

دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)
دانشکده مهندسی کامپیوتر

درس فناوری های حافظه
دکتر حامد فربه

رضا آدینه پور
۴۰۲۱۳۱۰۵۵

تمرین شبیه سازی اول

تدریسار:

مرتضی عادلخانی (madelkhani@aut.ac.ir)
سارا زمانی (sara.zamani73@aut.ac.ir)

۱. نام چهار شبیه‌ساز که در زمینه فناوری های حافظه مورد استفاده قرار می‌گیرند را نام ببرید و آنها را بررسی کنید. زبان برنامه نویسی هرکدام، برنامه هایی که می‌تواند پشتیبانی کند و چالش های احتمالی را بیان کنید. همچنین مزایا و معایب هرکدام را بررسی کنید

شبیه سازهایی که در این تمرین بررسی شده است:

نام شبیه‌ساز	کاربرد	زبان برنامه نویسی	مزایا	معایب	سایت
CACTI	تحلیل و شبیه سازی حافظه‌های نهان ^۱ بزرگ	C/C++	سرعت بالا	فاصله زیاد نسبت به دنیای واقعی	Website
NVSIM	طراحی و شبیه‌سازی حافظه های غیر فرار	C/C++	پشتیبانی از حافظه های جدید	عدم وجود نسخه رسمی از شبیه‌ساز	Website
DRAMSim	مدلسازی حافظه های پویا ^۲	C/C++	پشتیبانی از تکنولوژی های جدید	عدم ارائه گزارش جامع	Website
SimpleScaler	شبیه‌سازی اجزای کامپیوتر	C	شبیه‌سازی کامل یک سیستم کامپیوتری	عدم ارائه تحلیل های کامل	Website

۱ شبیه‌ساز CACTI

اولین بار این شبیه‌ساز در سال ۱۹۹۳ توسط دکتر جوپی^۳ و دکتر ویلتون^۴ در یکی از آزمایشگاه های شرکت HP توسعه داده شد. CACTI ابزاری تحلیلی^۵ است. به طور کلی عملکرد این شبیه ساز را می‌توان به دو دسته زیر تقسیم کرد:

۱. شبیه‌سازی و مدلسازی دسترسی به حافظه

۲. مدلسازی انواع مسیرها و سیم ها

اگرچه این شبیه‌ساز همه سلسله مراتب حافظه را شبیه‌سازی می‌کند اما همانطور که از دو حرف اول اسم این شبیه‌ساز مشخص است^۶، کاربرد اصلی این شبیه‌ساز در تحلیل حافظه‌های نهان است.

این شبیه‌ساز می‌توان سطوح مختلف حافظه نهان، سیاست های مختلف انجمنی^۷ و سیاست های جایگزینی^۸ و ... را برای ایجاد تغییرات و شبیه سازی فراهم کند.

در بخش اول، مجموعه ای از پارامترهای حافظه «به خصوص حافظه نهان^۹» را به عنوان ورودی می‌گیرد و پارامترهایی

^۳Dr. Jouppi

^۴Dr. Wilton

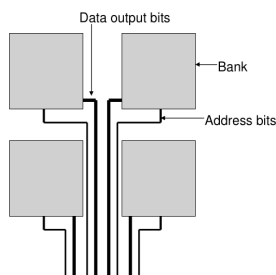
^۵Analytical

^۶Cache Architecture

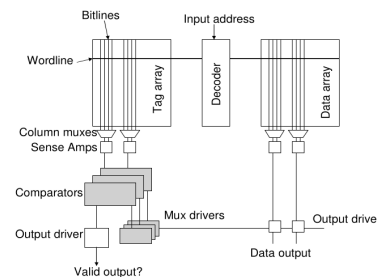
^۷Associativity

^۸Replacement policies

^۹Cache



(ب) ساختار فیزیکی Cache



(آ) ساختار منطقی Cache

شکل ۱: ساختار منطقی و فیزیکی Cache

مثل زمان دسترسی^{۱۰}، توان^{۱۱}، زمان چرخه^{۱۲}، مساحت^{۱۳} و ... را محاسبه می‌کند. در بخش دوم، CACTI قابلیت تست و مدل‌سازی تاخیر، توان، مساحت و ... در مسیرهای حافظه، مانند خط آدرس^{۱۴} و خطوط داده^{۱۵} را دارد.

CACTI به دو صورت در دسترس است:

۱. وب^{۱۶}

۲. سورس کد C++

که در ادامه نحوه نصب و کار با شبیه ساز ۷.۰ CACTI «آخرین ورژن» را توضیح خواهیم داد. برای دانلود و نصب شبیه‌ساز CACTI بر می‌توان طبق مراحل زیر عمل کرد:

۱.۱ دانلود و نصب

در گام اول می‌بایست وابستگی^{۱۷} های مورد نیاز را نصب کرد. برای نصب به صورت زیر عمل می‌کنیم:

```
$ sudo apt-get update
$ sudo apt-get install build-essential
```

۱.۱.۱ دانلود شبیه‌ساز

پس از نصب Dependency ها باید سورس کد شبیه‌ساز را دانلود کنیم. به صورت زیر عمل می‌کنیم:

```
$ git clone https://github.com/HewlettPackard/cacti.git
```

Access time^{۱۰}

Power^{۱۱}

Sycle time^{۱۲}

Area^{۱۳}

Word line^{۱۴}

Bit line^{۱۵}

quid.hpl.hp.com:9081/cacti/^{۱۶}

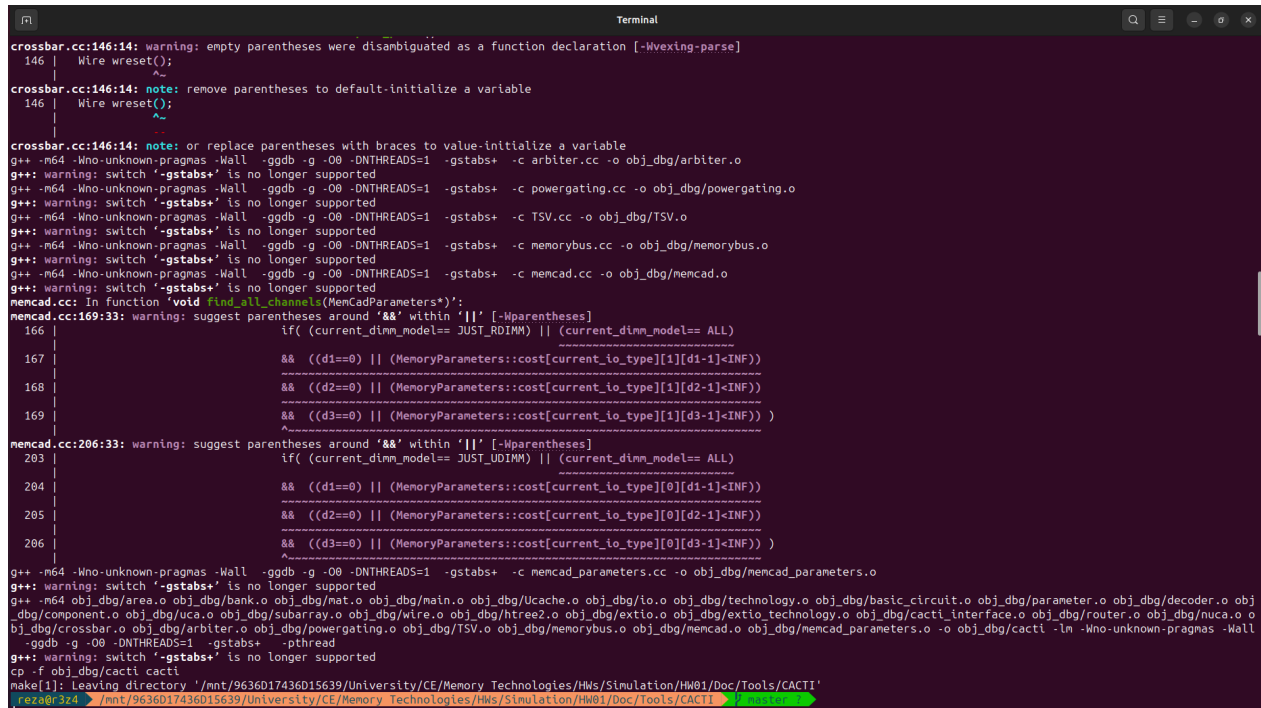
Dependency^{۱۷}

۲.۱.۱ بیلد شبیه ساز

پس از Clone کردن مخزن^{۱۸}، به دایرکتوری مخزن میرویم و شبیه ساز را Build می کنیم:

```
$ cd CACTI
$ make
```

اگر شبیه ساز به درستی Build شود، خروجی ترمینال به صورت زیر خواهد شد:



