Универзитет у Београду
Електротехнички факултет

Дипломски рад

Пример дизајна кеш меморије засноване на МОЕСИФ протоколу

Ментор:
Др Саша Стојановић, доцент

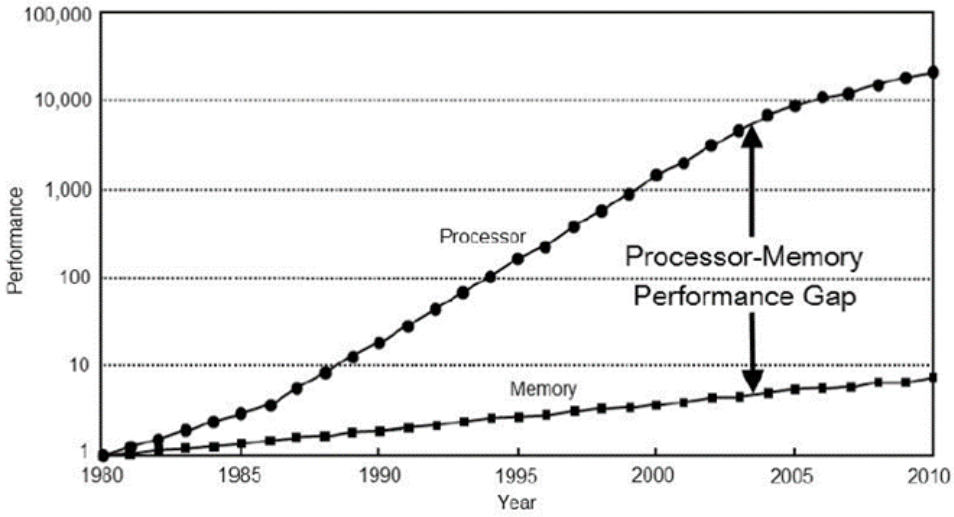
Кандидат:
Ђукић Јован 0047/2013

Београд, Септембар 2017.

САДРЖАЈ

1. Увод

У овом поглављу читалац ће се упознати са основном идејом овог рада. Прво ће бити представљене промене брзина рада процесора и меморије кроз историју рачунарства. Затим ће бити описане појаве услед којих је дошло до идеје кеш меморије и основни коцепти њеног рада. Потом ће бити описан проблем који се среће у данашњим процесорима са више језгара и начин на који је он решен. На крају ће бити дат садржај самог рада по поглављима.



Слика 1.1 перформансе процесора и меморије

Раст брзине процесора није праћен растом брзине меморије као што се види на слици 1.1. Кроз историју рачунарства јаз између процесора и меморије се све више повећавао. Тај проблем се зове „меморијски зид“. Због тога се дошло на идеју да се направи бржа меморија, кеш меморија. Међутим, због саме цене кеш меморије она је много мања и зато се мора користити на одређен начин.

У књизи *Computer Organization and Design: the Hardware/Software interface*[1] речено је да, уколико посматрамо адресе којима процесор приступа, можемо уочити две карактеристичне појаве: временска локалност и просторна локалност. Временска локалност, по дефиницији, каже да ако је процесор приступио одређеној меморијској локацији једном велика је вероватноћа да ће јој поново приступити у будућности. Најбољи пример овога је читање инструкција у петљи. Просторна локалност, по дефиницији, каже да ако је процесор приступио одређеној меморијској локацији да је велика вероватноћа да ће приступити суседним локацијама у будућности. Најбољи пример овога је приступ елементима низа.

Због ове две чињенице, кеш меморија ради тако што, приликом приступања процесора одређеној меморијској локацији, блок података коме припада тражени податак се копира у кеш меморију и затим се тражени податак достави процесору. Разлог копирања целог блока

података су горенаведене појаве. Међутим, у кеш меморији се у неком тренутку може наћи више блокова података, па је потребно водити евиденцију о томе који су блокови података пристуни у кеш меморији. Ово је постигнуто техникама пресликавања. Постоје три технике пресликавања: асоцијативна, директна и сет-асоцијативна.

Код асоцијативне технике пресликавања кеш меморија је подељена на улазе исте величине као блокови података оперативне меморије. Блок података оперативне меморије се може пресликати у било који улаз кеш меморије. Адреса траженог податка се дели на два дела: таг и офсет. Таг се користи као јединствени идентификатор блока података и помоћу њега се одређује да ли се он налази у кеш меморији или не. Офсет се користи за одабир одговарајуће речи из блока.

Код директне технике пресликавања кеш меморија је подељена на улазе исте величине као блокови података оперативне меморије. Међутим за разлику од асоцијативне технике пресликавања, блок оперативне меморије се може пресликати у само један улаз кеш меморије одређен адресом датог блока. Адреса траженог податка се дели на три дела: таг, индекс и офсет. Индекс одређује у који улаз кеш меморије се пресликава блок података из оперативне меморије. Таг и офсет имају исту функцију као кад технике асоцијативног пресликавања.

Код сет-асоцијативне технике пресликавања користи мешавина претходне две технике. Адреса траженог податка се дели на три дела: таг, индекс и офсет. Кеш меморија се састоји из одређеног броја скупова који је одређен ширином индекс дела адресе. Сваки скуп се састоји из одређеног броја улаза који је одређено сет-асоцијативношћу кеш меморије. Сваки блок података се пресликава у тачно одређен скуп који је одређен индекс делом адресе блока тако да се може рећи да је пресликавање на нивоу скупа директно. У оквиру скупа, блок података оперативне меморије се може пресликати у било који улаз скупа. Тако да је пресликавање у скупу асоцијативно. Таг и офсет имају исту функцију као кад технике асоцијативног пресликавања.

Када је кеш меморија пуна а у њој се не налази тражени блок подата, неки улаз кеш меморије се мора ослободити да би се направило места. Који улаз је у питању зависи од алгоритма замене која та кеш меморија користи. Такође, уколико је дати блок података у датом улазу „запрљан“, односно његов садржај се разликује од оног у оперативној меморији, садржај датог блока података у оперативној меморији се мора ажурирати да би у будућности приликом читања добијали коректан садржај.

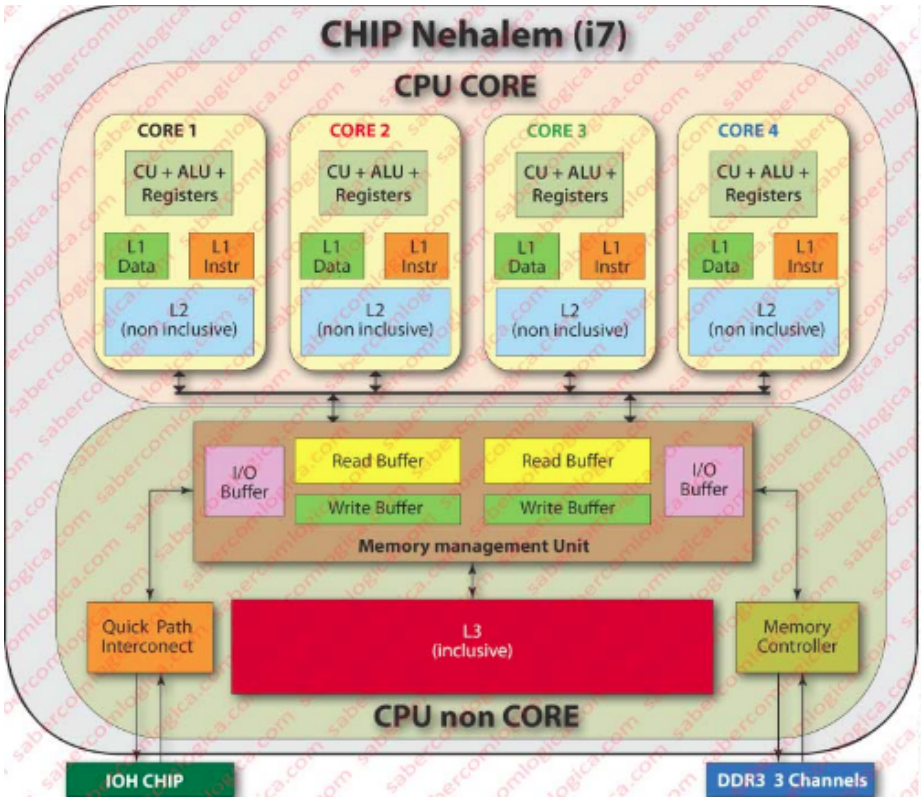
У процесорима са више језгра постоји више приватних кеш меморија. У најједноставнијем случају постоји један ниво кеш меморија, по једна кеш меморија за свако језгро. Из овог разлога може се доћи у ситуацију да постоје више копија истог блока података у приватним кеш меморијама са могућношћу уписа. Уколико нека кеш меморија упише неки податак у дати блок података, промена његовог садржаја неће бити видљива у осталим кеш меморијама. Међутим, да би се обезбедила коректност рада процесора свако језгро мора да види исти поредак операција уписа и читања као и све промене које се дешавају са садржајем меморије. Овакав меморијски систем се назива кохерентни меморијски систем и он се постиже протоколима за кеш кохеренцију. Постоје две групе оваквих протокола: протоколи засновани на „ослушкивању“ (енгл. *Snoopy-based*) и протоколи засновани на директоријумима (енгл. *Directory-based*).

У овом документу се описује једна од могућих имплементација кеш меморије у процесору са више језгара. Имплементација је одрађено у језику *SystemVerilog*, а тестирана је коришћењем *UVM* библиотеке компаније *Accelera*. Рад је састављен из 6 већих делова. Први део представља увод и у њему се читалац упознаје са идејом кеш меморије и проблемима

које се јављају у вишејезгарним процесорима и како се они решавају. Други део рада укратко описује кеш меморије са којима се можемо сусрести у данашњим процесорима. Трећи део рада описује дизајн саме кеш меморије. Такође, у овом делу ће бити описан сам протокол који обезбеђује кохерентни меморијски систем. Четврти део рада укратко описује *UVM* библиотеку компаније *Accelera* која је коришћења за верификацију дизајна и начин на који је она искоришћена. Пети део рада описује анализу перформанси имплементираног протокола за кеш кохеренцију. Шести део представља закључак самог рада.

2. Преглед модерних кеш меморија

Данашњи процесори садрже јако комплексне кеш меморије. Њихова комплексност се не огледа само у дизајну већ и њиховој организацији. У овом поглављу биће дат кратак преглед кеш меморија са којима се сусрећемо у данашњим процесорима. Конкретно, биће дата три примера и то: кеш меморија код *Intel Nehalem* архитектуре, кеш меморија код *AMD Bulldozer* архитектуре и кеш меморија код *ARM Cortex-A9 MPCore* процесора.

