r1,w1,r1,w0). V průběhu vzniku této chyby je několik významných okamžiků označených čísly nad grafem průběhu signálů.

- bod 1: march test provádí čtení jedničky, v buňce se závadou. V zápisu testu je to první r1, ⇧(**r1**,r1,w1,r1,w0). Tester nastaví signál "error" na "1" a tím indikuje vznik chyby, dále jsou na výstupu pomocí "erraddress", "datafail" a "dataok" předány adresa vadné buňky, přečtená chybná data a očekávaná správná data. V tohle příkladu to jsou: adresy chyby - "10100001", chybná data - "01111111" a očekávaná data "11111111". Tyto hodnoty jsou na simulaci vidět v levém sloupci, jsou vybrány ukazatelem simulátoru. Tester také nastaví na svůj výstup signálem "errtype" kód operace při které chyba vznikla, toto čtení má kód "0011"
- bod 2: Test zkusí přečíst danou buňku znovu, v zápisu testu druhé r1, ⇧(r1,**r1**,w1,r1,w0). Chyba stálá hodnota stále trvá, tak ji tester signálem "error" stále indikuje. Ale protože se již jedná o jinou operaci, změní se kód operace na "0100".
- bod 3: Pokročí se k další operaci a to k zápisu jedničky w1, ⇧(r1,r1,**w1**,r1,w0). I když se tento zápis nezdaří, není testerem žádná chyba detekována, signál "error" zůstává "1" protože tato adresa již byla označena jako vadná při předchozím čtení. Že se jedná o zápis lze poznat podle vysoké hodnoty signálu "we".
- bod 4: Následuje třetí čtení "1" r1, ⇧(r1,r1,w1,**r1**,w0). Opět je čtena chybná hodnota signál "error" zůstává na "1" a opět se mění kód operace při které tato chyba nastala. Následně dojde k zápisu "0" w0, ⇧(r1,r1,w1,r1,**w0**).
- bod 5: Blok testu na této paměťové buňce skončil, je inkrementována adresa, mění se signál "address". O buňce na nové adrese ještě není nic známo, proto se indikace chyby nastaví na "0" blok testu ⇧(r1,r1,w1,r1,w0) na ní začíná odznovu.