شکل ۲: Build موفقیت آمیز

۳.۱.۱ تنظیم پارامترهای Cache

در ورژن های قدیمی تر CACTI می توانستیم پارامترهای Cache را به صورت Command درون ترمینال تنظیم کنیم. اما از ورژن ۶ به بعد نرم افزار تمامی کانفیگ ها به درون فایلی با نام `cache.cfg` منتقل شده است. «شکل ۳»

پارامترهای مختلفی مانند سایز کش، Block size، نوع تکنولوژی و ... وجود دارد که ما تغییری در آن ایجاد نمی کنیم تنظیمات پیش فرض^{۱۹} را می پذیریم.

```
Cache size
//size (bytes) 2048
//size (bytes) 4096
//size (bytes) 32768
-size (bytes) 131072
//size (bytes) 262144
//size (bytes) 1048576
//size (bytes) 2097152
//size (bytes) 4194304
//size (bytes) 8388608
//size (bytes) 16777216
//size (bytes) 33554432
//size (bytes) 134217728
//size (bytes) 67108864
//size (bytes) 1073741824

# power gating
-Array Power Gating - "false"
-WL Power Gating - "false"
-CL Power Gating - "false"
-Bitline Floating - "false"
-Interconnect Power Gating - "false"
-Power Gating Performance Loss 0.01

# Line size
//block size (bytes) 8
-block size (bytes) 64

# To model Fully Associative cache, set associativity to zero
//associativity 0
-associativity 2
//associativity 4
//associativity 8

-read-write port 1
-exclusive read port 0
-exclusive write port 0
-single ended read ports 0

# Multiple banks connected using a bus
-UCA bank count 1
//technology (u) 0.022
//technology (u) 0.040
//technology (u) 0.032
-technology (u) 0.090
```

شکل ۳: محتوای درون فایل cache.cfg

۴.۱.۱ اجرای شبیه سازی

پس از ذخیره تنظیمات با دستور زیر، شبیه سازی را اجرا می کنیم:

```
$ ./cacti -infile cache.cfg
```

شبیه سازی با موفقیت انجام می شود و خروجی های تحلیل را به صورت «شکل ۴» گزارش می دهد.
درون این مخزن، تنظیمات دیگری همچون:

۱. تنظیمات DRAM

۲. تنظیمات حافظه DDR۳

۳. و ...

نیز وجود دارد که می توان آن ها را نیز همانند دستور قبل شبیه سازی کرد.

۲.۱ مزایا

۱. عام منظوره بودن:

شبیه ساز CACTI تمام اجزای یک حافظه را تحلیل و شبیه سازی می کند و صرفاً مختص به حافظه های نهان نیست. همچنین تخمینی از توان مصرفی مدار نیز ارائه می دهد که طراح می تواند با توجه به آن طراحی خود را از نظر توان مصرفی بهینه سازی کند.

```

reza@r324: /mnt/9636D17436D15639/University/CE/Memory Technologies/HWs/Simulation/HW01/Doc/Tools/CACTI$ ./cacti -infile cache.cfg
Cache size           : 131072
Block size           : 64
Associativity         : 2
Read only ports       : 0
Write only ports      : 0
Read write ports      : 1
Single ended read ports : 0
Cache banks (UCA)     : 1
Technology            : 0.09
Temperature           : 360
Tag size              : 42
Array type            : Cache
Model as memory        : 0
Model as 3D memory    : 0
Access mode           : 0
Data array cell type   : 0
Data array peripheral type : 0
Tag array cell type    : 0
Tag array peripheral type : 0
Optimization target    : 2
Design objective (UCA wt) : 0 0 0 100 0
Design objective (UCA dev) : 20 100000 100000 100000 100000
Cache model            : 0
Nuca bank              : 0
Wire inside mat        : 1
Wire outside mat       : 1
Interconnect projection : 1
Wire signaling          : 1
Print level            : 1
ECC overhead           : 1
Page size              : 8192
Burst length           : 8
Internal prefetch width : 8
Force cache config      : 0
Subarray Driver direction : 1
Iostate                : WRITE
dram_ecc                : NO_ECC
Io.type                 : DDR3
dram_dimm               : UDIMM
IO Area (sq.mm) = inf
IO Timing Margin (ps) = -14.1667
IO Voltage Margin (V) = 0.155
IO Dynamic Power (mW) = 1506.36 PHY Power (mW) = 232.752 PHY Wakeup Time (us) = 27.503
IO Termination and Bias Power (mW) = 2505.96

```

شکل ۴: خروجی شبیه سازی

۲. سرعت بالا:

این شبیه ساز محاسبات تخمینی مناسبی از مصرف انرژی حافظه، مساحت مصرفی و تاخیر ها برای پیکربندی های ۲۰ مختلف ارائه می کند که همین محاسبات تخمینی باعث افزایش سرعت در انجام شبیه سازی می شود و دید مناسبی به طراح برای ادامه روند طراحی می دهد.

۳. انعطاف پذیری بالا در شخصی سازی:

طراح می تواند بسته به نیاز و طراحی خود، به تمام قسمت های حافظه دسترسی داشته باشد و آنها را متناسب با نیاز خود تغییر دهد.

۳.۱ معایب

۱. عدم انجام محاسبات دقیق:

شبیه ساز CACTI برای افزایش سرعت شبیه سازی و جلوگیری از پیچیده شدن مدل ارائه شده به آن، یک سری ساده سازی ها انجام می دهد که همین امر باعث شده است که این ابزار صرفاً در محیط اکادمیک مورد استفاده قرار گیرد و استفاده آنچنانی ای در صنعت نداشته باشد.

۲. موازی نبودن روند دنیای حافظه ها و امکانات شبیه ساز

عدم توانایی و تحلیل حافظه های نو ظهور مانند DDR۵ و RERAM و CAM

۳. بلادرنگ^{۲۰} نبودن:

اگر در زمان کار یک حافظه بخواهیم تغییراتی در آن انجام دهیم و همانجا تغییرات ناشی از آن را مشاهده کنیم. این امر با شبیه ساز CACTI امکان پذیر نمی باشد.

^{۲۰}Configurations
^{۲۱}Real-Time

۴. عدم وجود Community گسترده و قوی:

متأسفانه برای آموزش و تحلیل این شبیه‌ساز برای مبتدیان منابع و Community بسیار محدودی وجود دارد که می‌تواند باعث سردرگمی شود.

۲ شبیه‌ساز NVSIM

شبیه‌ساز NVSIM ابزاری برای تحلیل و شبیه‌سازی حافظه‌های غیرفرار^{۲۲} است که عمدتاً برای تحلیل و تخمین مساحت، توان و انرژی مصرفی در حافظه‌های غیر فرار استفاده می‌شود.

برخلاف شبیه‌ساز CACTI، شبیه‌ساز NVSIM از شبیه‌سازی و تحلیل حافظه‌های نوظهور هم پشتیبانی می‌کند. حافظه‌هایی که در NVSIM قابلیت تحلیل دارند را می‌توان به صورت زیر دسته‌بندی کرد:

1. NAND Flash
2. SRAM
3. DRAM
4. PCM (Phase Change Memory)
5. STT RAM (Spin Torque Transfer RAM)
6. ReRAM (Resistive RAM)
7. FBDRAM (Floating Body Dynamic RAM)
8. eDRAM

شبیه‌ساز NVSIM از اصول طراحی مشابه با CACTI استفاده می‌کند اما برخلاف CACTI انعطاف پذیری بیشتری در سازماندهی بانک‌های حافظه دارد. چنین انعطاف پذیری‌ای برای حافظه‌های غیر فرار نوظهور ضروری است چرا که وضعیت اکثر حافظه‌های نوظهور نا شناخته است و این انعطاف پذیری دست طراح را برای بررسی بیشتر بخش‌های مختلف حافظه باز می‌کند.

همانند CACTI این شبیه‌ساز نیز با زبان ++C نوشته شده است و فایل کانفیگ حافظه را در دو فرمت .cfg و .cell دریافت می‌کند.

شبیه‌ساز NVSIM از داده‌های ITRS برای مدل‌های خود استفاده می‌کند. این ابزار فناوری‌های ۱۸۰nm، ۹۰nm، ۶۵nm، ۴۵nm و ۳۲nm را پوشش می‌دهد و از ترانزیستور کارایی بالا^{۲۳}، توان عملیاتی کم^{۲۴} و Low stand-by power استفاده می‌کند.

«شکل ۵» سازماندهی سطوح حافظه را در NVSIM نشان می‌دهد.

همانطور که مشخص است، سلسله مراتب حافظه در NVSIM از سه سطح بانک^{۲۵}، مت^{۲۶} و زیرآرایه^{۲۷} تشکیل می‌شود.