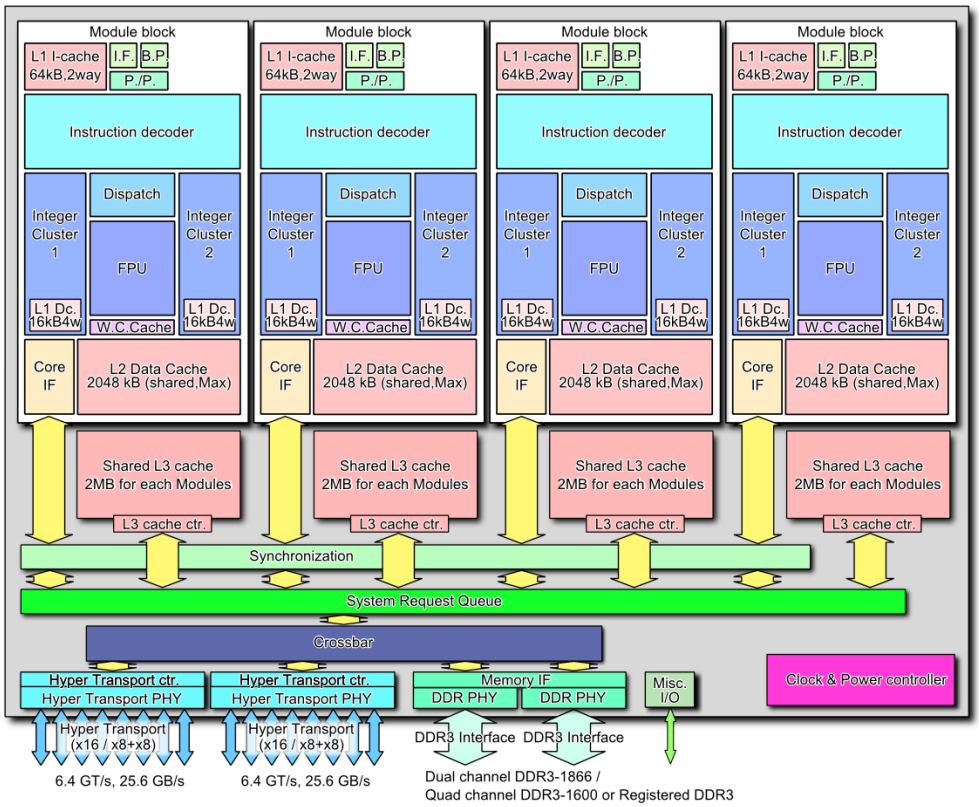


Слика 2.1 Блок дијаграм *Intel Nehalem* архитектуре[2]

На слици 2.1 дат је блок дијаграм *Intel Nehalem* архитектуре која се среће у *Intel Core i3*, *Intel Core i5* и *Intel Core i7* процесорима. Као што је наведено на интересантној веб страници[2], меморијска хијерархија код ове архитектуре се састоји из три нивоа кеш меморије. Први, *L1* ниво, се састоји из две мање кеш меморије. Једна кеш меморија служи зачување инструкција, док друга кеш меморија служи за чување података. Други, *L2* ниво, се састоји из једне, обједињене кеш меморије у којој се налазе и подаци и интрукције. Овај ниво није инклузиван, односно подаци нађени у њему не морају да представљају надскуп података из првог ниво. И први и други припадају такозваном *CPU CORE* делу чипа и приватни су, односно свако језгро поседује сопствени први и други ниво кеш меморије. Трећи, *L3* ниво, се састоји из једне веће кеш меморије која се налази на делу чипа који се зове *CPU non CORE* и који ради на другачијој фреквенцији. Кеш меморија на овом нивоу је дељена између језгара и

за разлику од првог и другог нивоа, овај ниво је инклузиван односно подаци у њему престављају надскуп података који се налазе у прва два нивоа.

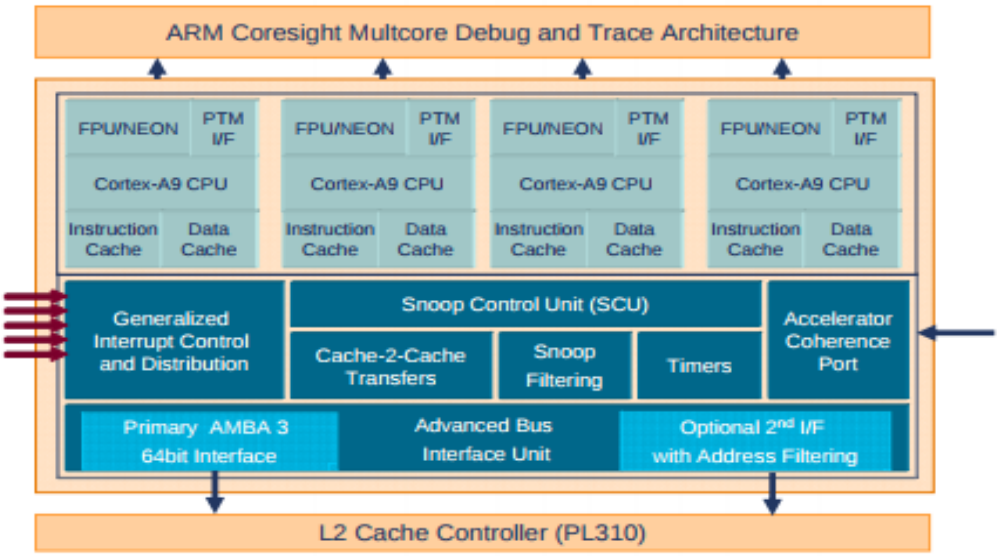
Према *Intel* приручнику за развој софтвера[3], протокол који се користи за одржавање кохерентног меморијског система је МЕСИ протокол. Један је од најпознатијих протокола данас. Сам протокол се спроводи на трећем нивоу меморијске хијерархије. Интересантна чињеница је да за сваки блок података који се налази у дељеној кеш меморији на трећем нивоу постоје бити инклузије. Ови бити носе информацију о томе да ли је дати блок података присутан у приватним кеш меморијама. Користе се да би се приликом извршавања неке акције протокола ажурирала стања копија датог блока података уколико се оне налазе у приватним кеш меморијама.



Слика 2.2 Блок дијаграм *AMD Bulldozer* архитектуре

На слици 2.2 дат је блок дијаграм *AMD Bulldozer* архитектуре. Према водичу за софтверску оптимизацију *AMD Bulldozer* архитектуре[4] постоје три нивоа меморијске хијерархије. Први, *L1* ниво, који се састоји из кеш меморије за инструкције и кеш меморије за податке. Други, *L2* ниво, се састоји из једне, обједињене кеш меморије у којој се налазе и подаци и инструкције. За разлику од *Intel Nehalem* архитектуре, овај ниво је инклузиван, односно подаци у другим нивоу представљају надскуп података у прво нивоу. Трећи, *L3* ниво се састоји из једне дељене кеш меморије подељене у банке тако да свако језгро поседује по једну банку. Овај ниво није инклузиван, већ представља *victim cache*. Наиме, алокација блокова података у трећем нивоу се дешава једино при избацивању блокова података из другог нивоа. Прилико приступа блоку података у трећем нивоу блок података се или оставља ту, уколико постоји вероватноћа да је дељен између језгара, или уклања и смешта у први ниво, уколико је велика вероватноћа да га користи само једно језгро.

У водичу за развој софтвера на *AMD64* архитектури[5] наведено је да се за одржавање кохерентног меморијског система користи МОЕСИ протокол. Представља надоградњу МЕСИ протокола. У МЕСИ протоколу при прелазу из модификованог у дељено стање морамо ажурирати оперативну меморију. Овим се непотребно троши пропусни опсег интерконекционе мреже јер ће можда поново доћи до уписа у дати блок. МОЕСИФ протокол уводи ново стање, стање власништва (енгл. *Owned*) које представља дељено и модификовано стање и у њега се прелази прилико читања модификованог блока из кеш меморије. Код МОЕСИФ протокола се прилико читања датог блока података не ажурира оперативна меморија, већ је то одговнорност кеш меморије која поседује копију блока података у стању власништва да прилико избацивања датог блока ажурира оперативну меморију.



Слика 2.3 Блок дијаграм *ARM Cortex-A9 MPCore* процесора[6]

На слици 2.3 приказан је блок дијаграм *ARM Cortex-A9 MPCore* процесора. Као што је наведено у документу о кохеренцији у *ARM* технологијама са више језгара[6], код овог процесора постоји два нивоа меморијске хијерархије. Први, *L1* ниво се састоји из две кеш меморије. Једна служи за чување података а друга служи за чување инструкција. Свако језгро поседује своје, приватне кеш меморије првог нивоа. Други, *L2* ниво се састоји из једне обједињене кеш меморије. За одржавање кохерентног меморијског система користи се МЕСИ протокол који модификован за трансфер података између кеш меморија. Сам протокол се спроводи у јединици означеној *SCU* која је поред тога одговорна за арбитрацију, трансфер податак између кеш меморија, комуникацију и сл.

Интересантан додатак јесте *Accelator Coherence Port* који представља обичан *AXI* порт. На њега се може прикључити било који уређај који може да генерише меморијске трансакције. Прилико генерисања трансакција, оне пролазе кроз логику за одржавање кохеренције и уколико се дата трансакција тиче податка који се налази у некој од кеш меморија предузимаје се одређене акције да би се одржао кохерентни меморијски систем. На овај начин могуће додати било који уређај који генерише меморијске трансакције а не води рачуна о кохерентном меморијско систему.

3. Опис дизајна

У овом поглављу дат је опис дизајна кеш меморија и начина комуникације између њих помоћу које се одржава кохерентан меморијски систем. Биће описан сваки део решења и то редом: алгоритам замене, јединица са директним пресликавањем, јединица са сет-асоцијативним пресликањем, арбитар, магистрала, контролер који опслужује захтеве језгра, контролер који надгледа магистралу, секвенцијална логика која спречава конфликте у кеш меморији, сам МОЕСИФ протокол помоћу којег се одржава кохерентни меморијски систем и меморијски систем.

3.1. *LRU* алгоритам замене

Алгоритам замене је сличан основном *LRU* агоритму. Постоји по алгоритам за сваки скуп. За сваки улаз из скупа постоји бројач који се поставља на нулу сваки пут кад се улазу приступи. Бројачи чије су вредности мање од вредности бројача који одговара улазу којем се тренутно приступа се инкрементирају. Улаз за замену је онај чији бројач има највећу вредност, односно чија се бинарна вредност састоји од свих јединица.