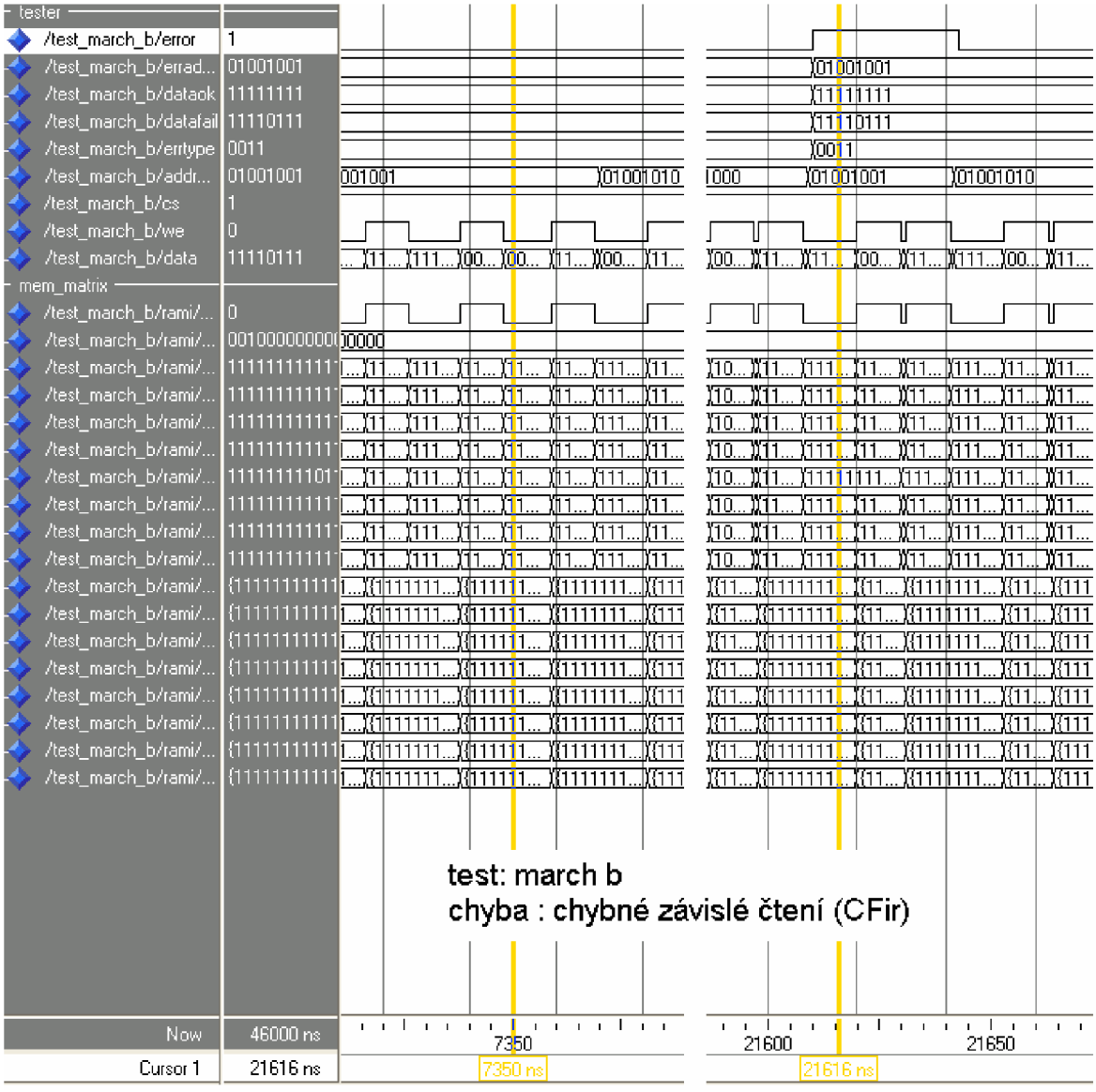

Obrázek 16. Ukázka simulace obvodu, kurzor je v oblasti kde je nastaven signál error na 1. detekuje se porucha

Zajímavější situace je při výskytu závislé poruchy, při ní obvykle hodnota v závislé buňce závisí jednak na operacích, které jsou nad ní prováděny, a jednak na stavu jiné buňky, buňky řídící. Typickým projevem takové poruchy je, že pokud je v řídící buňce nějaká hodnota, například “1“ chová se buňka korektně. Naopak pokud je v řídící buňce hodnota opačná, v tomhle příkladu “0“ dochází při čtení ze závislé buňky k chybám.

Při detekci takové chyby potom záleží, na vzájemné poloze řídící a závislé buňky. V případě symetrického testu je chyba detekována v polovině průchodů, v té druhé polovině má řídící buňka správnou hodnotu a poruchy se neprojeví. Toto je ukázáno na chybě typu chybné závislé čtení detekované testem march b.

Porucha byla implementování následujícím způsobem, buňka řídící má vyšší adresu než závislá, pokud je v řídící buňce uložena hodnota “1“ potom čtení ze závislé buňky vrací invertovanou hodnotu.

V testu march b se vyskytuje sekvence {⇳(w0); ⇧(r0,w1,r1,w0,r0,w1);⇧(r1,w0,w1);}. V prvním průchodu ⇳(w0) se paměť vyplní nulami, chyba se vyskytne pouze pokud je v řídící buňce “0“ nula, tudíž zatím je vše v pořádku. Následující sekvence ⇧(r0,w1,r1,w0,r0,w1) testuje buňky a plní paměť od nejnižší adresy k nejvyšší jedničkami. Protože však má řídící buňka vyšší adresu než závislá, tak v okamžiku kdy se testuje závislá buňka, v které by mohlo dojít k chybnému čtení, je v řídící buňce stále ještě “0“ a tudíž se závislá buňka chová korektně. tato situace je na obrázku č.17 vlevo, závislá buňka má adresu “01001001“. Po skončení tohoto cyklu je celá paměť vyplněna jedničkami. Při dalším průchodu testu blokem ⇧(r1,w0,w1) již je v řídící buňce uložena “1“, čtení z buňky závislé vrací inverzní hodnotu a je detekována chyba při čtení. Na obrázku č.17 vpravo.