^{۲۲} Non-volatile

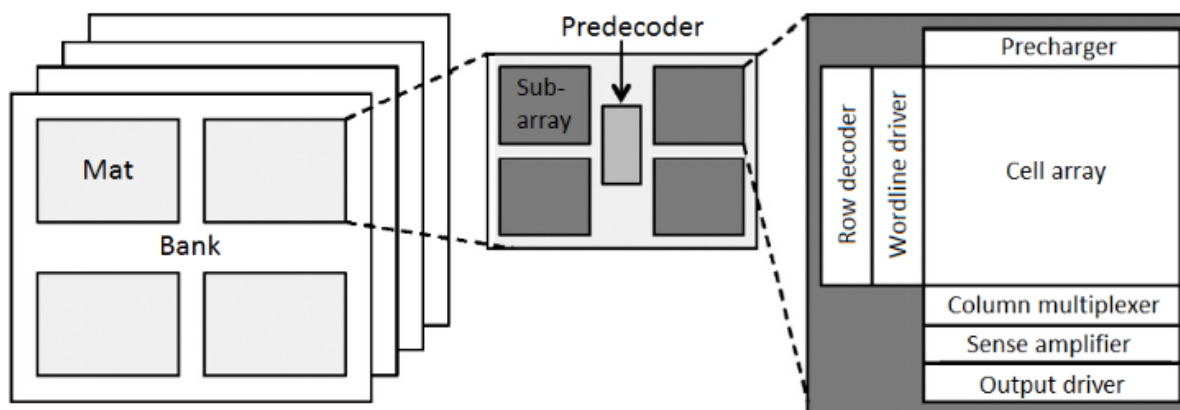
^{۲۳} High performance

^{۲۴} Low power operation

^{۲۵} Bank

^{۲۶} Mat

^{۲۷} Sub array



شکل ۵: سازماندهی سطوح حافظه در NVSIM

بانک ساختار سطح بالاییست که در NVSIM مدل شده است. یک تراشه حافظه غیر فرار می تواند چندین بانک داشته باشد. بانک یک واحد حافظه کاملاً مهم و کاربردی است و می توان آن را به طور مستقل اداره کرد. در هر بانک سلول ها در ساختار H-tree و یا با گذرگاه^{۲۸} هایی به هم متصل شده اند. هر مت از چندین زیر آرایه و یک بلوک پیش رمزگشا^{۲۹} تشکیل می شود.

شبیه ساز NVSIM سه نوع طراحی حافظه را مدل سازی می کند:

1. RAM
2. Set associative cache
3. CAM

برای دانلود و نصب این شبیه ساز می توان به صورت زیر عمل کرد:

۱.۲ دانلود و نصب

ابتدا می بایست مخزن NVSIM را دریافت کنیم. این کار را می توان به صورت زیر انجام داد:

```
$ git clone https://github.com/lpentecost/nvsim-merged
```

پس از دریافت کامل NVSIM با دستور زیر وارد فایل مخزن می شویم:

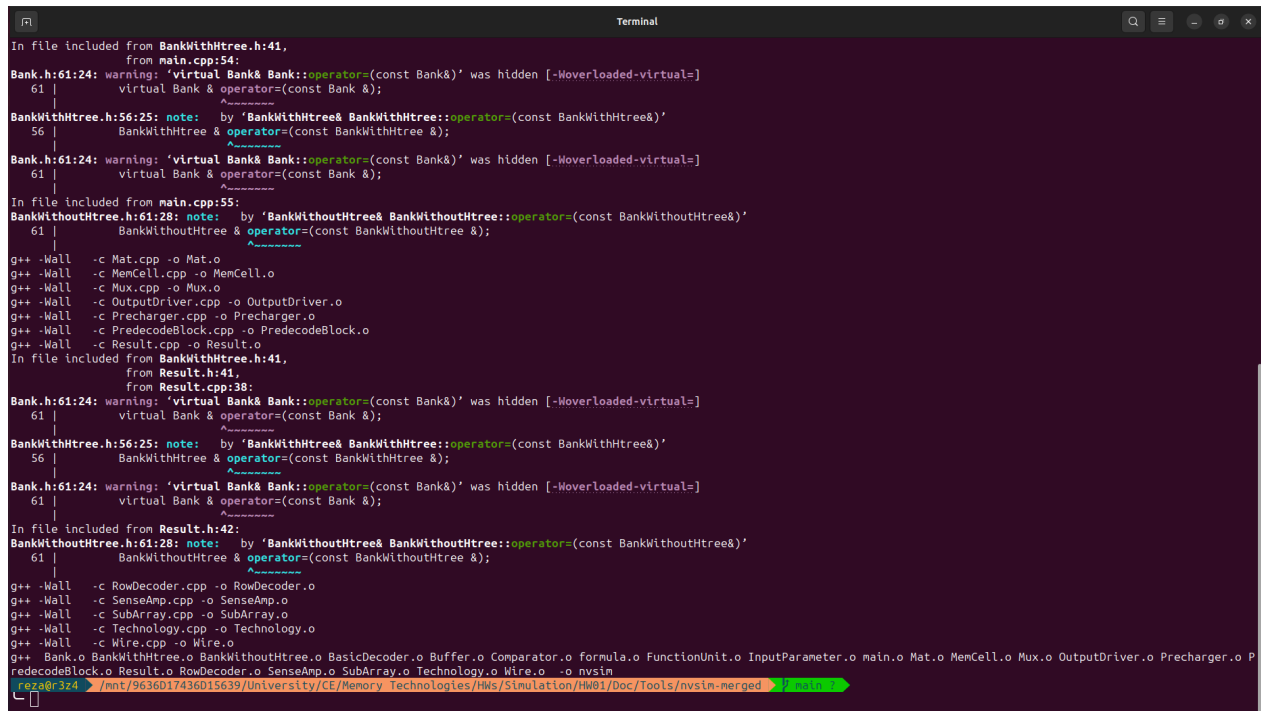
```
$ cd nvsim-merged
```

^{۲۸} bus
^{۲۹} Pre decoder

حال می‌بایست شبیه ساز را بیلد کنیم:

\$ make

اگر شبیه‌ساز به درستی بیلد شود، خروجی می‌بایست به صورت زیر شود:



شکل ۶: بیلد موفق NVSIM

۲.۲ تنظیمات فایل کانفیگ

همانند CACTI می‌بایست فایل کانفیگ را به شبیه‌ساز بدهیم. برای انجام این کار از فایل کانفیگ sample.cfg که به صورت پیش‌فرض وجود دارد استفاده می‌کنیم. این فایل، فایل کانفیگ یک Memristor ۶۴ بیتی است. محتویات درون فایل sample.cfg به صورت زیر است: «شکل ۷»

۳.۲ اجرای شبیه‌سازی

پس از تنظیمات کانفیگ با دستور زیر می‌توان شبیه‌سازی را انجام داد:

\$./nvsim sample.cfg

به جای sample.cfg می‌توان هر فایل کانفیگ دیگری را قرار داد. دقایقی طول خواهد کشید تا شبیه‌سازی به اتمام برسد. پس از تمام شدن شبیه‌سازی خروجی آن به صورت زیر خواهد بود: «شکل ۸»

```
DesignTarget: RAM
-CacheAccessMode: Normal
-OptimizationTarget: ReadEDP
-OutputFilePrefix: test
-EnablePruning: Yes
-ProcessNode: 22
-Capacity (MB): 1
-WordWidth (bit): 64
-DeviceRoadmap: HP
-LocalWireType: LocalAggressive
-LocalWireRepeaterType: RepeatedNone
-LocalWireUseLowSwing: No
-GlobalWireType: GlobalAggressive
-GlobalWireRepeaterType: RepeatedNone
-GlobalWireUseLowSwing: No
-Routing: H-tree
-InternalSensing: true
-MemoryCellInputFile: sample_cells/sample_RRAM.cell
-Temperature (K): 350
-BufferDesignOptimization: latency
//ForceBank (Total Ax8, Active CxD): 4x4, 1x4
//ForceMat (Total Ax8, Active CxD): 2x2, 1x2
//ForceMuxSenseAmp: 2
//ForceMuxOutputLev1: 1
//ForceMuxOutputLev2: 1
//UseCactiAssumption: Yes
//ApplyReadLatencyConstraint: 37.4
//ApplyWriteLatencyConstraint: 0.5
//ApplyReadDynamicEnergyConstraint: 0.5
"sample.cfg" 52L, 1021B
```

شکل ۷: فایل کانفیگ sample.cfg

همانطور که از «شکل ۸» مشخص است، خروجی شبیه سازی شامل تحلیلی از مساحت^{۳۰}، زمان بندی^{۳۱} و توان مصرفی حافظه است.