Једина разлика у односу на стандардни *LRU* алгоритам је та што сада поред контролера који опслужује захтеве језгра постоји и контролера који надгледа магистралу и који између осталог инвалидира неки улаз због уписа неког другог језгра у тај исти улаз. У овом случају је потребно ажурирати алгоритам тако да дати улаз пређе на врх листе слободних улаза. Сви бројачи чије су вредности веће од бројача који одговара улазу који се инвалидира декрементирају, а у бројач улаза који се инвалидира се уписује највећа вредност, односно бинарна вредност која се састоји из само јединица, тако да дати улаз долази на врх листе улаза за замену.

У случају да се приступ и инвалидација два блока која се налазе у истом скупу десе истовремено поступа се на следећи начин: сви бројачи чије су вредности мање од вредности два бројача који одговарају приступаном и инвалидираном улазу се инкрементирају, док се сви бројачи чије су вредности веће од вредности два бројача који одговарају приступаном и инвалидираном улазу декрементирају. Сви бројачи који се налазе између те две вредности остају непромењени. У бројач који одговара улазу којем се тренутно приступило се уписује нула, а у бројач који одговара тренутно инвалидираном улазу се уписује највећа вредност, односно бинарна вредност коју се састоји из само јединица. Истовремени приступ и инвалидација улаза није могућа, а начин на који је то обезбеђен биће објашњен нешто касније.

3.2. Јединица са директним пресликвањем

Ова јединица служи за чување свих података неопходних за рад кеш меморије. Састоји се из три меморије. Једна служи за чување самих блокова података, па је стога и сама подељена на улазе исте величине као блокови оперативне меморије. Друга меморија служи за чување тагова који служе за идентификацију блокова података. Последња, служи за чување стања у коме се блок података налази.

Свака од тих меморија садржи по два порта за читање, један за котролер који опслужује захтеве језгра и један за контролер који надгледа магистралу. Са портовима за упис је другачије. Контролер који опслужује захтеве језгра поседује по порт за упис за сваку од меморија из разлога што је његов посао не само опслуживање захтева већ и довлачење блока податак из оперативне меморије, инвалидација осталих копија блока података у другим кеш меморија и сл. Контролер који надгледа магистралу поседује један порт за писање и то за меморију за чување стања блокова. Разлог за ово је тај што је једно од задужења контролера који надгледа магистралу инвалидација блока података која се извршава тако што се његово стање поставља на невалидно, односно поставља се тако да се дати блок података више не налази у кеш меморији.

Такође постоје и два порта за погодак (енгл. *hit*). Један за контролер који опслужује захтеве језгра и један за контролер који надгледа магистралу. Контролер који опслужује језгро претражује кеш меморију за блоком података којем језгро жели да приступи. Контролер који надгледа магистралу претражује кеш меморију за блоком података чија се адреса тренутно налази на магистрали. Приликом претраживања кеш меморије, адреса блока којем се тренутно приступи, било са магистрале или од стране језгра, се дели на три дела: таг, индекс и офсет. Помоћу индекс дела адресе се одређује тачан улаз који се претражује, а помоћу таг дела и садржаја таг меморије и меморије која чува стања на позицији одређеној индекс делом адресе се одређује да ли се дати блок податка налази у кешу. Уколико се налази у кешу, одоговарајући сигнал поготка се подиже на логички ниво јединице и на порт за читање се избације вредност речи блока података одређеном офсет делом адресе уколико је захтев за читање, у супротном се у дату реч уписује уколико је захтев за упис.

3.3. Јединица са сет-асоцијативним пресликавањем

Јединица са сет-асоцијативним пресликавањем се састоји из више мањих јединица са директим пресликавањем. Сви улази са истим редним бројем чине један скуп. За сваки скуп постоји по један *LRU* алгоритам за замену. Број портова и њихови типови су исти као код мање јединице са директним пресликавањем.

Приликом приступа кеш меморији адреса се дели на три дела: таг, индекс и офсет. Помоћу индекс дела се одређује којем се скупу приступа, односно којим се улазима мањих кеш меморија приступа, јер сви они чине један скуп. Помоћу таг дела адресе и садржаја таг меморије и меморије која чува стање блока на позији одређеној индекс делом адресе се одређује да ли се дати блок налази у мањим кеш меморијама или не. Погодак у сет-асоцијативној јединици се добија као „или“ функција сигнала поготка у свим мањим јединицама. На основу ових сигнала се такође одређује у којој од мањих јединица се десио погодак. Уколико се ради о захтеву за читање, излаз мање јединице у којој се десио погодак се пропушта на портове за читање сет-асоцијативне јединице. Уколико се ради о захтеву за упис, контролни сигнали за упис и података који се уписује се пропуштају само до мање јединице у којој се десио погодак и упис се дешава само у тој јединици.

Прилико приступа сет-асоцијативној јединици, било од стране контролера који опслужује захтеве језгра или од контролера који надгледа магистралу, потребно је ажурирати *LRU* алгоритам. Обзиром на то да постоји по један алгоритам за сваки скуп, индекс део адресе одређујем који се алгоритам ажурира. На основу тога у којој се мањој јединици десио погодак се одређује број улаза у скупу у којем се налази тражени блок и он доводи на улаз алгоритма за замену. Контролни сигнали алгоритма за замену се пропуштају само до оног који одговара сету у којем се десио погодак.

3.4. Арбитар

Арбитар је јединица која одређује који од уређаја који имају потребу за магистралом (издали су захтеве) добијају право да користе магистралу. Уређаји постављају линије за захтев на логички ниво јединице када имају потребу за магистралом. Након што добију одобрење од арбитра, тако што арбитар подиже линију за одобрење на логички ниво јединице, обаве трансакцију на магитрали и након тога спусте линију за захтев на логички ниво нуле. Након овога арбитар додељује другим уређајима магистрлау уколико постоји захтева. Додељивање одобрења од стране арбитра иде по приоритетима, односну у тренутку када уређај који користи магистралу заврши са коришћењем, арбитар додељује магистралу оном уређају који је поставио линију захтев на логички ниво јединице који има највећи приоритет. Приоритет у овом раду представља редни број линије за захтев.

3.5. Магистрала

У систему постоји једна магистрала и она се састоји из следећих група линија: адресне линије, линије за излазне податке, линије за улазне податке, контролне линије помоћу којих се задаје захтев за читање или упис, контролана линија која назначава да је захтев опслужен, контролне линије које престављају тип трансакције која се тренутно дешава на магистрали, контролне линије којима контролери за надгледање магистрале назначавају да су блок података чија је адреса на магистрали инвалидирали и контролне линије које назначавају контролеру који опслужује захтеве језгра да је блок података у који се тренутно уписује инвалидиран у свим осталим кеш меморијама уколико је инвалидација потребна.

Првих пет група линија представљају део магистрале који служи за пренос података. Адресне линије служе за избацивање адреса блока. Линије за излазне податке служе за избацивање података који се уписују у оперативну меморију. Линије за улазне податке служе за избацивања података који су тренутно прочитани из оперативне меморије или неке друге кеш меморије. Контролне линије којим се задаје захтеве за читање се дижу на одговорајући логички ниво у зависности од тога да ли се ради о захтеву за читање или захтеву за упис. Контролна линија која назначава да је захев опслужен се подиже на логички ниво јединице када се захтев за читање или упис опслужи од стране оперативне меморије или неке друге кеш меморије.

Контролне линије које престављају тип тренутне трансакције на магистрали су линије које контролишу контролери који опслужују захтеве језгра. Они их поставаљају на једну од могућих вредности из скупа трансакција које се могу десити на магистрали. Тај скуп чине трансакција читања, трансакција инвалидације, трансакција ексклузивног читања и трансакција враћања блока података назад у оперативну меморију. Трансакција читања се извршава када се тражени блок података не налази у кеш меморији и потребно га је прочитати из оперативне меморије или неке друге кеш меморије. Трансакција инвалидације се извршава када се дати блок података налази у кеш меморији и језгро је издало захтев за упис али то није једина копија у систему, па је стога потребно инвалидирати све остале копије у систему. Трансакција ексклузивног читања је мешавина претходне две акције. Она се извршава када се тражени блок података не налази у кеш меморији а језгро је издало захтеве за упис, па се онда уједно и чита и све остале копије се инвалидирају. Трансакција враћања блока података назад се извршава када је блок података који се избацује из кеш меморије „запрљан“, односно његов садржај се разликује од оног у оперативној меморији и оперативна меморија се мора ажурирати. Контролери који надгледају магистралу посматрају која се трансакција дешава и понашају се на одговарајућу начин.

Контролне линије за инвалидацију се користе приликом акције инвалидације блока података. Служе контролерима који надгледају магистралу да обавесте контролер који опслужује захтеве језгра који је започео транскацију инвалидације да је копија у његовој кеш меморији инвалидирана и да може да настави са уписом. Контролер који опслужује захтеве језгра користи „и“ функцију ових линија да би утврдио да ли је су све копије инвалидиране.

3.6. Контролер који опслужује захтеве језгра

Контролер који опслужује захтеве језгра представља контролну логику чија је одговорност да на основу тренутног стања података у кеш меморији и на основу самог протокола изврши одређене радње и достави језгру тражени податак, уколико је у питању захтев за читање, или да упише података на тражену локацију уколико је у питању захтев за упис.