Obrázek 17. Ukázka simulace obvodu, vlevo - Odhalení chyby při čtení, vpravo - průchod stejnou buňkou bez detekce chyby.

## 6.1 Typy testů

Pro testování pamětí jsou implementovány tyto march testy:

- March SS
- March B
- March Y
- Mats +
- Movi

A také test pomocí inverze vzorů, jak je popsán v kapitole o detekci poruch.

March testy nejsou pro testování slovně orientovaných pamětí příliš vhodné. Protože při čtení nebo zápisu více bitů najednou se zvyšuje množina vlivů na ostatní buňky (v případě závislých chyb). Pokud jsou však do paměti zaneseny chyby tak, aby v každém slově byla nejvýše jedna buňka, jejíž hodnota je ovlivněna nějakou chybou, a zároveň nejvýše jedna buňka která spouští jinou chybu, je možné bez újmy na obecnosti považovat tuto paměť za bitově orientovanou. V takovém případě by měly march testy uspět.

Pro ostatní případy je nutné použít algoritmus pro testování slovně orientovaných pamětí. Obvykle testování pomocí zápisu vzorů.

## 6.2 výsledky testů

Byly testovány všechny kombinace implementovaných poruch a testů. V každém průchodu byla k testeru připojena paměť a v ní zaneseno deset poruch jednoho typu. Počet odhalených poruch byl potom základem pro určení úspěšnosti testu. Úspěšnost jednotlivých testů spolu s dobou jednoho kompletního průchodu je v tabulce č.2.

| test / porucha | march SS | march Y | march B | Mats + | Movi | inv. Vzorů |
|---|---|---|---|---|---|---|
| stálá hodnota (SF) | 100% | 100% | 100% | 100% | 100% | 100% |
| chyba inverze (TF) | 100% | 100% | 100% | 100% | 100% | 100% |
| destruktivní zápis (WDF) | 100% | 0% | 0% | 0% | 0% | 0% |
| destruktivní čtemí (RDF) | 100% | 100% | 100% | 100% | 100% | 100% |
| maskované destruktivní čtení (DRDF) | 100% | 100% | 0% | 0% | 100% | 0% |
| chybné čtení (IRF) | 100% | 100% | 100% | 100% | 100% | 100% |
| závislá hodnota (CFst) | 100% | 100% | 100% | 50% | 100% | 100% |
| závislá destruktivní operace (CFds) | 100% | 100% | 100% | 100% | 100% | 100% |
| závislá inverze (CFtr) | 100% | 100% | 100% | 40% | 100% | 100% |
| závislý destruktivní zápis (CFwd) | 80% | 0% | 0% | 20% | 0% | 0% |
| maskované závislé destruktivní čtení (CFdrd) | 100% | 50% | 0% | 0% | 100% | 0% |
| chybné závislé čtení (CFir) | 100% | 100% | 100% | 100% | 100% | 100% |
| **Doba trvání testu (us)** | 56 | 23 | 48 | 14 | 37 | 14 |

Tabulka č.2. Úspěšnost testů při detekci různých typů poruch

Z výsledků je patrné, že žádný test neodhalil všechny poruchy, které byly uvažovány. Jako nejúspěšnější se zde jeví march SS spolu s movi. Velká úspěšnost při detekci je však vykoupena velkou časovou náročností. Nejrychlejší z testovaných je algoritmus mats+. Rychlost zde však je na úkor schopnosti detekovat poruchy. Do jisté míry je to způsobeno absencí symetrie jak bylo vysvětleno dříve.