۴.۲ مزایا

۱. امکان شبیه سازی حافظه های جدید:
شبیه ساز NVSIM قادر به شبیه سازی و تحلیل حافظه های جدید و نو ظهوری مثل ReRAM، STT RAM و ... می باشد.
۲. قابلیت تغییر و شخصی سازی در سطوح پایین:
شبیه ساز NVSIM با طراح این امکان و دسترسی را می دهد تا تغییرات خود را در سطح بانک ها و سلول های حافظه اعمال کند.
۳. متن باز بود:
شبیه ساز NVSIM متن باز است و هرکس می تواند آن را متناسب با نیاز های خود توسعه دهد.

۵.۲ معایب

۱. بلادرنگ نبودن:
اگر در زمان کار یک حافظه بخواهیم تغییراتی در آن انجام دهیم و همانجا تغییرات ناشی از آن را مشاهده کنیم. این امر با شبیه ساز NVSIM امکان پذیر نمی باشد.

```
Terminal
|--- Predecoder Latency = 64.395ps
|--- Subarray Latency = 390.647ps
|--- Row Decoder Latency = 225.527ps
|--- Bitline Latency = 2.238ps
|--- Senseamp Latency = 100.000ps
|--- Mux Latency = 13.026ps
|--- Precharge Latency = 114.350ps
- Write Latency = 20.418ns
|--- H-Tree Latency = 5.344ps
|--- Mat Latency = 20.413ns
|--- Predecoder Latency = 64.395ps
|--- Subarray Latency = 20.348ns
|--- Row Decoder Latency = 225.527ps
|--- Charge Latency = 36.438ps
- Subarray Buf R/W Latency = 24.000ps
- Subarray Buf XOR Latency = 210.000ps
- Read Bandwidth = 28.626GB/s
- Write Bandwidth = 393.154MB/s
Power:
- Read Dynamic Energy = 4.252pJ
|--- H-Tree Read Dynamic Energy = 2.717pJ
|--- Mat Dynamic Energy = 1.536pJ per mat
|--- Predecoder Dynamic Energy = 0.060pJ
|--- Subarray Dynamic Energy = 1.476pJ per active subarray
|--- Row Decoder Dynamic Energy = 0.104pJ
|--- Mux Decoder Dynamic Energy = 0.395pJ
|--- Bitline & Cell Read Energy = 0.002pJ
|--- Senseamp Dynamic Energy = 0.512pJ
|--- Mux Dynamic Energy = 0.074pJ
|--- Precharge Dynamic Energy = 0.389pJ
- Write Dynamic Energy = 45.549pJ
|--- H-Tree Write Dynamic Energy = 2.717pJ
|--- Mat Dynamic Energy = 42.832pJ per mat
|--- Predecoder Dynamic Energy = 0.060pJ
|--- Subarray Dynamic Energy = 42.773pJ per active subarray
|--- Row Decoder Dynamic Energy = 0.104pJ
|--- Mux Decoder Dynamic Energy = 0.395pJ
|--- Mux Dynamic Energy = 0.074pJ
- Subarray Buf R/W Energy = 0.000pJ
- Subarray Buf XOR Energy = 0.000pJ
- Leakage Power = 93.185mW
|--- H-Tree Leakage Power = 0.000pW
|--- Mat Leakage Power = 5.824mW per mat
Finished!
reza@r324: /mnt/9636D17436D15639/University/CE/Memory Technologies/HMs/Simulation/HW01/Doc/Tools/nvsim-merged
```

شکل ۸: خروجی شبیه سازی

۲. عدم وجود نسخه رسمی:

تمام نسخه های موجود در اینترنت نسخه های Modify شده هستند. و یک نسخه اصلی از شرکت سازنده موجود نیست.

۳ شبیه ساز DRAMSIM

مدلسازی و شبیه سازی دقیق حافظه های پویا^{۳۲} از این جهت مهم است که تکنولوژی سعی بر آن دارد که CPU و DRAM را به صورت مجتمع در یک تراشه ارائه دهد. این کار تراکم بالا، عملکرد بهینه و همچنین مصرف انرژی کمتری را برای DRAM ها فراهم می کند.

به همین دلیل است که به ابزاری دقیق و با قابلیت اطمینان بالا برای مدلسازی و شبیه سازی DRAM ها نیاز داریم. شبیه ساز DRAMSIM شبیه سازی است که دقیقاً با همین هدف طراحی شده است. این شبیه ساز ورژن های مختلفی مثل DRAMSIM^۱، DRAMSIM^۲ و DRAMSIM^۳ دارد که در این گزارش به بررسی آخرین ورژن آن یعنی DRAMSIM^۳ می پردازیم.

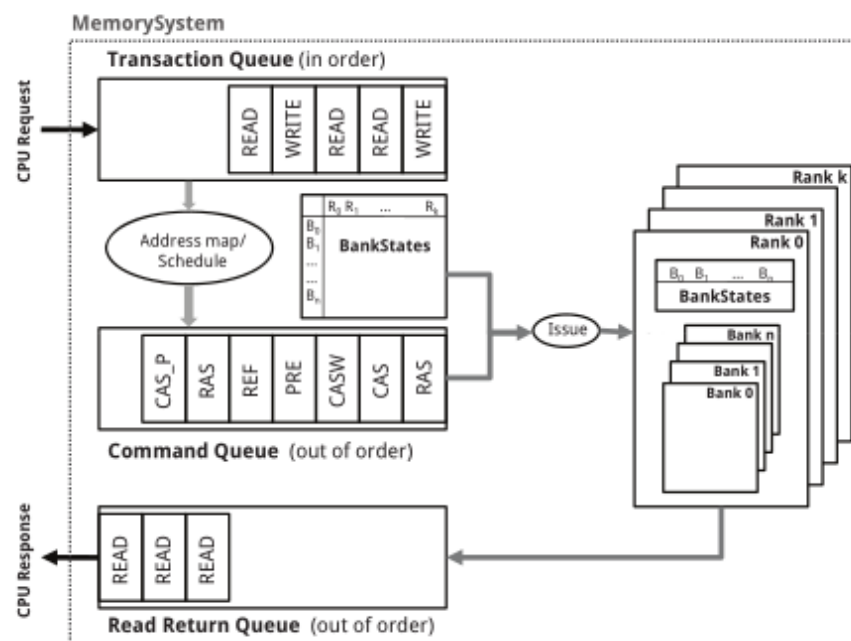
شبیه ساز DRAMSIM به صورت کاملاً ماژولار نوشته شده است. ماژولار بودن این امکان را به توسعه دهندگان می دهد که با پیشرفت فناوری، ویژگی های جدید را به DRAMSIM اضافه نمایند. همچنین می توان این شبیه ساز را با شبیه سازهای رایج CPU همانند GEM^۵ به عنوان شبیه ساز حافظه، ادغام کرد.

^{۳۲}Dynamic

این شبیه ساز نیز همانند سایر شبیه ساز های بررسی شده به زبان C++ توسعه داده شده است و می تواند پروتوکل های DRAM زیر را مدلسازی کند:

1. DDR3
2. DDR4
3. LPDDR3
4. LPDDR4
5. GDDR5
6. GDDR6
7. HBM
8. HMC
9. STT-MRAM

ساختار یک بلوک اصلی DRAMSIM در «شکل ۹» آورده شده است.



شکل ۹: ساختار یک بلوک از شبیه ساز DRAMSIM

برای دانلود و نصب این شبیه ساز می توان به صورت زیر عمل کرد:

۱.۳ دانلود و نصب

با دستور زیر مخزن شبیه ساز را دریافت می کنیم و به مسیر مخزن می رویم:

```
$ git clone https://github.com/umd-memsys/DRAMsim3
$ cd DRAMsim3
```