Без обзира који се инвалидирајући протокол за кеш кохеренцију користи, понашање контролера се може свести на следећи алгоритам:

```
if (hit) begin
  if (protocol.invalidateRequired) begin
    invalidateAllOtherCopies;
  end else begin
    doRequest;
  end
end else if (protocol.writeBackRequired) begin
  writeBackToMemory;
end else if (protocol.exclusiveReadRequired) begin
  exclusiveRead;
end else begin
  read;
end
```

Слика 4.6.1 Алгоритам рад контролера који опслужује захтеве језгра

На слици 4.6.1 приказан је алгоритам рада контролера који опслужује захтеве језгра. Ови кораци се извршавају у сваком такту. Пре било какве акције прво се проверава да ли је тражени блок података присутан у кеш меморији. Уколико није, прво се проверава да ли је садржај блока података који се тренутно налази у кеш меморији и који одабран за избацивање промењен, односно да ли је „запрљан“, и да ли треба ажурирату оперативну меморију. Уколико је то неопходно, контролер извршава трансакцију враћања блока података у оперативну меморију. Након тога, уколико је то потребно, излази се на магистралу са захтевом за читање. У зависност од тога да ли језгро жели да упише податак или да прочита података, на магистрали се извршава или трансакција ексклузивног читања или трансакција читања. Када се блок података довуче у кеш меморију или уколико је већ тамо, проверава се да ли је потребно инвалидирате све остале копије истог. Уколико треба, на магистрали се извршава трансакција инвалидације и инвалидирају се све остале копије. Уколико није потребно инвалидирати копије, захтев се опслужује, односно податак се доставља језгру или се уписује на тражену локацију.

```
task readBlock();
  case (readBlockState)
    READ_BLOCK_BUS_GRANT_WAIT: begin
      if (arbiterInterface.grant == 1) begin
        masterInterface.readEnabled <= 1;
        readBlockState              <= READ_BLOCK_WAITING_FOR_FUNCTION_COMPLETE;
      end
    end

    READ_BLOCK_WAITING_FOR_FUNCTION_COMPLETE: begin
      if (masterInterface.functionComplete == 1) begin
        cacheInterface.writeData <= 1;
        readBlockState           <= READ_BLOCK_WRITING_DATA_TO_CACHE;
      end
    end

    READ_BLOCK_WRITING_DATA_TO_CACHE: begin
      cacheInterface.writeData    <= 0;
      masterInterface.readEnabled <= 0;
      wordCounter                 <= wordCounter + 1;

      if ((& wordCounter) == 1) begin

        readBlockState <= READ_BLOCK_WAITING_FOR_DOWNGRADE;
      end else begin
        readBlockState <= READ_BLOCK_BUS_GRANT_WAIT;
      end
    end

    READ_BLOCK_WAITING_FOR_DOWNGRADE: begin
      cacheInterface.writeTag   <= 1;
      cacheInterface.writeState <= 1;
      readBlockState            <= READ_BLOCK_WRITING_TAG_AND_STATE_TO_CACHE;
    end

    READ_BLOCK_WRITING_TAG_AND_STATE_TO_CACHE: begin
      cacheInterface.writeTag   <= 0;
      cacheInterface.writeState <= 0;

      readBlockState  <= READ_BLOCK_BUS_GRANT_WAIT;
    end
  endcase
endtask : readBlock
```

Слика 4.6.2 протокол за извршавање трансакције читања

На слици 4.6.2 је приказан протокол по по коме се извршава трансакција читања. Иако сам контролер није имплементиран као машина стања (енгл. *State Machine*), сама трансакција се извршава на тај начин. Најпре се чека на дозволу за коришћење магистрале од стране арбитра. Када се дата дозвола добије, контролне линије које назначавају да се ради о захтеву за читање се подижу на логички ниво јединице. Када нам оперативна меморија или нека друга кеш меморија одговори на захтев за читање, податак који се налази на линијама за улазне податке се уписује на одговарајуће место у кеш меморији и прелази се на следећи. Након читања целог блока података, у кеш меморију се на одговарајућој позицији уписује таг и стање новог блока података прописано коришћеним протоколом за кеш кохеренцију.

```
task invalidateBlock();
  case (invalidateBlockState)
    INVALIDATE_BLOCK_WAITING_FOR_INVALIDATE_ACKNOWLEDGEMENTS: begin
      if (arbiterInterface.grant == 1) begin
        if (commandInterface.isInvalidated == 1) begin
          cacheInterface.writeState <= 1;
          invalidateBlockState      <= INVALIDATE_BLOCK_WRITING_STATE;
        end
      end
    end

    INVALIDATE_BLOCK_WRITING_STATE: begin
      cacheInterface.writeState <= 0;
      invalidateBlockState      <= INVALIDATE_BLOCK_WAITING_FOR_INVALIDATE_ACKNOWLEDGEMENTS;
    end
  endcase
endtask : invalidateBlock
```

Слика 4.6.3 протокол по коме се извршава транскација инвалидације

На слици 4.6.3 је приказан протокол по коме се извршава транскација инвалидације. Након што контролер добије одбрење за коришћење магистрале од арбитра, на магистрали се извршава трансакција инвалидације. Када контролер уочи да су све копије инвалидиране, уписује ново стање, које је прописано коришћеним протоколом за кеш кохеренцију, и након тога наставља са уписом.

```
task writeBackBlock();
  case (writeBackBlockState)
    WRITE_BACK_BLOCK_BUS_GRANT_WAIT: begin
      if (arbiterInterface.grant == 1) begin
        masterInterface.writeEnabled <= 1;
        writeBackBlockState          <= WRITE_BACK_BLOCK_WAITING_FOR_FUNCTION_COMPLETE;
      end
    end

    WRITE_BACK_BLOCK_WAITING_FOR_FUNCTION_COMPLETE: begin
      if (masterInterface.functionComplete == 1) begin
        masterInterface.writeEnabled <= 0;
        wordCounter                  <= wordCounter + 1;

        if ((& wordCounter) == 1) begin
          cacheInterface.writeState <= 1;

          writeBackBlockState <= WRITE_BACK_BLOCK_WRITING_STATE_TO_CACHE;
        end else begin
          writeBackBlockState <= WRITE_BACK_BLOCK_BUS_GRANT_WAIT;
        end
      end
    end

    WRITE_BACK_BLOCK_WRITING_STATE_TO_CACHE: begin
      cacheInterface.writeState <= 0;
      writeBackBlockState       <= WRITE_BACK_BLOCK_BUS_GRANT_WAIT;
    end
  endcase
endtask : writeBackBlock;
```

Слика 4.6.4 протокол по коме се извршава трансакција враћања блока података

На слици 4.6.4 приказан је протокол по коме се извршава трансакција враћања блока податак, односно ажурирање оперативне меморије. Као што се види, овај протокол је скоро идентичан протоколу за извршавање трансакције читања. Једина разлика је што се овде издају захтеви за упис уместо захтева за читање и што се по завршетку уписа целог блока

података у оперативну меморију у меморији која чува стања на позицији ослобођеног улаза уписује невалидно стање.

```
task readExclusiveBlock();
  case (readExclusiveBlockState)
    READ_EXCLUSIVE_BLOCK_BUS_GRANT_WAIT: begin
      if (arbiterInterface.grant == 1) begin
        masterInterface.readEnabled <= 1;
        readExclusiveBlockState     <= READ_EXCLUSIVE_BLOCK_WAITING_FOR_FUNCTION_COMPLETE;
      end
    end

    READ_EXCLUSIVE_BLOCK_WAITING_FOR_FUNCTION_COMPLETE: begin
      if (masterInterface.functionComplete == 1) begin
        cacheInterface.writeData <= 1;
        readExclusiveBlockState  <= READ_EXCLUSIVE_BLOCK_WRITING_DATA_TO_CACHE;
      end
    end

    READ_EXCLUSIVE_BLOCK_WRITING_DATA_TO_CACHE: begin
      cacheInterface.writeData    <= 0;
      masterInterface.readEnabled <= 0;
      wordCounter                 <= wordCounter + 1;

      if ((& wordCounter) == 1) begin
        readExclusiveBlockState <= READ_EXCLUSIVE_BLOCK_WAITING_FOR_INVALIDATE;
      end else begin
        readExclusiveBlockState <= READ_EXCLUSIVE_BLOCK_BUS_GRANT_WAIT;
      end
    end

    READ_EXCLUSIVE_BLOCK_WAITING_FOR_INVALIDATE: begin
      if (commandInterface.isInvalidated == 1) begin
        cacheInterface.writeTag   <= 1;
        cacheInterface.writeState <= 1;

        readExclusiveBlockState <= READ_EXCLUSIVE_BLOCK_WRITING_TAG_AND_STATE_TO_CACHE;
      end
    end

    READ_EXCLUSIVE_BLOCK_WRITING_TAG_AND_STATE_TO_CACHE: begin
      cacheInterface.writeTag   <= 0;
      cacheInterface.writeState <= 0;

      readExclusiveBlockState  <= READ_EXCLUSIVE_BLOCK_BUS_GRANT_WAIT;
    end
  endcase
endtask : readExclusiveBlock
```

Слика 4.6.5 протокол по коме се извршава трансакција ексклузивног читања

На слици 4.6.5 је приказан протокол по коме се извршава трансакција ексклузивног читања. Протокол је скоро идентичан протоколу за извршавање трансакције читања. Једина разлика је пре него што упишемо таг и ново стање на одговарајућу позицију, морамо сачекати да све остале кеш меморије инвалидарију копије блока података који се тренутно читао.

3.7. Контролер који надгледа магистралу

Контролер који надгледа магистралу представља контролну логику чија је одговорност извршава инвалидацију блокова података и доставља садржај истих уколико се они налазе у „власништу“ његове кеш меморије. Да ли је кеш меморија „власник“ датог блока података или не је прописано коришћеним протоколом за кеш кохеренцију.

Као и код контролера који опслужује захтеве језгра, без обзира који се инвалидирајући протокол за кеш кохеренцију користи, понашање овог контролера се може свести на следећи алгоритам:

```
case (busAction)
  BUS_READ: begin
    if (hit) begin
      if (owned) begin
        supplyData;
      end
      if (blockState != protocol.nextState) begin
        writeNextState;
      end
    end
  end

  BUS_INVALIDATE: begin
    if (hit) begin
      invalidateBlock;
    end
  end

  BUS_READ_EXCLUSIVE: begin
    if (hit) begin
      if (owned) begin
        supplyData;
      end
      invalidateBlock;
    end
  end
endcase
```

Слика 4.7.1 Алгоритам рада контролере који надгледа магистралу

На слици 4.7.1 приказан је алгоритам рада контролера који надгледа магистралу. Ови кораци се извршавају у сваком такту. Уколико се тренутно на магистрали извршава трансакција читања и блок података се не налази у кеш меморији не предузимају се икакве акције. Уколико се налази у кеш меморији и уколико је дата кеш меморија „власник“ истог, подаци се достављају кеш меморијиа која је започела трансакцију читања. Уколико се блок података налази у кеш меморији, без обзира да ли је кеш меморија „власник“ или не, стање истог се ажурира уколико је то потребно. Уколико се тренутно на магистрали извршава трансакција инвалидације и дати блок података се налази у кеш меморија потребно га је инвалидарита и јавити кеш меморији која је започела трансакцију инвалидације да је исти инвалидиран. Уколико се не налази у кеш меморији не предузимају се икакве акције. Уколико се тренутно на магистрали извршава трансакција ексклузивног читања и блок података се не налази у кеш меморији не предузимају се икакве акције. Уколико се налази у кеш меморији и уколико је дата кеш меморија „власник“ истог, подаци се достављају кеш меморији која је започела трансакцију ексклузивног читања. Уколико се блок података налази у кеш меморији, без обзира да ли је кеш меморија „власник“ или не, потребно га је инвалидирати и јавити контролер започео трансакцију да је исти инвалидиран.