Zároveň byly ověřeny výpočty časové složitosti jednotlivých testů. Při simulaci a při výpočtech bylo pro jednoduchost uvažováno, že operace čtení i zápisu slova do paměti trvá 11 ns. Teoretické doby trvání jednotlivých march testů byly uvedeny v kapitole 4.1.2. Pro test inverze vzorů je časová náročnost následující. Nejdříve se celá paměť naplní vzorem dat tedy za požití notace z kapitoly 4.1.2 se doba trvání této fáze určí jako součin $nt_w$ (n je velikost paměti, $t_w$ doba zápisu jednoho slova). Následně je provedena kontrola, zda zápis proběhl v pořádku a do paměti se zapisuje inverze původního vzoru. tato fáze trvá $nt_r + nt_w$. Poslední fáze je stejná jako předchozí, jenom probíhá opačným směrem. Trvá opět $nt_r + nt_w$ . Celková doba testu tedy bude $n(3t_w+2t_r)$. V tomto případě kde čtení a zápis trvá stejnou dobu 5nt. Vypočtené a změřené hodnoty jsou shrnuty v tabulce č.3.

| test | odhadovaná doba [us] | skutečná doba[us] |
|---|---|---|
| march SS | 56 | 56 |
| march Y | 23 | 23 |
| march B | 48 | 48 |
| mats + | 14 | 14 |
| movi | 37 | 37 |
| inverze vzorů | 14 | 14 |

tabulka č.3 srovnání doby trvání testů

Doby trvání testů byly získány ze simulace, tedy shoda mezi odhadovanou dobou a zjištěnou není zase tak překvapující.

Poruchy adresového dekodéru byly testovány na bezchybném poli paměťových buněk. Chyby, které vznikaly, byly natolik fatální, že je detekovaly bez problémů všechny testy ve všech případech.

# 7 Možnosti dalšího vývoje

Do budoucna by bylo vhodné zejména implementovat dekodér závad, který by na základě dat z testeru dokázal specifikovat typ závady paměti. Dále rozšířit model paměti o simulaci dynamických chyb, tedy takových chyb, které se objevují pouze pokud dojde k specifické posloupnosti čtení nebo zápisů. Dále by bylo vhodné rozšířit projekt v těchto oblastech.

- Možnost připojení jiných typů pamětí a implementovat testy, které se na tyto paměti obvykle aplikují. Například aby bylo možno testovat dynamické paměti, musel by tester navíc obsahovat obvody pro pravidelné obnovování dat. Dále by přibyly signály pro řízení přenosu adresy, zvlášť adresy řádku a sloupce.
- S možností připojený dynamických pamětí souvisí testování poruch při časování signálů. Existuje mnoho kombinací poruch, které jsou způsobeny nevhodným načasování signálů jako například nedostatečná doba platnosti adresy řádku nebo sloupce. Tyhle poruchy jsou pro dynamické paměti specifické a nelze je opomenout.
- Zavedení dalších poruch do pamětí. Jako například poruchy datové nebo adresové sběrnice, přerušení některých vodičů a podobně. I tyto poruchy se mohou v paměti vyskytnout a bylo by zajímavé zjistit schopnosti testů tyto poruchy detekovat.
- Pro automatizaci testování je nutné zajistit načítání konfigurace systému (paměti, testeru) ze souboru. V současné době se provádí výběr testu přímo v zdrojovém kódu. Stejně tak velikost testované paměti, typ a umístění chyb, které se v paměti vyskytují. Pro další experimentování by bylo vhodné, kdyby se všechny tyto parametry daly jednoduše měnit v konfiguračním souboru, bez nutnosti zásahu do zdrojového kódu a nového překladu celého projektu.
- Na základě experimentů nalézt lepší vzor dat pro testování slovně orientovaných pamětí. V případě, že je testována paměť u které známe vnitřní topologii. Tedy je známo, která adresa, odkazuje kterou paměťovou buňku (skupinu paměťových buněk) můžeme určit které buňky spolu fyzicky sousedí, tudíž se mohou vzájemně ovlivňovat. Na základě této informace by bylo možné najít optimální vzor dat, které se budou do paměti zapisovat, pomocí něhož půjde odhalit velké množství poruch.
- Vytvořit dekodér chyb. Tester pouze poskytuje informace o tom, že nastala chyba, na jaké adrese se to stalo, jaká data byla do paměti na tomhle místě uložena, jaká data byla skutečně přečtena, při jaké operaci k chybě došlo a jakým směrem se paměť zrovna procházela. Na základě těchto informací jo možné určit množinu typů chyb do které daná chyba patří. Někdy i jednoznačně určit o jakou chybu se jedná.