۲.۳ بیلد شبیه ساز

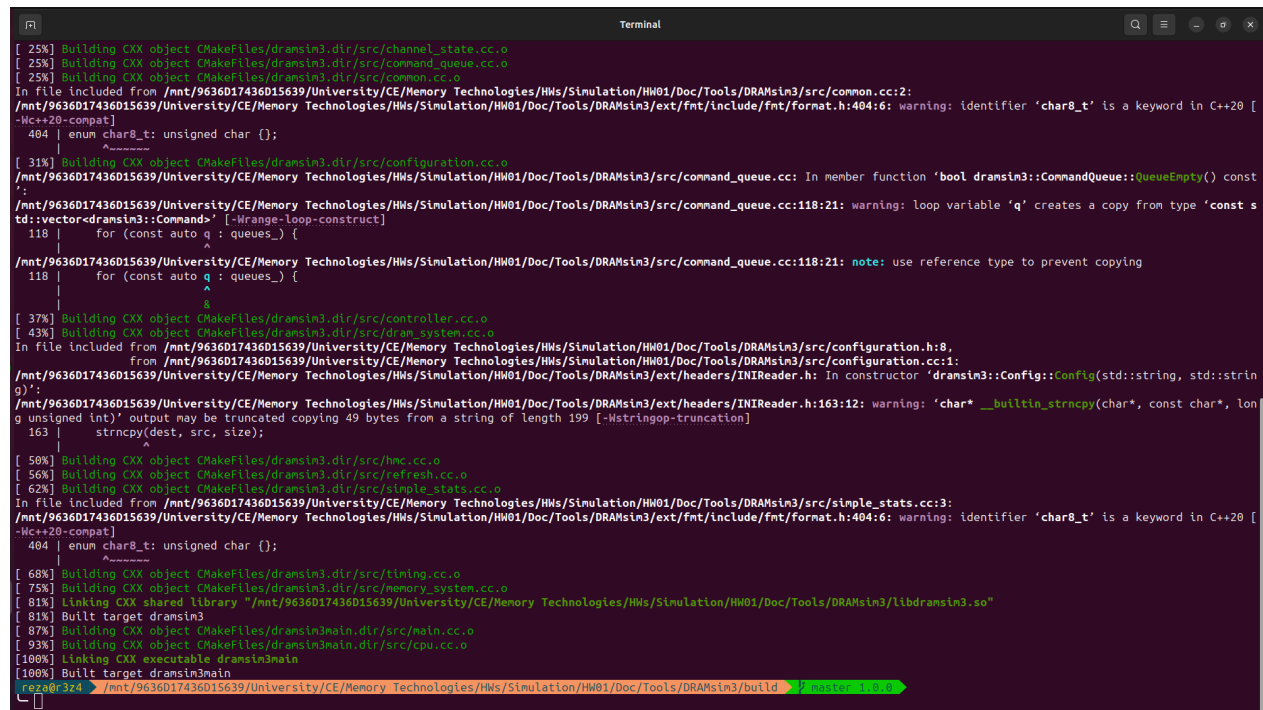
ابتدا با دستور زیر، یک دایرکتوری برای بیلد می سازیم:

```
$ mkdir build
```

سپس به درون دایرکتوری ایجاد شده می رویم و شبیه ساز را بیلد می کنیم:

```
$ cd build
$ cmake ..
$ make -j4
$ cmake .. -D THERMAL=1
```

اگر شبیه ساز به درستی بیلد شود، می بایست همانند «شکل ۱۰» پیغام Built target بر روی ترمینال مشاهده شود.



```
[ 25%] Building CXX object CMakeFiles/drams3.dir/src/channel_state.cc.o
[ 25%] Building CXX object CMakeFiles/drams3.dir/src/command_queue.cc.o
[ 25%] Building CXX object CMakeFiles/drams3.dir/src/common.cc.o
In file included from /mnt/9636017436015639/University/CE/Memory Technologies/HMs/Simulation/HW01/Doc/Tools/DRAMsim3/src/common.cc:2:
/mnt/9636017436015639/University/CE/Memory Technologies/HMs/Simulation/HW01/Doc/Tools/DRAMsim3/ext/fmt/include/fmt/format.h:404:6: warning: identifier 'char8_t' is a keyword in C++20 [-Wc++20-compat]
  404 | enum char8_t: unsigned char {};
      | ~~~~~
[ 31%] Building CXX object CMakeFiles/drams3.dir/src/configuration.cc.o
/mnt/9636017436015639/University/CE/Memory Technologies/HMs/Simulation/HW01/Doc/Tools/DRAMsim3/src/command_queue.cc:118:21: warning: loop variable 'q' creates a copy from type 'const std::vector<drams3::Command>' [-Wrange-loop-construct]
  118 |     for (const auto q : queues_) {
      |           ^
/mnt/9636017436015639/University/CE/Memory Technologies/HMs/Simulation/HW01/Doc/Tools/DRAMsim3/src/command_queue.cc:118:21: note: use reference type to prevent copying
  118 |     for (const auto q : queues_) {
      |           ^
[ 37%] Building CXX object CMakeFiles/drams3.dir/src/controller.cc.o
[ 43%] Building CXX object CMakeFiles/drams3.dir/src/dram_system.cc.o
In file included from /mnt/9636017436015639/University/CE/Memory Technologies/HMs/Simulation/HW01/Doc/Tools/DRAMsim3/src/configuration.h:8,
                  from /mnt/9636017436015639/University/CE/Memory Technologies/HMs/Simulation/HW01/Doc/Tools/DRAMsim3/src/configuration.cc:1:
/mnt/9636017436015639/University/CE/Memory Technologies/HMs/Simulation/HW01/Doc/Tools/DRAMsim3/ext/headers/INIReader.h: In constructor 'drams3::Config::Config(std::string, std::string)':
/mnt/9636017436015639/University/CE/Memory Technologies/HMs/Simulation/HW01/Doc/Tools/DRAMsim3/ext/headers/INIReader.h:163:12: warning: 'char* __builtin_strncpy(char*, const char*, long unsigned int)' output may be truncated copying 49 bytes from a string of length 199 [-Wstringop-truncation]
  163 |     strncpy(dest, src, size);
      |     ^~~~~~
[ 50%] Building CXX object CMakeFiles/drams3.dir/src/hmc.cc.o
[ 56%] Building CXX object CMakeFiles/drams3.dir/src/refresh.cc.o
[ 62%] Building CXX object CMakeFiles/drams3.dir/src/simple_stats.cc.o
In file included from /mnt/9636017436015639/University/CE/Memory Technologies/HMs/Simulation/HW01/Doc/Tools/DRAMsim3/src/simple_stats.cc:3:
/mnt/9636017436015639/University/CE/Memory Technologies/HMs/Simulation/HW01/Doc/Tools/DRAMsim3/ext/fmt/include/fmt/format.h:404:6: warning: identifier 'char8_t' is a keyword in C++20 [-Wc++20-compat]
  404 | enum char8_t: unsigned char {};
      | ~~~~~
[ 68%] Building CXX object CMakeFiles/drams3.dir/src/timing.cc.o
[ 75%] Building CXX object CMakeFiles/drams3.dir/src/memory_system.cc.o
[ 81%] Linking CXX shared library "/mnt/9636017436015639/University/CE/Memory Technologies/HMs/Simulation/HW01/Doc/Tools/DRAMsim3/libdrams3.so"
[ 81%] Built target dramsim3
[ 87%] Building CXX object CMakeFiles/drams3main.dir/src/main.cc.o
[ 93%] Building CXX object CMakeFiles/drams3main.dir/src/cpu.cc.o
[100%] Linking CXX executable drams3main
[100%] Built target drams3main
reza@0324 ~$ /mnt/9636017436015639/University/CE/Memory Technologies/HMs/Simulation/HW01/Doc/Tools/DRAMsim3/build >> make -j4 -B
```

شکل ۱۰: بیلد موفق شبیه ساز DRAMSIM

۳.۳ اجرای شبیه سازی

ابتدا یک پوشه برای ذخیره فایل های خروجی شبیه سازی ایجاد می کنیم:

```
$ mkdir output
```

سپس با دستور زیر شبیه سازی را برای فایل کانفیگ sample_trace.txt انجام می دهیم.

```
$ ./build/dramsim3main configs/DDR4_8Gb_x8_3200.ini -c 100000 -t  
tests/example_trace.txt -o output/
```

درون مسیر configs/ فایل های کانفیگ مختلفی وجود دارد که برای اجرای این برنامه از کانفیگ DDR4_8Gb استفاده کرده ایم.

شبیه ساز DRAMSIM این قابلیت را به ما می دهد که به صورت نموداری هم خروجی بگیریم از شبیه ساز. برنامه های این نمودار ها به زبان پایتون نوشته شده است که مسیر script/ وجود دارد. نمودار های تاخیر^{۳۳} ورودی و خروجی ها برای شبیه سازی انجام شده به صورت زیر گزارش شده است:

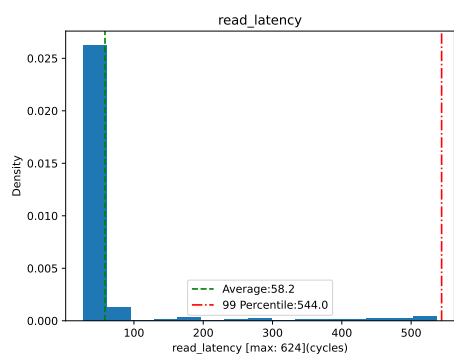
۴.۳ مزایا

۱. امکان شبیه سازی تکنولوژی های جدید DRAM:
امکان مدل سازی تکنولوژی های جدید مثل DDR4 و GDDR6 را دارد.
۲. انعطاف پذیری بالا در پیکربندی:
به طراحان اجازه می دهد تا پارامترهای مختلف مرتبط با DRAM مانند پارامترهای زمان بندی، ویژگی های دستگاه و ویژگی های معماری را پیکربندی کنند.
۳. قابلیت همگام سازی با شبیه ساز های سیستمی
می توان از DRAMSIM در شبیه ساز های سیستمی مانند GEM5 به عنوان شبیه ساز DRAM استفاده کرد.