Имплементација контролера који надгледа магистралу једноставнија од имплементације контролера који опслужује захтеве језгра и приказана је на слици 4.7.2.

```
always_ff @(posedge clock, reset) begin
  slaveInterface.functionComplete <= 0;
  cacheInterface.writeState       <= 0;
  invalidateEnable                <= 0;
  case (commandInterface.commandIn)
    BUS_READ: begin
      if (cacheInterface.hit == 1) begin
        if (arbiterInterface.grant == 0) begin
          if (protocolInterface.stateIn != cacheInterface.stateOut) begin
            cacheInterface.writeState <= 1;
          end
        end else begin
          if (slaveInterface.readEnabled == 1) begin
            slaveInterface.functionComplete <= 1;
          end else if (slaveInterface.functionComplete == 1 && slaveInterface.address[OFFSET_WIDTH - 1 : 0] == 0) begin
            cacheInterface.writeState <= 1;
          end
        end
      end
    end

    BUS_INVALIDATE: begin
      if (commandInterface.isInvalidated == 0) begin
        cacheInterface.writeState <= 1;
        invalidateEnable          <= 1;
      end
    end

    BUS_READ_EXCLUSIVE: begin
      if (cacheInterface.hit == 1) begin
        if (arbiterInterface.grant == 0) begin
          cacheInterface.writeState <= 1;
          invalidateEnable          <= 1;
        end else begin
          if (slaveInterface.readEnabled == 1) begin
            slaveInterface.functionComplete <= 1;
          end else if (slaveInterface.functionComplete == 1 && slaveInterface.address[OFFSET_WIDTH - 1 : 0] == 0) begin
            cacheInterface.writeState <= 1;
            invalidateEnable          <= 1;
          end
        end
      end
    end
  endcase
end
```

Слика 4.7.2 Имплементација контролера који надгледа магистралу

Уколико је тренутна трансакција на магистрали трансакција читање и уколико се тражени блок података не налази у кеш меморији не предузимају икакве акције. Ако се налази у кеш меморији и ако је кеш меморија „власник“ истог и контролер има одобрење од арбитра да може да користи магистралу, садржај блока података се доставља кеш меморији која је започела трансакцију читања. Независно од тога да ли је кеш меморија „власник“ или није, стање датог блока података у кеш меморији се ажурира уколико је то потребно.

Уколико је тренутна трансакција на магистрали инвалидација и уколико се дати блок података налази у кеш меморији, он се инвалидира тако што на одговарајућу позицију у меморији стања упише невалидно стање и јави контролеру који је започео трансакцију

инвалидације да је инвалидиран. Ако се блок података не налази у кеш меморији не предузимају се икакве акције.

Уколико се тренутно на магистрали извршава трансакција ексклузивног читања и уколико се блок података не налази у кеш меморији не предузимају се икакве акције. Ако је исти присутан у кеш меморији и кеш меморија је „власник“ датог блока података, његов садржај се доставља кеш меморији која је започела трансакцију ексклузивног читања. Потом се, без обзира да ли кеш меморија „власник“ или не, дати блок података инвалидира. Након тога се кеш меморији која је започела трансакцију ексклузивног читања јавља да је он инвалидиран.

3.8. Секвенцијална логика која спречава конфликте у кеш меморији

Овај јединица је задужена за спречавања конфликта у кеш меморији. Ова јединица се може схватити као нека врста омотача који пропушта захтеве или трансакције до контролера када је то безбедно. У овом решењу конфликте ситуације се дешавају кад су адреса којој језгро приступа и адреса која се налази на магистрали исте и оне су следеће:

1) Читање од стране језгра и трансакција инвалидације од стране друге кеш меморије
2) Читање од стране језгра и трансакција ексклузивног читање од стране друге кеш меморије
3) Упис од стране језгра и трансакција читања од стране друге кеш меморије
4) Упис од стране језгра и трансакција инвалидације од стране друге кеш меморије
5) Упис од стране језгра и трансакција ексклузивног читања од стране друге кеш меморије
6) Враћање блока података, односно ажурирање оперативне меморије
7) Трансакција инвалидација блока података који се већ налази у кеш меморији

Ситуације 1 и 2 се решавају тако што се опслужује захтев који је дошао први. На пример уколико је захтев од језгра дошао први он ће бити опслужен први, а трансакција инвалидације или ексклузивног читања неће стићи до контролера који надгледа магистралу док се захтев за читање не опслужи.

Ситуације 3, 4 и 5 су конфликте једино ако се остале копије блока података у који се уписује не морају инвалидирати. Оне се решавају на исти начин као ситуације 1 и 2.

Ситуација 6 је конфликта зато што ако дозволимо да контролер који надгледа магистралу активира приликом ове трансакције, трансакција неће стиће до оперативне меморије што је била и намера. Из овог разлога морамо спречити да трансакција стигне до контролера који надгледа магистралу да би дозволили оперативној меморији да се активира.

Ситуација 7 је конфликта из разлога што се дати блок података може већ налазити у кеш меморији и што контролер који надгледа магистралу може да припада кеш меморији чији је контролер који опслужује захтева језгра започео трансакцију инвалидације. У том случају трансакција не сме да стигне до контролера који надгледа магистралу јер би се дати блок података инвалидирао што није била намера. Ова ситуација се решава тако што се захтев не прослеђује до контролера који надгледа магистралу, а линија којом си обавештава контролер који је опслужује захтеве језгра који је започео трансакцију инвалидације подигне на логички ниво јединице.

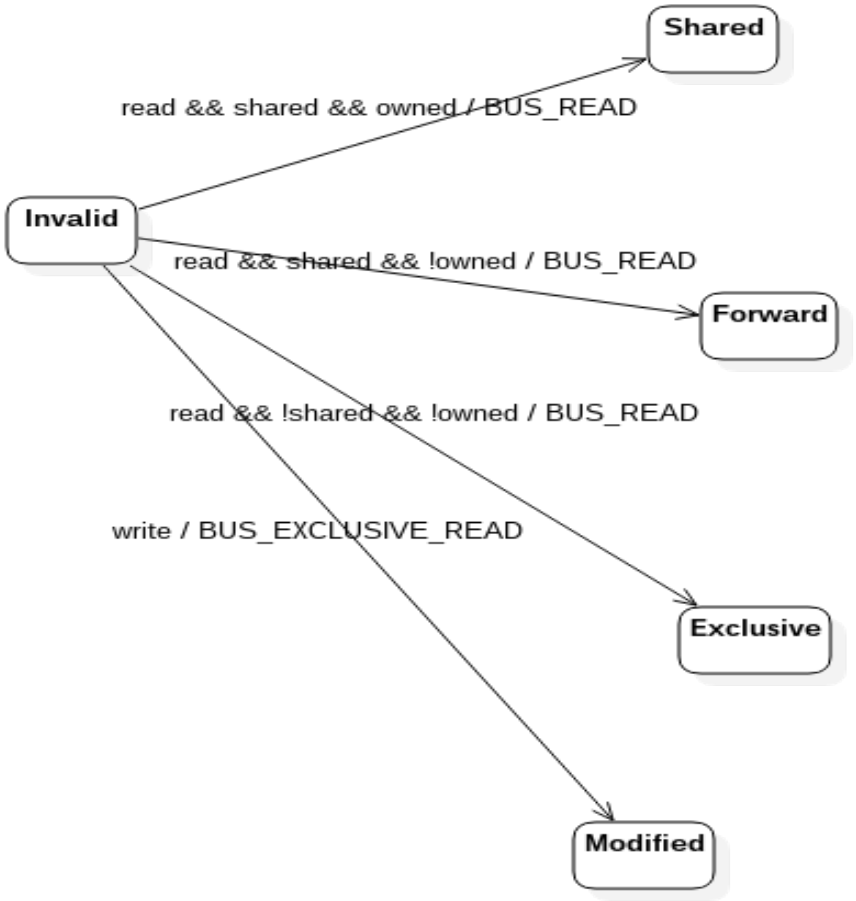
3.9. МОЕСИФ протокол

МОЕСИФ протокол спада у групу инвалидирајућих протокола за кеш кохеренцију. Предтавља мешавину МОЕСИ и МЕСИФ протокола као што је наведено на вебб страници енциклопедије *Wikipedia*[7]. Његово име је састављено из почетних слова имена стања у којима се бложови података у кеш меморији могу наћи. Стања су следећа: модификовано, власништво, ексклузивно, дељено, невалидно и прослеђено (енгл. *Modified*, *Owned*, *Exclusive*, *Shared*, *Invalid* и *Forward*).

Када је блок података у модификованом стању значи да је извршен упис у исти и да је то једина копија у систему. Када је блок података у стању власништва то значи да је то језгро последње извршило упис у исти. У систему може, а не мора постојати још копија датог блока података али је та кеш меморија одговорна за ажурирање оперативне меморије приликом његовог избацивања. Када је блок података у ексклузивном стању то значи да у систему не постоји других копија али у исти није извршен упис. Када је блок података у дељеном стању то значи да може, а не мора постојати још копија истог у систему и да у њега није извршен упис. Кад је блок података у невалидном стању то значи да није присутан у кеш меморији. Када је блок у прослеђеном стању то значи да је дато језгро последње читало из истог. У том случају дата кеш меморија је одговорна за опслуживање свих следећих трансакција за читање датог блока података јер је најмање вероватно да ће исти бити избачен из те кеш меморије. Стања власништва и прослеђености су уведена да би се приступи оперативној меморији смањили.