Navíc by bylo nutné implementovat testy, specializované na typické poruchy dynamických pamětí. Jako například, poruchy časování nebo nedostatečné obnovování. Implementace takové paměti by oproti statické měla několik Při implementaci tohoto druhu pamětí by bylo nutné uchovávat u každé paměťové buňky

## 7.1 Testování dynamických pamětí

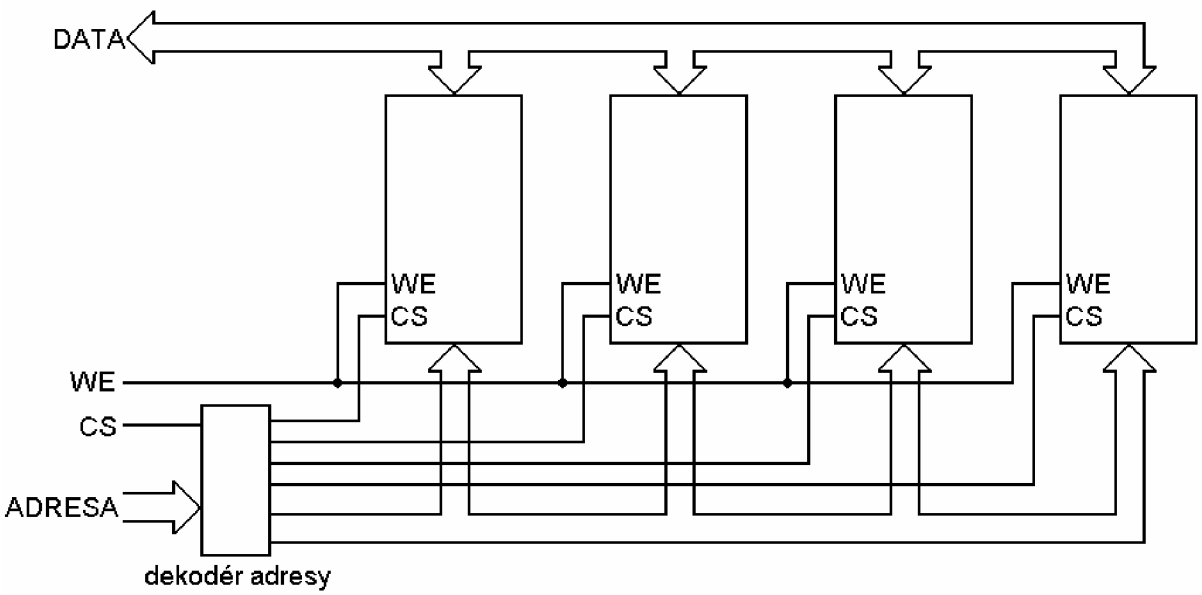
Pro rozšíření testeru o možnost testování dynamických pamětí by musel navíc obsahovat rozhraní pro komunikaci s paměťmi tohoto typu. To znamená implementovat zařízení, které by generovalo řídící signály a rozdělovalo adresu buňky nebo slova na adresu řádku a sloupce.

Model dynamické paměti by byl také složitější. Hlavním rozdílem by byla nutnost modelování ztráty dat samovybíjením. To znamená, že by se pro každou paměťovou buňku musela zaznamenat doba, kdy byl její obsah naposledy obnoven. Tato vlastnost, by se nejlépe implementovala pomocí dekrementace čítače řízeného hodinovým signálem. Pomocí něj by se také hezky simulovaly některé poruchy.

Paměť by musela být schopna dále simulovat poruchy nedostatečného obnovování, nebo poruchy časování, které jsou pro ni typické.[15] Z důvodu, že se dynamické paměti vyrábí obvykle ve velkých kapacitách, musel by se při návrhu testů brát větší zřetel na dobu trvání.