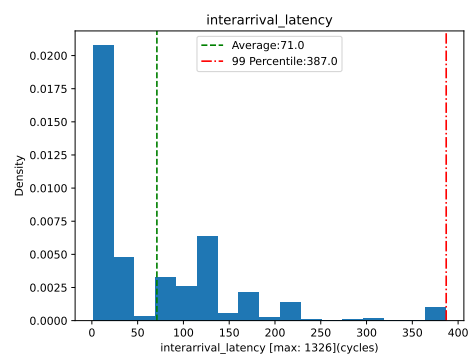
۵.۳ معایب

۱. وابستگی به مدل:
خروجی شبیه سازی کاملاً وابسته به مدل و کانفیگ انتخاب شده برای DRAM است.
۲. عدم گزارش توان و مساحت مصرفی:
این شبیه ساز، گزارشی از توان مصرفی حافظه مدل شده را ارائه نمی دهد.

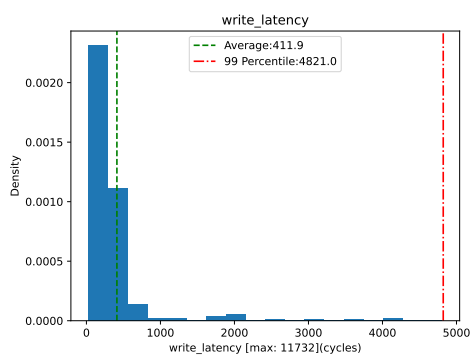
^{۳۳}latency



(ب) نمودار Read latency



(آ) نمودار Interarrival latency



(ج) نمودار Write latency

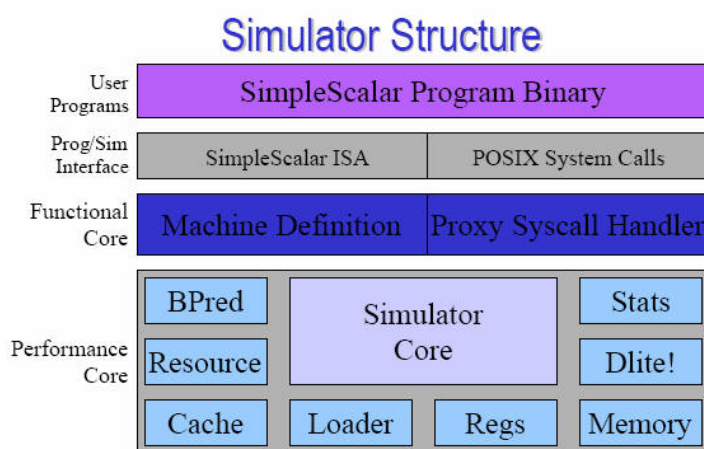
شکل ۱۱: نمودارهای خروجی شبیه‌سازی

۴ شبیه ساز SimpleScaler

این شبیه ساز حاصل رساله دکتری آقای آستین^{۳۴} از دانشگاه ویسکانسین^{۳۵} است که به زبان C نوشته شده است. شبیه ساز SimpleScaler صرفاً مخصوص شبیه سازی حافظه ها نیست. بلکه مانند Gem5 تمام اجزای یک سیستم کامپیوتری را مدل و شبیه سازی می کند و کاربر می تواند برنامه ای مخصوص به خود را بنویسد و با مشخص کردن اجزای سیستم، آن را بر روی سیستم مدل شده اجرا کند.

این شبیه ساز به صورت پیش فرض قادر به شبیه سازی ISA های Alpha و PISA^{۳۶} است اما این امکان را به کاربر می دهد که ISA های دیگر را هم اضافه کند.

ساختار این شبیه ساز در «شکل ۱۲» آورده شده است.



شکل ۱۲: ساختار SimpleScaler

شبیه ساز SimpleScaler شامل مجموعه ای از زیرشبیه ساز^{۳۷} هاست که برای شبیه سازی ریزمعماری^{۳۸} های مختلف استفاده می شوند. عبارت اند از:

۱. Sim-fast:

مفسر سریع دستورالعمل هاست که بدون در نظر گرفتن رفتار خطوط لوله^{۳۹} و حافظه پنهان یا هر بخش دیگری از ریزمعماری، شبیه سازی را انجام می دهد.

۲. Sim-safe:

این حالت مقداری از حالت قبل کند تر است، زیرا دسترسی به حافظه ها را در شبیه سازی در نظر می گیرد.

۳. Sim-profile:

این شبیه ساز تعداد شبیه ساز تعداد دستورالعمل های پویا و تعداد کلاس های دستورالعمل ها را گزارش می کند.

^{۳۴}Austin Todd

^{۳۵}University of Wisconsin Madison

^{۳۶}Portable ISA

^{۳۷}Sub simulator

^{۳۸}Micro architecture

^{۳۹}Pipeline

۴. Sim-cache:

این شبیه‌ساز سیستمی با حافظه نهان را شبیه‌سازی می‌کند که می‌توان حافظه های نهان را به صورت دلخواه تنظیم کرد.

۵. Sim-bpred:

این شبیه‌ساز شاخه ۴۰ های برنامه را پیشینی و گزارش می‌کند.

۶. Sim-outorder:

تمام ویژگی های قبلی در این شبیه ساز وجود دارد.

۱.۴ دانلود و نصب

شبیه‌ساز را به صورت زیر دانلود می‌کنیم:

```
$ git clone https://github.com/stevekuznetsov/simple-scalar.git  
$ cd simple-scalar
```

۲.۴ نصب پیشنیازها

پیشنیاز ها را به صورت زیر نصب می‌کنیم:

```
$ sudo apt-get update  
$ sudo apt-get install build-essential  
$ sudo apt-get install flex bison  
$ sudo apt-get install libx11-dev
```

۳.۴ بیلد شبیه‌ساز

پس از نصب پیش‌نیاز ها، شبیه‌ساز را به صورت زیر بیلد می‌کنیم:

```
$ make config-alpha  
$ make
```

اگر شبیه‌ساز به درستی بلد شود، خروجی ترمینال می‌بایست به صورت «شکل ۱۳» شود.

۴.۴ اجرای برنامه

فایل exe برنامه های پیش‌فرض شبیه‌ساز در مسیر زیر قرار دارند:

```
Terminal
machine.o libexo/libexo.a `./sysprobe -libs' -lm
gcc -o ./sysprobe -flags -DDEBUG -O0 -g -Wall -c sim-eio.c
gcc -o sim-eio `./sysprobe -flags' -DDEBUG -O0 -g -Wall  sim-eio.o main.o syscall.o memory.o regs.o loader.o endian.o dlite.o symbol.o eval.o options.o stats.o eio.o range.o misc.o ma
chine.o libexo/libexo.a `./sysprobe -libs' -lm
gcc -o ./sysprobe -flags -DDEBUG -O0 -g -Wall -c sim-bpred.c
gcc -o ./sysprobe -flags -DDEBUG -O0 -g -Wall -c bpred.c
bpred.c: In function 'bpred_lookup':
bpred.c:1057:3: warning: enumeration value 'BPredTaken' not handled in switch [-Wswitch]
1057 |     switch (pred->class) {
      |           ^~~~~~
bpred.c:1057:3: warning: enumeration value 'BPredNotTaken' not handled in switch [-Wswitch]
bpred.c:1057:3: warning: enumeration value 'BPred_NUM' not handled in switch [-Wswitch]
bpred.c: In function 'bpred_update':
bpred.c:1416:3: warning: enumeration value 'BPredTaken' not handled in switch [-Wswitch]
1416 |     switch (pred->class) {
      |           ^~~~~~
bpred.c:1416:3: warning: enumeration value 'BPredNotTaken' not handled in switch [-Wswitch]
bpred.c:1416:3: warning: enumeration value 'BPred_NUM' not handled in switch [-Wswitch]
gcc -o sim-bpred `./sysprobe -flags' -DDEBUG -O0 -g -Wall  sim-bpred.o bpred.o main.o syscall.o memory.o regs.o loader.o endian.o dlite.o symbol.o eval.o options.o stats.o eio.o range
.o misc.o machine.o libexo/libexo.a `./sysprobe -libs' -lm
gcc -o ./sysprobe -flags -DDEBUG -O0 -g -Wall -c sim-profile.c
sim-profile.c: In function 'sim_main':
sim-profile.c:649:22: warning: variable 'fault' set but not used [-Wunused-but-set-variable]
649 |     enum md_fault_type fault;
      |                      ^~~~~~
gcc -o sim-profile `./sysprobe -flags' -DDEBUG -O0 -g -Wall  sim-profile.o main.o syscall.o memory.o regs.o loader.o endian.o dlite.o symbol.o eval.o options.o stats.o eio.o range.o m
isc.o machine.o libexo/libexo.a `./sysprobe -libs' -lm
gcc -o ./sysprobe -flags -DDEBUG -O0 -g -Wall -c sim-cache.c
gcc -o ./sysprobe -flags -DDEBUG -O0 -g -Wall -c cache.c
gcc -o sim-cache `./sysprobe -flags' -DDEBUG -O0 -g -Wall  sim-cache.o cache.o main.o syscall.o memory.o regs.o loader.o endian.o dlite.o symbol.o eval.o options.o stats.o eio.o range
.o misc.o machine.o libexo/libexo.a `./sysprobe -libs' -lm
gcc -o ./sysprobe -flags -DDEBUG -O0 -g -Wall -c sim-outorder.c
sim-outorder.c: In function 'ruw_dispatch':
sim-outorder.c:3855:7: warning: variable 'br_taken' set but not used [-Wunused-but-set-variable]
3855 |     int br_taken, br_pred_taken; /* if br, taken? predicted taken? */
      |       ^~~~~~
sim-outorder.c: In function 'sim_main':
sim-outorder.c:4586:17: warning: variable 'target_PC' set but not used [-Wunused-but-set-variable]
4586 |     md_addr_t target_PC; /* actual next/target PC address */
      |                 ^~~~~~
gcc -o ./sysprobe -flags -DDEBUG -O0 -g -Wall -c resource.c
gcc -o ./sysprobe -flags -DDEBUG -O0 -g -Wall -c ptrace.c
gcc -o sim-outorder `./sysprobe -flags' -DDEBUG -O0 -g -Wall  sim-outorder.o cache.o bpred.o resource.o ptrace.o main.o syscall.o memory.o regs.o loader.o endian.o dlite.o symbol.o ev
al.o options.o stats.o eio.o range.o misc.o machine.o libexo/libexo.a `./sysprobe -libs' -lm
my work is done here...
reza@324 ~$ cd /mnt/9636017436015639/University/CE/Memory Technologies/HWs/Simulation/HW01/Doc/Tools/simple-scalar & master
```