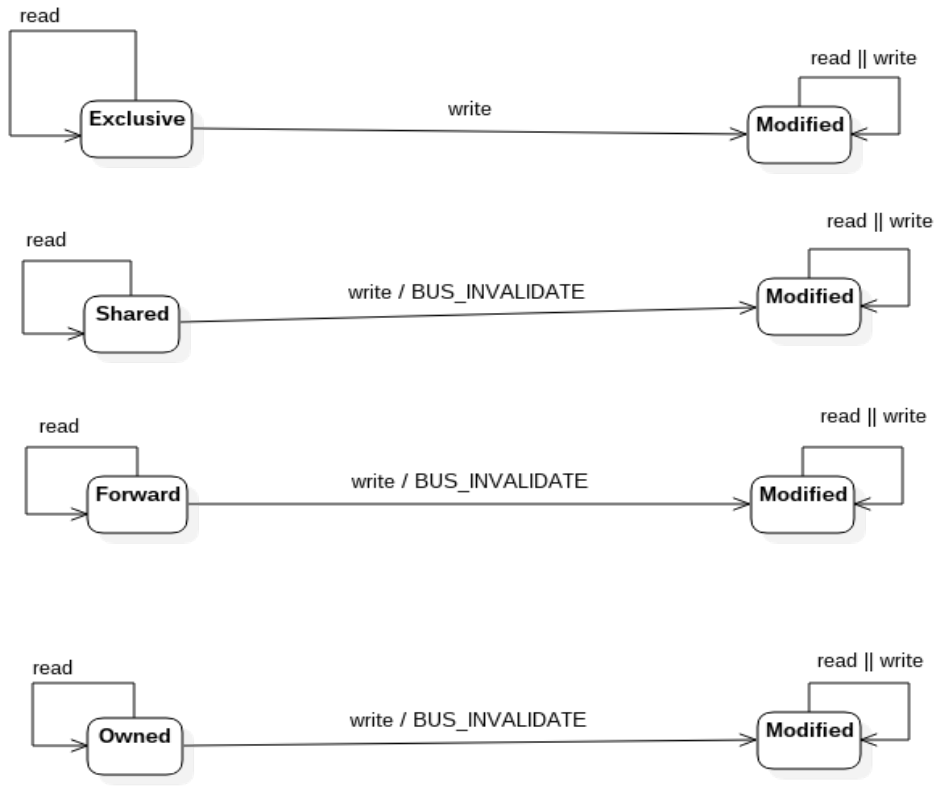
За саму имплемтацију протокола потребне су додатне линије на магистрали, *shared* и *owned* линија. *Shared* линија има исту сврху као у МЕСИФ протоколу. Приликом трансакције читања назначава да ли постоји копије прочитаног блока података у другим кеш меморијама. *Owned* линија је додата у МОЕСИ протоколу. Прилико трансакције читања назначава да ли постоји копија прочитаног блока података у другој кеш меморијаме која је у стању власништва. Разлог је следећи. Замислимо систем са две кеш меморије. У првој кеш меморији се налази блок података у модификованом стању. Уколико друто језгро жели да прочита податак из истог, друга кеш меморија га мора прво прочитати. Приликом тог читања у првој кеш меморији блок података прелази у стање власништва, док у другој прелази у стање прослеђености (уколико не постоји *owned* линија). Затим, после извесног времена, друга кеш меморија избаци дати блок података без ажурирања оперативне меморије јер је он био у прослеђеном стању. Сваки следећи пут дати блок података ће бити прочитан из оперативне меморије уместо из прве кеш меморије у којој се заправо налази једина ажурна копија истог. Стога, ова линија је потребна да би у оваквим случајевима нагласила да је ново стање блока података приликом читања друге кеш меморије дељено и да исти ипак мора бити прочитан из прве кеш меморије, а не оперативне меморије.

Сам дијаграм стања протокола је компликован тако да ћемо овде приказивати део по део дијаграма. Прво ћемо приказати прелазе који се дешавају као последица операција језгра.



Слика 4.9.1 МОЕСИФ протокол

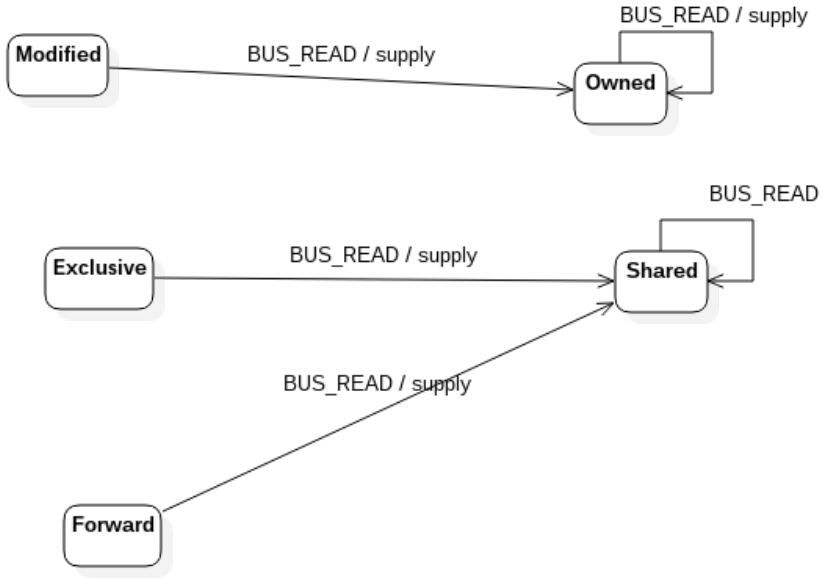
На слици 4.9.1 приказан је први део протокола. Сви блокови података су на почетку у невалиндом стању јер нису присутну у кешу. Уколико је језгро издало операцију читања, прилико довлачења блока података у кеш меморију стање блока зависи од стања линија *shared* и *owned*. Уколико су обе на логичком нивоу јединице ново стање ће бити дељено. Ако је сам *shared* линија подигнута на логички ниво јединице то значи да се исти налази у другим кеш меморијама али да ни у једној није у стању власништва, па стога ново стање блока ће бити прослеђено и дата кеш меморија ће бити одговоран за опслуживање свих трансакција за читање датог блок података од стране других кеш меморија. Ако су и једна и друга линија на нули то значи да нема копија датог блока у систему и ново стање блока је ексклузивно. У све три ситуације трансакција која се извршава на магистрали је трансакција читања. Уколико језгро жели да упише у дати блок ново стање блока ће бити модификовано. У овом случају трансакција која ће се извршити на магистрали је трансакција ексклузивног читања.



Слика 4.9.2 МОЕСИФ протокол

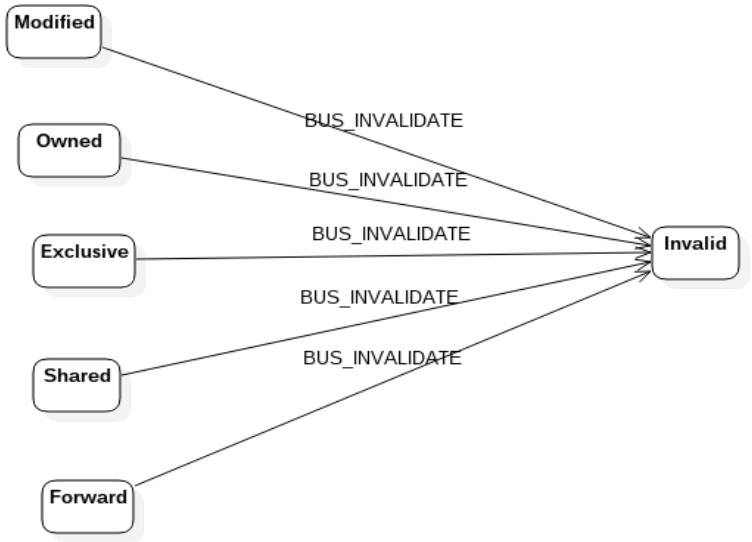
На слици 4.9.2 приказан је други део протокола. Уколико се блок налази у кеш меморији прилико читања неће бити извршена трансакција на магистрали и блок остаје у истом стању као што се види са слике. Уколико је језгро изадало захтев за упис и уколико се блок налази у ексклузивном или модификованом стању, неће бити трансакције на магистрала јер постоји само једна копија датог блока у систему, па нема потребе за инвалидацијом других копија. У свим осталим случајевим остале копије истог у другим кеш меморија се морају инвалидирати пре уписа, па се стога на магистрали извршава трансакција инвалидације. Приликом уписа ново стање блока је модификовано.

Сада ћемо описати прелазе који се дешавају као последица трансакција на магистрали.



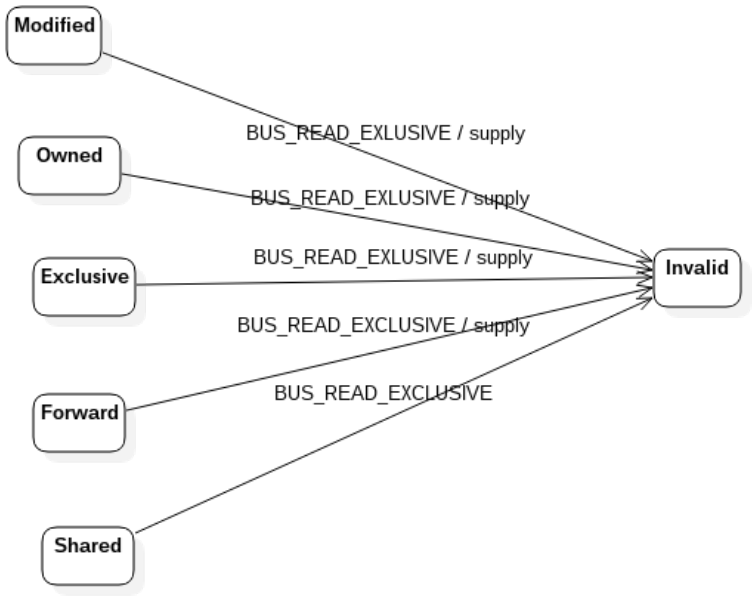
Слика 4.9.3 МОЕСИФ протокол

На слици 4.9.3 приказани су прелази који се дашавају услед трансакције читања. У сличају да је блок у модификованом, ексклузивном, прослеђеном или у стању власништа прилико операције читања дата кеш меморија доставља податке. Разлози за то су што када је блок у модификованом или ексклузивном стању то је једина копија у сустему, када је у прослеђеном стању избегава се приступ оперативној меморији, а кад је блок у стању власништва, оперативна меморија није ажурна па дата кеш меморија мора доставити податке да би кеш меморија која је издала захтев за читање имала најажурнију верзију података.



Слика 4.9.4 МОЕСИФ протокол

На слици 4.9.4 приказани су прелази који се дешавају приликом трансакције инвалидације. Приликом инвалидације нема преноса података на магистрали а ново стање је увек невалидно.



Слика 4.9.5 МОЕСИФ протокол

На слици 4.9.5 приказани су прелази који се дешавају приликом трансакције ексклузивног читања. У случају да се блок налази у модификованом, ексклузивном, прослеђеном или стању власништва дата кеш меморија је дужна да достави потребне податке из истог разлога као код трансакције читања. У сви случајевима се тражени блок инвалидира и ново стање блока је невалидно.

Сама имплементација протокола је јако тривијална. Наиме, то је обична комбинациона мрежа која као улазне податке има следеће параметер: тип захтева које је издало језгро, стање блока података у кеш меморији који се налази на позицији одређеном адресом коју је издало језгро, трансакцију која се тренутно врши на магистрали и стање блока података у кеш меморији који се налази на позицији одређеној адресом на магистрали. Излазни параметри протокола су: да ли је потребна инвалидација осталих копија датог блока података, да ли је потребно враћање истог, односно ажурирање оперативне меморије, да ли је потребна трансакција ексклузивно читање и да ли је дати контролер који надгледа магистралу одговоран за достављање података.