## 7.2 Zvýšení kapacity paměti

Pro přesnější odhad doby trvání testu je vhodné, aby testovaná paměť měla obdobnou velikost, jako moduly používané v praxi. Tím se jednak daný systém přiblíží realitě ale hlavně se sníží možná nepřesnost, která při přechodových jevech. Zvýšení kapacity paměti by bylo možné dosáhnout dvěma způsoby. Buďto zvýšením kapacity hotového paměťového modulu, nebo řazením několika modulů do většího bloku.[16] Návrh takového modulu je na obrázku č.18.




Obrázek 18. Návrh sloučení paměťových bloků do většího modulu

V takovéto paměti by byla slova uložena v jednotlivých blocích. Adresování jednotlivých bloků by probíhalo pomocí signálu "cs", který by generoval dekodér adresy, podle hodnoty vstupu adresy. Nový dekodér adresy by bylo možné vypustit, v tom případě by ale nebylo možná adresovat jednotlivé bloky a šířka slova by vzrostla tolikrát, kolik bloků paměti by paměťový modul obsahoval. V tomto příkladu (slovo 8 bitů, 4 bloky) by slovo narostlo na 32 bitů. Tím pádem by se musela také rozšířit vstup-výstupní datová sběrnice.

Úpravy testeru pro takovou paměť by spočívaly v rozšíření prostoru, který je schopen adresovat a také by se muselo upravit časování, v důsledku přidání nového dekodéru adresy. Co se testů týče, musela by se zvážit nová topologie paměti. V takovém modulu se vyskytnou nové kombinace adres, ukazující na paměťové buňky které spolu fyzicky sousedí.

# 8 Závěr

V dnešním trendu vysoké integrace, kdy se výrobci snaží dostat na omezenou plochu čipu co největší objem dat, nabývá testování pamětí na významu. Protože při snižování rozměrů se zvyšuje možnost vzniku přeslechů u vodičů, které jsou příliš blízko u sebe. Dále také čím více prvků daný obvod má, tím více jich navzdory vyspělé technologii výroby může selhat.

I přes vysokou spolehlivost současných paměťových modulů, není rozhodně možnost vzniku chyb zanedbatelným problémem. Obzvlášť u systémů od kterých požadujeme vysokou spolehlivost. Jednou z cest je použití vestavěných testů. Již nějakou dobu se prosazují ECC RAM (error correcting code) s možností detekce a opravy chyb za běhu. Daní za spolehlivost je ovšem pomalejší odezva a vyšší cena.

Testování pamětí má široké využití v mnoha oborech. Od výstupní kontroly při výrobě čipů před sestavením do paměťového modulu nebo zabudováním do vestavěného systému přes testování paměťových modulů před vložením do počítačového systému až po kontrolu spolehlivosti systémů, u kterých požadujeme vysokou dostupnost. Test paměti také může být prvním krokem při opravě nefunkčního zařízení. I když poslední dobou je externí testování vytlačováno vestavěnou diagnostikou.

Součástí práce bylo seznámit se s problematikou testování pamětí a navrhnout tester v jazyce VHDL. Proto bylo nutné pochopit princip funkce pamětí a také příčiny vzniku chyb v nich. Soustředil jsem se na obecný náhled na různé typy pamětí bez ohledu na nějakou konkrétní implementaci. To bylo nutné až při samotné konstrukci testeru. Výsledkem práce je modul, který za pomoci několika algoritmů dokáže odhalit závady, které mohou v paměti vzniknout. Součástí je také model paměti, na kterém jsem tyto závady simuloval a ověřoval jeho funkčnost. Tím jsem zároveň experimentálně ověřil schopnosti několika typů testů odhalit různé poruchy pamětí.