شکل ۱۳: بیلد موفق شبیه ساز SimpleScaler

```
$ tests/bin/
```

همچنین سورس کد برنامه را می توان از مسیر زیر بررسی کرد:

```
$ tests/src/
```

به عنوان نمونه، برنامه test-math را که چندین عمل ریاضی مثل توان، سینوس، تانژانت و ... را برای چندین ورودی حساب می کند را شبیه سازی و اجرا می کنیم. برای انجام این کار دستور زیر را اجرا می کنیم:

```
$ ./sim-safe tests/bin/test-math
```

خروجی شبیه سازی به صورت زیر گزارش می شود:

```
sim: ** starting functional simulation **
pow(12.0, 2.0) == 144.000000
pow(10.0, 3.0) == 1000.000000
pow(10.0, -3.0) == 0.001000
str: 123.456
x: 123.000000
str: 123.456
x: 123.456000
str: 123.456
x: 123.456000
123.456 123.456000 123 1000
sinh(2.0) = 3.62686
sinh(3.0) = 10.0179
h=3.60555
atan2(3.2) = 0.982794
pow(3.60555,4.0) = 169
169 / exp(0.982794 * 5) = 1.24102
3.93117 + 5*log(3.60555) = 10.3435
cos(10.3435) = -0.606798, sin(10.3435) = -0.794856
x      0.5x
x0.5   x
x      0.5x
-1e-17 == -1e-17 Worked!
warning: partially supported sigprocmask() call...

sim: ** simulation statistics **
sim_num_insn      49430 # total number of instructions executed
sim_num_refs      13640 # total number of loads and stores executed
sim_elapsed_time   1 # total simulation time in seconds
sim_inst_rate     49430.0000 # simulation speed (in insts/sec)
ld_text_base      0x0120000000 # program text (code) segment base
ld_text_size      180416 # program text (code) size in bytes
ld_data_base      0x0140000000 # program initialized data segment base
ld_data_size      41984 # program init'ed '.data' and uninit'ed '.bss' size in bytes
ld_stack_base     0x01ff9b000 # program stack segment base (highest address in stack)
ld_stack_size     16384 # program initial stack size
ld_prog_entry     0x012000f750 # program entry point (initial PC)
ld_environ_base   0x01ff97000 # program environment base address
ld_target_big_endian 0 # target executable endian-ness, non-zero if big endian
mem.page_count    29 # total number of pages allocated
mem.page_mem      232k # total size of memory pages allocated
mem.ptab_misses   75 # total first level page table misses
mem.ptab_accesses 535692 # total page table accesses
mem.ptab_miss_rate 0.0001 # first level page table miss rate
```

شکل ۱۴: خروجی اجرای برنامه test-math

۵.۴ مزایا

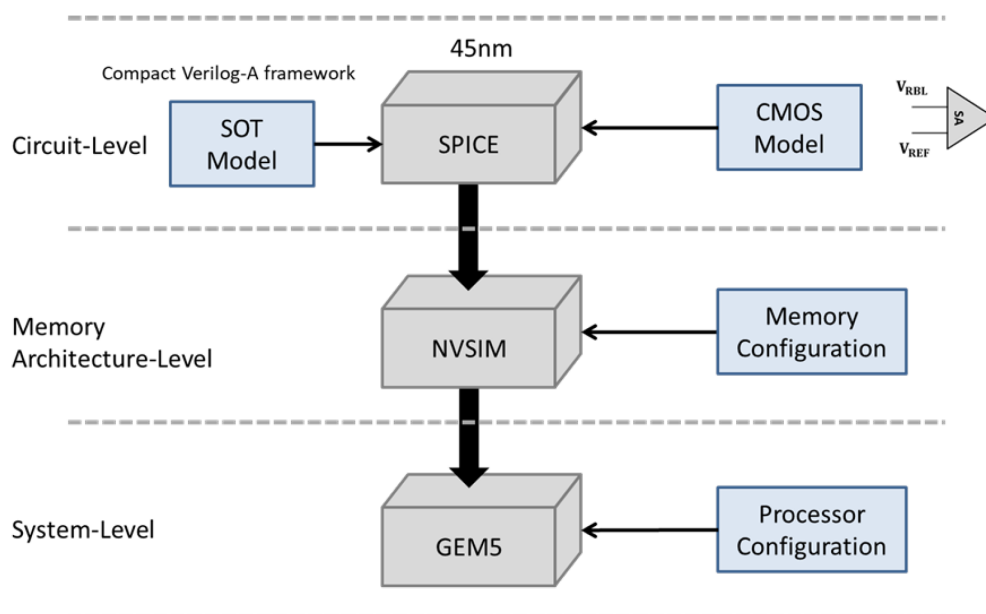
۱. متن باز بودن
این شبیه ساز نیز همانند سایر شبیه ساز های معرفی شده متن باز است.
۲. شبیه ساز کامل کامپیوتر
این شبیه ساز یک سیستم کامپیوتری را به صورت کامل شبیه سازی می کند و به کاربر این اجازه را می دهد که روند اجرای یک برنامه را از ابتدا کلاک به کلاک بررسی نماید.
۳. پشتیبانی از معماری های مختلف
این شبیه ساز از معماری ها و ISA های مختلف پشتیبانی می کند.

۶.۴ معایب

۱. عدم وجود دسترسی مستقیم به حافظه
نمی توان به پارامترهای مربوط به حافظه به صورت ریز و جزئی همانند شبیه ساز های معرفی شده قبلی دسترسی داشت.
۲. عدم پشتیبانی از حافظه های جدید
۳. عدم ارائه تحلیل مفصل از اجرای برنامه همانند GEM5
شبیه ساز SimpleScaler تحلیل کوتاه و مختصری از اجرای برنامه ارائه می دهد و این تحلیل بسیار کمتر از آنچه ای است که GEM5 ارائه می دهد.

۲. در پاسخ به سوال دوم اینگونه می توان گفت که، طراحی معماری پردازنده ای خاص، به طور کلی شامل سه مرحله می شود.

۱. طراحی در سطح سیستم
۲. طراحی معماری حافظه ها
۳. طراحی در سطح مدار



شکل ۱۵: مراحل طراحی معماری یک پردازنده

معمولا در ابتدای کار، طراحی در سطح بالا به صورت رفتاری^{۴۱} با ابزار ها و شبیه ساز هایی مثل GEM5 یا SimpleScaler انجام می شود. همچنین در این مرحله خوب است که طراحی انجام شده به وسیله زبان های توصیف سخت افزار مانند VHDL، Verilog و SystemC شبیه سازی شود تا از صحت عملکرد و سنتز پذیری طراحی مطمئن شویم.