3.10. Меморијски систем

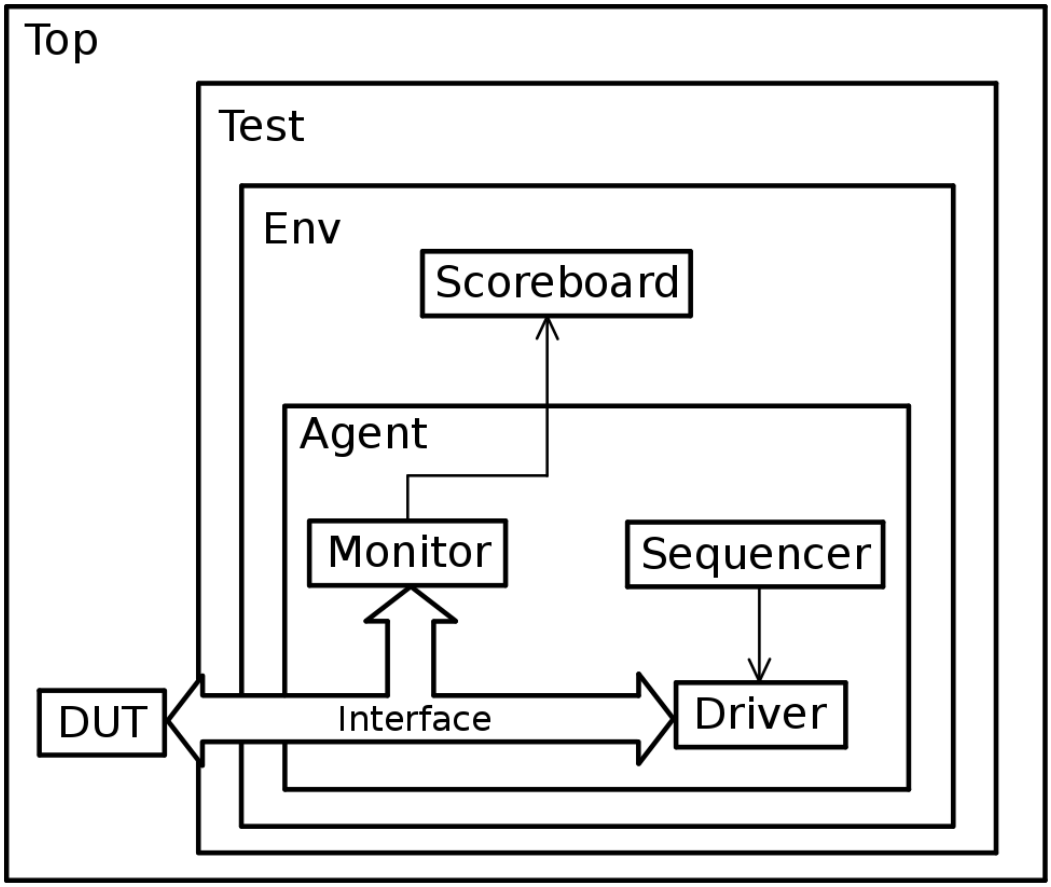
Целокупни меморијски систем се састоји из више кеш меморија које су повезане једном магистралом. Постоји само један ниво кеш меморија. Испод тог нивоа кеш меморија налази оперативнам меморија из које се читају подаци приликом промашаја у кеш меморији. Цео систем је лако конфигурабилан, односно сваки имплементирани модул је параметризован па је веома лако променити параметре као што су величина таг, индекс и офсет поља, број кеш меморија, број тактова који је потребан оперативној меморији да обради захтев итд.

4. ТЕСТИРАЊЕ

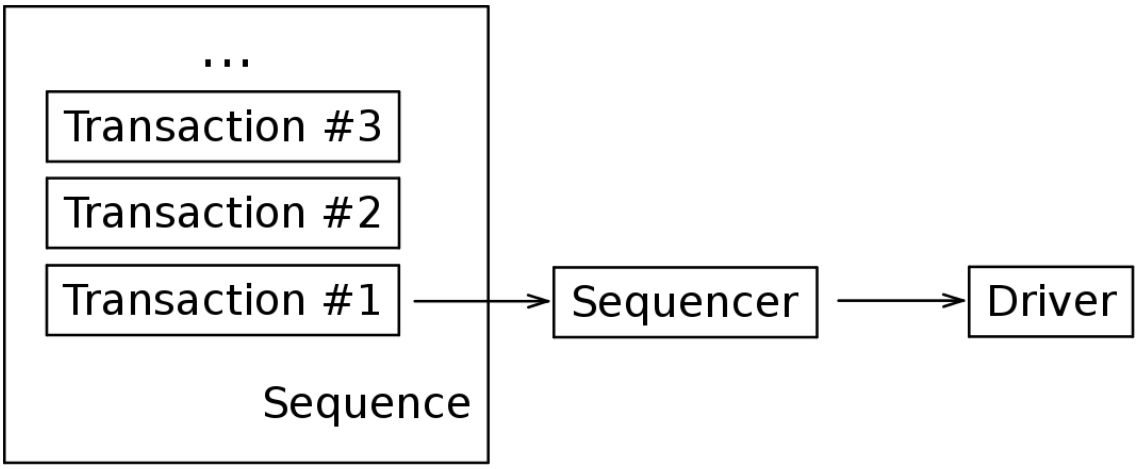
У овом поглављу дат је опис *UVM* библиотеке и окружења за тестирање које она пружа. Биће описане компоненте самог теста и тестови који су написани за верификацију имплементације овог рада.

4.1. Компоненте *UVM* библиотеке

Цела библиотека се ослања на објектно-оријентисани део језика *SystemVerilog*. Разлог за то је поновно коришћење кода које пружа сама објектно-оријентисана парадигма. Главни део библиотеке је класа која се назива *uvm_component* коју користимо као родитељску класу приликом дефинисања компоненти тест, међутим *UVM* библиотека пружа већ унапред дефинисане компоненте које се могу искористити за имплементацију теста. Оне су трансакција, секвенца, секвенцер, драјвер, монитор, агент, табела резултата, окружење и тест.



Слика 5.1.1 UVM тест са компонентама



Слика 5.1.2 Секвенца, секвенцер и драјвер

4.1.1. Трансакција

Трансакција (енгл. *Transaction)* представља најмањи трансефер података који може бити извршен од стране модула кога тестирамо. Представљена је класом *uvm_sequence_item*. Могу садржати варијабле и ограничења која се морају поштовати прилико рандомизације. Са обзиром на то да су трансакције на веома високом нивоу апстракције, нису свесне протокола по којем компоненте и модули који се тестирају комуницирају. Из тога разлога се могу користити и проширивати да би задовољиле услове других тестова.

4.1.2. Секвенца

Секвенца (енгл. *Sequence)* преставља скуп трансакција које се у извршавају у датом тесту. Представљена је класом *uvm_sequence*. Секвенца преко секвенцера доставља трансакције драјверу који их изврашава. На слици 5.1.2 је приказано како ове компоненте комуницирају.

4.1.3. Секвенцер

Секвенцер (енгл. *Sequencer)* је компонента која одговорна за серијализацију и распоред прослеђивања (могуће је дефинисати приоритете) трансакција драјверу. Представљена је класом *uvm_sequencer*. На сликакам 5.1.1 и 5.1.2 представљена је позиција секвенцера у тесту.

4.1.4. Драјвер

Драјвер (енгл. *Driver)* је компонента која задужена за контролисање нивоа улазних сигнала модула. Драјвер итеративно извршава трансакцију по трансакцију коју добија од секвенцера. Као што се види са слике 5.1.1, драјвер је повезан са модулом преко интерфејса. Представљен је класом *uvm_driver*.

4.1.5. Монитор

Монитор (енгл. *Monitor)* је компонента која задужена за праћење трансакција које извршавају и прављење „пакета“ који представљају резултате једне транскације. Монитор је, слично драјверу, повезан са модулом преко интерфејса. Међутим његово је задужење је

праћење стања сигнала, односно праћење трансакција које извршавају и њихових резултата, и прављење „пакета“ који осликавају стање модула за каснију анализу. На слици 5.1.1 дата је пожиција монитора у систему. Представљен је класом *uvm_monitor*.

4.1.6. Агент

Агент (енгл. *Agent)* је компонента која преставља „кутију“ у којој се налазе секвецер, драјвер и монитор. Представљен је класом uvm_agent.

4.1.7. Табела резултата

Табела резултата (енгл. *Scoreboard)* је компонента која преставља златну имплементацију (енгл. *Golden implementation*), односно представља класну имплементацију модула који тестирамо. На основу пакет које прима од монитора анализира рад модула који се тестира и обавештава о грешкама које су се десиле. Представљен је класом *uvm_scoreboard*.

4.1.8. Окружење

Окружење (енгл. *Environment)* је компонента која преставља „кутију“ у којој се налазе агенти и табела резултата. Представљен је класом *uvm_env*.

4.1.9. Тест

Тест (енгл. *Test)* је главна компонента теста. Њено задужење је инстанцирање окружења и покретањем секвенци над секвенцерима у агентима окружења. Представљен је класом *uvm_test*.

4.2. Тестови

Постоје два типа теста који су се користили за верификацију приликом имплементације: основни тест и тест целокупног систем.

4.2.1. Основни тест

Основни тест је најједноставнија варијанта теста који се може имплементирати помоћу *UVM* библиотеке. Састоји из једног типа трансакције, једне секвенце, једног драјвер, једног монитора, једног агента, једне табле резултата, једног окружења и једног теста. У самој имплементацији рада постоји пакет основних класа које су изведене из компоненти *UVM* библиотеке које су већ пружене за лакше писање тестова. Сваки тест који написан за конкретну компоненту садржи класе које су изведен из класа овог пагета. Класе се изводе по потреби. Најћешће су то трансакција, драјвер, монитор, табела резултата и тест.

Разлог за извођење трансакција, драјвера монитора и табеле резултата је очигледан. Ово је потребно зато што се модули разликују и зато што су улазни подаци, излазни подаци и сам протокол комуникације различити.

Разлог за извођење из теста је нешто сложенији. Наиме, као што је написано на веб страници водича за коришћење библиотеке[8], сама библиотека поседује механизам фабрике. Прилико дефинисања компоненти теста користимо предефинисане макрое да региструје дате компоненте унутар фабрике библиотеке, а касније користимо ту фабрику за инстанцирање компоненти тест. Међутим, могуће је преписати типове регистроване код фабрике али тако да остале компоненте које је користе фабрику за инстанцирање компоненти то неће видети, већ ће фабрику користити на исти начин као пре. Наравно компоненте које ће бити инстанциране

су компоненте чији су типови преписали оригиналне типове из фабрике. Из овог разлога се за сваки тест дефинише нова класа теста која ће регистровати све промене у типовима које смо направили прилико дефинисања новог теста. На тај начин обезбеђујем да претходно дефинисане компоненте у основном тесту користе новодефинисане компоненте уз минимално мењање кода.

4.2.2. Тест целокупног система

Овај тест није изведен из основног теста из разлог што се овде тестира рад целог система, односно самих кеш меморија као и комуникација која се дешава између њих. У овом случају нам је потребан по један агент за сваку кеш меморију. Тест се састоји из низа меморијских трансакција, које моги бити или упис или читање. Монитори надгледају рад кеш меморија и достављају табелама резултата податке о трансакцијама које су се одвијале. Табела резултата након тога проверава да ли је одржан кохерентни меморијски систем, односно да ли је прочитана вредност иста као и последња вредност која је уписана на ту меморијску локацију.

5. Анализа перформанси

Саме кеш меморије су имплементиране тако да су независне од тога који се инвалидирајући протокол за кеш кохеренцију користи. Наиме, кораци које извршавају контролер који опслужује захтеве језгра или контролер који надгледа магистралу су исти у случају било ког инвалидирајућег протокола за кеш кохеренцију. Од њега зависе само одлуке да ли ће се копије блока података инвалидирати, да ли ће се блок података вратити у оперативну меморију и сл. Из овог разлога имплементирање других инвалидирајућих протокола за кеш кохеренцију није захтевало много измена оригиналног кода. Поред МОЕСИФ протокола за кеш кохеренцију имплементирани су МСИ, МЕСИ, МЕСИФ и МОЕСИ. Такође је имплементиран систем који не поседује кеш меморије. На овај начин можемо упоредити перформансе МОЕСИФ са другим протоколима за кеш кохеренцију из исте групе и да ли се уопште исплати имати кеш меморију.

За сам имплементацију новог инвалидирајућег протокола за кеш кохеренцију било је потребно имплементирати нову комбинациону мрежу која представља сам протокол и у неким ситуацијама је такође било потребно променити магистралу. Разлог за мењање магистрале је тај што нема сваки протокол стање власништва као оно у МОЕСИФ протоколу, односно нема стање у којем кеш меморија одговорна за ажурирање оперативне меморије. Зато је у оваквим случајевим приликом трансакције читања потребно поред достављања подата кеш меморији која је започела трансакцију ажурирати оперативну меморију. Из тога разлога је основна магистрала морала бити проширена комбинационом логиком која ће ово обезбедити.

Међутим овде долазимо до новог проблема. Сама кеш меморија се исплати једино ако секвенца адреса којима се приступа поштује временску и просторну локалност. То јесте случај са реалним језгрима, међутим ми овде морамо то вештачки обезбедити.

```
virtual function void myRandomize();
  int fillCount = 0;
  while (fillCount < SEQUENCE_ITEM_COUNT) begin
    bit[ADDRESS_WIDTH - 1 : 0] address         = $urandom_range(SIZE_IN_WORDS - 1, 0);
    bit[DATA_WIDTH - 1    : 0] data            = $urandom();
    bit                        isRead          = $urandom();
    int                        adjacentCount   = $urandom_range(MAX_ADJACENT_ADDRESSES, MIN_ADJACENT_ADDRESSES);
    int                        repetitionCount = $urandom_range(MAX_NUMBER_OF_REPETITIONS, MIN_NUMBER_OF_REPETITIONS);

    for (int i = 0; i < repetitionCount && fillCount < SEQUENCE_ITEM_COUNT; i++) begin
      for (int j = 0; j < adjacentCount && fillCount < SEQUENCE_ITEM_COUNT; j++, fillCount++) begin
        this.address[fillCount] = (address + j) % SIZE_IN_WORDS;
        this.data[fillCount]    = data;
        this.isRead[fillCount]  = isRead;
      end
    end
  end
endfunction : myRandomize
```

Слика 6.1 вештачко обезбеђивање просторне и временске локалности

На слици 6.1 приказан је исечак кода који илуструје на који начин је обезбеђена просторна и временска локалност. Сва имена написана великим словима представљају константе које је могуће мењати. Најпре се одреди почетна адреса низа локација којима се

приступа и податак који се уписује ако је у питању захтев за упис (променљива *isRead* има вредност нула). Затим се одреди коликом бројју суседних локација приступамо и колико пута понављамо такав приступ. Цео поступак се понваља док се не попуне три низа која представљају поља класе која чувају адресе, податке и индикатора да ли је у питању захтев за читање. Након тога се дати низови користе за генерисање трансакција.

Анализа је спровођења за адресу ширине осам бита од чега је таг поље ширине четири бита, индекс поље ширине два бита и офсет поље ширине два бита. Сет асоцијативност кеш меморије је један бит. Број трансакција је четиристо. Постоје четири језгра у систему, односно постоје четири драјвера а самим тиме и четири кеш меморије, једна магистрала и једна оперативна меморија. Симулација је изведена како један тест. Постоје више модула, односно система, и окружења и над сваким се покреће иста секвенца адреса која је изгенерисана пре почетка теста.

Таβела 6.1 перформансе различитих конфигурација система

Протокол	**Број тактова потребан за завршетак свих трансакција**
-(нема кеш меморије)	17602
МСИ	6170
МЕСИ	6170
МЕСИФ	6170
МОЕСИ	4042
МОЕСИФ	4042

Као што се види у таβели 6.1, МОЕСИФ и МОЕСИ имају најβоље перформансе за дату секвенцу трансакција. Разлог за то је одложено враћање блока у оперативну меморију која је доста спорија од саме кеш меморије, односно стање *Owned*. Као што је већ наведено, прилико трансакције читања уколико се блок налази у стању „власништва“ (ово је дефинисано протоколом) у датој кеш меморија, она има обавезу да упоредо са достављањем података другој кеш меморији ажурира оперативну меморију. У случају МОЕСИ и *МОЕСИФ* ажурирање је одложено до самог избацивања блока из кеш меморије па самим тим је и смањена потреба за приступу оперативној меморији. Самим тим ће ова два протокола дати βоље резултате у за дату секвенцу трансакција.

6. Закључак

Циљ овог рад је био да се прикаже један пример дизајна кеш меморије која је заснована на МОЕСИФ протоколу за кеш кохеренцију. У том смислу, поред саме имплементације свих модула неопходних за сам рад кеш меморије, укључено је и тестирање самих модула помоћу *UVM* библиотеке, која је данас широко у употреби. Поред само имплементације, извршена је и једноставна анализа протокола заједно са образложењима зашто су резултати баш такви.

Сама имплементација рада није најефикаснија. Реални системи имају доста комплексније кеш меморије, не само у погледу дизајна већ и у организацији саме кеш меморије (број нивоа, кешева по нивоу и сл.). Међутим дизајн конфигурабилан, у смислу да се сами параметри кеш меморије могу мењати по потреби (дужина адресе и поља, број тактова који потребан оперативној меморији да опслужи захтев и сл.) па се може искористити за тестирање различитих конфигурација и анализу њихових перформанси. Поред тога, имплементација новог инвалидирајућег протокола за кеш кохеренцију је могућа уз малу дораду кода. Такође, важно је напоменути да су у саму израду укључени тестови који се такође могу лако дорадити тако да могу да послуже за тестирање нових модула. Стога, иако није најефиксанији дизајн кеш меморије данас, може се користити у едукативне сврхе, односно показивања принципа рада кеш меморије и инвалидирајућих протокола за кеш кохеренцију, самог начина тестирања који данас широко примењив као и анализе перформанси различитих конфигурација.

Сам рад се такође може унапредити на много начина са обзиром на то да је сама област која се бави кеш меморијама и протоколима за кеш кохеренцију од доста велико значаја за даљи развој процесора и самим тим јако заступљена. Једно од могућих унапређења, предложено у интересантном раду[9], јесте дефинисање новог протокола за кеш кохеренцију који ће се показати боље од оних који су демонстрирано у овом раду. Такође, један од великих недостатака кеш меморија које су засноване на инвалидирајућим протоколима за кеш кохеренцију је магистрала која као интерконекциона мрежа има слаб пропусни опсег у односу на интерконекционе мреже општијег типа. Један од наредних корака може бити имплементација произвољне интерконекционе мреже која обезбеђује коректан рад ове групе протокола. Један од таквих, предложена у раду[10], јесте мрежа општег типа која је заснована на временским маркерима који обезбеђују исправан поредак трансакција иако оне нису серијализоване као у случају са магистралом.

Литература

[1] David A. Patterson, John L. Hennessy, "Computer Organization and Design: The Hardware/Software Interface", 4th edition, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2008

[2] , "Intel Nehalem (i7) Cache", http://sabercomlogica.com/en/ebook/a-case-study-intel-nehalem-i7-coherence-in-cache/, pristupano septembar 2017

[3]Intel, "Intel® 64 and IA-32 ArchitecturesSoftware Developer's Manual", 2016

[4]Advanced Micro Devices, "Software Optimization guide for AMD family 15h processors", 2014

[5]Advanced Micro Devices, "AMD64 Architecture Programmer's Manual Volume 2: System Programming", 2013

[6] , "The Effect and Technique ofSystem Coherence in ARMMulticore Technology", http://www.mpsoc-forum.org/previous/2008/slides/8-6%20Goodacre.pdf, pristupano septembar 2017

[7] , "MESIF protocol", https://en.wikipedia.org/wiki/MESIF_protocol, pristupano septembar 2017

[8] , "How UVM Factory Works..??", http://www.learnuvmverification.com/index.php/2015/08/19/how-uvm-factory-works/, pristupano septembar 2017

[9] Peter Hornyack, ""Screwdriver:" An HDL implementation of an advanced cache coherence protocol", https://pdfs.semanticscholar.org/73ca/b7706a6495186c77dc53bd486624e4923303.pdf?_ga=2.104863428.697828691.1498385124-490165764.1498385124, pristupano septembar 2017

[10] Milo M. K. Martin, Daniel J. Sorin, Anastassia Ailamaki, Alaa R. Alameldeen, Ross M. Dickson,Carl J. Mauer, Kevin E. Moore, Manoj Plakal, Mark D. Hill, David A. Wood, "Timestamp Snooping: An Approach for Extending SMPs", https://www.cis.upenn.edu/~milom/papers/asplos2000_timestamp_snooping.pdf, pristupano septembar 2017

Списак скраћеница

L1	*Level 1*
L2	*Level 2*
L3	*Level 3*
SCU	*Snoop Control Unit*
AXI	*Advanced Extensible Interface*
LRU	*Least Recently Used*
UVM	*Universal Verification Methodology*

СПИСАК СЛИКА

СПИСАК ТАБЕЛА