# Literatura


[1] Martinec, D.: Nové druhy pamětí RAM. VŠE v Praze
Dokument dostupný na URLhttp://sorry.vse.cz/~xmard16/vyukproj/pameti.htm(červenec 2007)

[2] Team autorů FITkit, vedoucí Dr.Ing. Fučík, O.: Dokumentace k FITkit-u
Dokument dostupný na URL http://merlin.fit.vutbr.cz/FITkit/ (červenec 2007)

[3] Merta, L.: Bakalářská práce - tester RAM
Brno, Vysoké učení technické 2005

[4] Xtronics : How memory works
Dokument dostupný na URL
http://xtronics.com/memory/how_memory-works.htm (červenec 2007)

[5] Micron technology: Various methods of DRAM refresh
Dokument dostupný na URL
http://download.micron.com/pdf/technotes/DT30.pdf (červenec 2007)

[6] Messmer, H-P., Dembowski, K.:velká kniha hardware. Brno, CP Books 2005

[7] Rosh, W.L.: Hardware bible. Indianopolis, Que publishing 1999

[8] Adams, R.D.:Hihg performance memory testing. London, Kluwer Academic Publishers 2003

[9] Lastra, A.: Digital logic and computer design
The university of North Carolina at chappel hill Dokument dostupný na URL
http://www.cs.unc.edu/~lastra/comp190/Notes/index.html (červenec 2007)

[10] Hornung, T.: Bakalářská práce - Návrh a implementace vestavěné diagnostiky ve VHDL.
Brno, Vysoké učení technické 2005

[11] Drábek, V. podklady k předmětu diagnostika a bezpečné systémy, VUT v Brně
Dokument dostupný na URL http://www.fit.vutbr.cz/study/courses/DIA/public/testram914.doc
(červenec 2007)

[12] Kolektiv autorů DocMemory.: Memory test background
Dokument dostupný na URL
http://www.ddrtester.com/page/news/showpubnews.asp?title=Memory+Test+Background&num=38
(červenec 2007)

[13] Haraszti, T.P.:Cmos memory circuits. London, Kluwer academic Publishers 2002

[14] Memtest86 - dokumentace
Domovská stránka http://www.memtest86.com/ (červenec 2007)

[15] Innoventions Inc. Stránky produktů
Domovská stránka http://www.memorytesters.com/ (červenec 2007)

[16] Cohen, B.:VHDL Coding styles and Methodologies, 2nd edition.
London, Kluwer Academic Publishers 1999

# Seznam příloh

# Příloha 1

Seznam souborů a složek na přiloženém CD

/readme.txt - úvod obsahující tento seznam

/zprava/zprava.pdf - Text této práce v pdf.

/zprava/zprava.doc - Text této práce v doc.

/zprava/obrazky/ - Adresář s obrázky použítými v textu.

/zdrojove kody - Adresář se zdrojovými kódy projektu

/zdrojove kody/navod.txt- Návod k simulaci projektu.

/zdrojove kody/vadny dekoder/address_decoder_bad.vhd - dekodér adres s vloženými poruchami

/zdrojove kody/vadne pameti/ - Složka s kódem paměťového pole do kterého byly úmyslně vloženy poruchy.

/zdrojove kody/vadne pameti/zkratky.txt - Seznam zkratek použitých v názvech adresářů.

/zdrojove kody/vadne pameti/SF_TF_WDF - Adresář s kódem paměťového pole do kterého byly úmyslně vloženy poruchy : state fault, transition fault a write disturb fault.

/zdrojove kody/vadne pameti/CFst_CFds_CRft - Adresář s kódem paměťového pole do kterého byly úmyslně vloženy poruchy : state coupling fault, disturb coupling fault a transition coupling fault.

/zdrojove kody/vadne pameti/RDF_DRDF_IRF - Adresář s kódem paměťového pole, do kterého byly úmyslně vloženy poruchy : read destructive fault, deceptive read destructive fault a incorrect read fault.

/zdrojove kody/vadne pameti/CFwd_CFdrd_CFir - Adresář s kódem paměťového pole, do kterého byly úmyslně vloženy poruchy : deceptive read destructive coupling fault, incorrect coupling fault a write destructive coupling fault.

/zdrojove kody/bez zavad/ - Adresář s kompletním systémem paměť - tester bez úmyslně vložených poruch. Při simulaci naplní paměť čísly 0 až 255 a přečte je.

/zdrojove kody/testbench/ - Adresář s ukázkou testovacích kódů pro některékomponenty paměti.