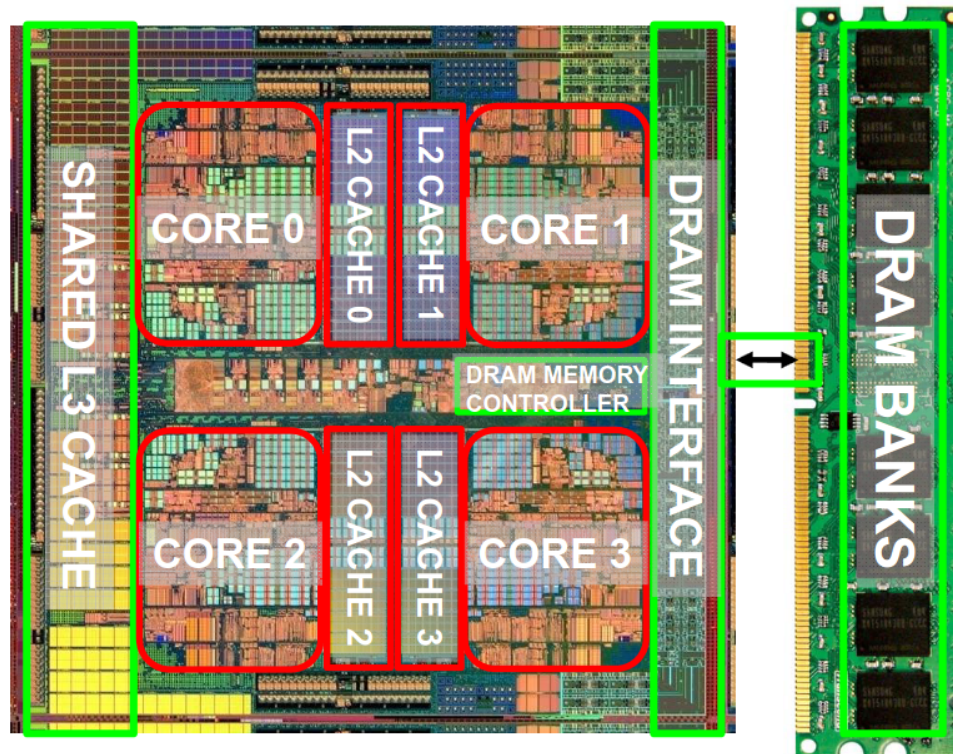
پس در این فاز از طراحی، شبیه ساز هایی که اینجانب به عنوان مدیر تیم به هم تیمی هایم پیشنهاد می دهم، شبیه ساز GEM5 و زبان توصیف سخت افزار SystemC است.

در فاز دوم طراحی، پس از شبیه سازی رفتاری پردازنده، می بایست سلسله مراتب حافظه پردازنده را طراحی کنیم. سلسله مراتب حافظه شامل طراحی و مدلسازی حافظه اصلی پردازنده و حافظه های پنهان است. پس طراحی حافظه را به دو دسته تقسیم می کنیم:

۱. طراحی cache ها
۲. طراحی حافظه اصلی

برای طراحی Cache ها از شبیه ساز CACTI استفاده می کنیم چرا که این شبیه ساز صرفا مختص به مدلسازی و طراحی سلسله مراتب Cache هاست و تحلیل دقیق و جامعی از پارامتر های Cache ارائه می دهد.

^{۴۱} Behavioral



شکل ۱۶: ساختار پردازنده AMD Barcelona

برای شبیه‌سازی و تحلیل حافظه اصلی مدار، به دلیل آنکه می‌بایست از حافظه‌های نو ظهور برای بهبود عملکرد پردازنده استفاده شود، شبیه‌ساز NVSIM را به تیمم پیشنهاد می‌دهم. چرا که این شبیه‌ساز جدیدترین و بروزترین حافظه‌ها را پشتیبانی می‌کند.

به دلیل آنکه صرفاً هدف ما طراحی پردازنده‌ست، از بررسی شبیه‌سازها برای طراحی حافظه‌های RAM جانبی خودداری می‌کنیم اما اگر می‌خواستیم این بخش را نیز طراحی کنیم، می‌توانیم از شبیه‌ساز DRAMSIM استفاده کنیم.

پس از تحلیل و طراحی سلسله مراتب حافظه با شبیه‌سازهای پیشنهاد شده CACTI و NVSIM و DRAM-SIM می‌بایست در فاز بعدی، هر دو قسمت قبل را در سطح مدار تحلیل و تست کنیم.

بهترین، دقیق‌ترین و کامل‌ترین ابزار برای شبیه‌سازی انواع مدارهای دیجیتال و آنالوگ شبیه‌ساز HSPICE است. این ابزار امکانات و تحلیل‌های بسیار جامعی از مدار با ما ارائه می‌کند. پس پیشنهاد بنده در این فاز، استفاده از شبیه‌ساز HSPICE است.

در فاز آخر اگر سه فاز قبلی را با موفقیت پشت سر گذاشته باشیم و بخواهیم پروسسور طراحی شده را بسازیم، می‌بایست عملیات Routing پروسسور را انجام دهیم و در نهایت مدل طراحی شده را به کارخانه‌های Fab-rication بدم تا آن را برای ما بسازند. در این فاز شبیه‌ساز Cadence را برای تحلیل نهایی، Placement و Routing به هم‌تیمی‌هایم پیشنهاد می‌دهم.

۳. یکی از ابزارهایی که برای شبیه سازی مدارهای دیجیتال و آنالوگ و عمدتاً بانک های حافظه به کار می رود، شبیه ساز HSPICE است. بر این اساس به سوالات زیر پاسخ دهید.

۱. دلایل ظهور شبیه ساز های دیگر توسعه یافته با زبان های C++ و Python و ... چیست؟

شاید این قیاس درستی نباشد که بگویم چرا شبیه ساز های سطح بالا توسعه داده شده است. این را از این جهت عرض می کنم که شبیه ساز HSPICE کاربرد بسیار گسترده ای در الکترونیک دارد و در این گزارش صرفاً بخش بسیار کوچکی از کاربردهای آن یعنی طراحی سلول های حافظه را بررسی کردیم. به همین دلیل توسعه شبیه ساز های جدید بدین معنا نیست که جایگزین HSPICE بشوند. عمدتاً شبیه ساز های جدید تمرکز بر روی افزایش سرعت تحلیل و افزایش راحتی رابط کاربری ابزار دارند. برای مثال شبیه ساز HSPICE سینتکس بسیار راحت اما بد قلقی دارد و زمان زیادی را میطلبت برای آنکه استفاده از آن برای کاربر راحت شود.

همچنین شبیه ساز HSPICE ابزاریست غیر رایگان و برای استفاده از آن می بایست لایسنس آن را از شرکت Synopsys خریداری کنیم^{۴۲} می توان گفت دلیل دیگر توسعه شبیه ساز های جدید م عمدتاً متن باز به همین دلیل است که استفاده و توسعه شبیه ساز راحت باشد. چرا که با ظهور تکنولوژی های جدید و حافظه های نو ظهور، مدت زیادی طول خواهد کشید تا شرکت Synopsys تنظیمات شبیه سازش را با تکنولوژی جدید بروز کند اما اگر ابزار متن باز وجود داشته باشد، فرآیند بروزرسانی و توسعه ابزار برای تکنولوژی جدید سریعتر انجام خواهد شد.

۲. در مورد اهمیت پیاده سازی با استفاده از ابزارهایی مانند HSPICE بحث کنید.

ابزار های خانواده SPICE مانند PSPICE، HSPICE، LTSPICE و Cadence عضو جدایی ناپذیر طراحی مدارهای الکترونیکی هستند. چرا که برای تحلیل های زمانی، فرکانسی، انرژی مصرفی و ... طراحی انجام شده به این ابزار ها نیاز داریم.

این ابزار ها به دلیل آنکه شرکت های معتبر و نامداری در این زمینه از آنها پشتیبانی می کنند، بسیار دقیق هستند و کوچکترین جزئیات را می توان با آنها شبیه سازی کرد.

شاید مهمترین دلیل استفاده از این ابزار ها را می توان به صورت زیر بیان کرد:
این ابزار ها مدارات را در پایین ترین سطح تجرید یعنی در سطح ترانزیستور شبیه سازی می کنند و تمام پدیده های غیر ایده آل ترانزیستور را مدل می کنند. این امر از این جهت اهمیت دارد که وقتی شبیه سازی را در سطح بالا و بدون در نظر گرفتن ریز ترین جزئیات انجام می دهیم همه چیز ایده آل است اما وقتی به سطوح تجرید پایین تر می آییم، اثرات غیر خطی قطعات، محدودیت های ساخت و ... بر روی طراحی انجام شده اثر می گذارند و می بایست طراحی را مجدداً با شرایط و امکانات موجود با این ابزار ها تحلیل و شبیه سازی کنیم. به عبارتی دیگر می توان گفت ابزار های نام برده نظیر HSPICE نزدیک ترین نتایج را نسبت به دنیای واقعی به طراحان گزارش می دهند.

^{۴۲} هر چند که ما در ایران آن را کرک می کنیم و به رایگان استفاده می کنیم، ولی خب (: