



«به نام خدا»

طراحی سیستم های نهفته مبتنی بر هسته

تکلیف کامپیوتری شماره ۳: یادگیری استفاده از ابزار TCE برای پیادهسازی روی پردازنده TTA

پردیس دانشکده های فنی دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

استاد: دکتر احمد شعبانی

نیمسال دوم سال تحصیلی ۱۴۰۰-۱۴۰۱

نگارش: علی ایمانقلی ۸۱۰۱۹۷۶۹۲

## Part1 → Installation → Question1

ورژن برنامه:

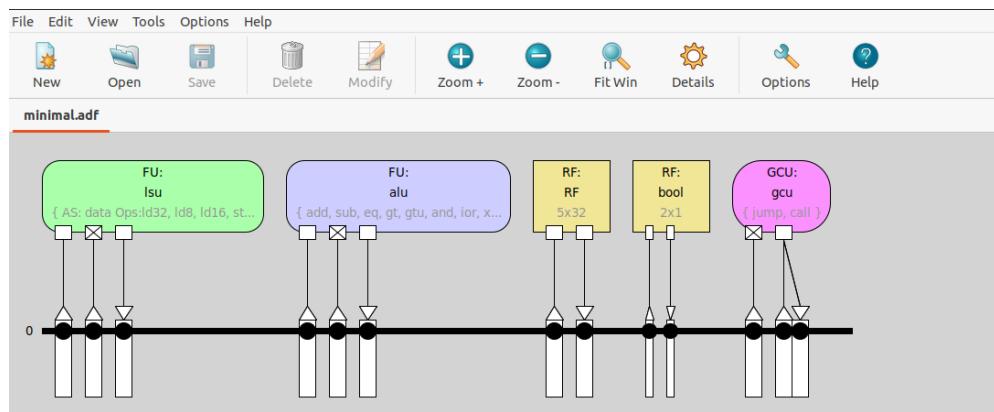
```
root@imangholi:/home/ali/Desktop/CA3-Ali-Imangholi-810197692/tce-devel/tce
root@imangholi:/home/ali/Desktop/CA3-Ali-Imangholi-810197692/tce-devel/tce# ttasim --version
ttasim - TCE Simulator command line interface 2.0-pre-r4204
root@imangholi:/home/ali/Desktop/CA3-Ali-Imangholi-810197692/tce-devel/tce#
```

اجرای ارزیابی درونی سیستم:  
خود برنامه تمامی قابلیت های خود را با اجرای smoke test ارزیابی می کند:

```
root@imangholi:/home/ali/Desktop/CA3-Ali-Imangholi-810197692/tce-devel/tce
root@imangholi:/home/ali/Desktop/CA3-Ali-Imangholi-810197692/tce-devel/tce# tce-selftest -v
Testing TCE installation, this can take up to two hours.
Time to go for a lunch? Use -v to see the progress.
test_noOpt (_main_.TestCompiler)
Test C compilation with -O0 ... ok
test_opt (_main_.TestCompiler)
Test C compilation with -O3 ... ok
test_custom_operation_from_c_code (_main_.TestCustomOps)
Test creating a simple custom operation with simulation code and ... ok
test_basic_plugins (_main_.TestExplorer)
Test the ImplementationSelector and Evaluate explorer plugins (this ... ok
test_default_plugins found (_main_.TestExplorer)
Check that the default Explorer plugins are listed. ... ok
test_proge_and_pig (_main_.TestProGeAndPig)
Test the processor generator and the program image generator (with ... ok
test_compiled_simulation (_main_.TestSimulator)
Test the compiled engine of the simulator ... ok
test_debugging_simulation (_main_.TestSimulator)
Test the debugging engine of the simulator ... ok
Ran 8 tests in 166.507s
OK
root@imangholi:/home/ali/Desktop/CA3-Ali-Imangholi-810197692/tce-devel/tce#
```

## Part 2 → Question2

سیستم مینیمال:

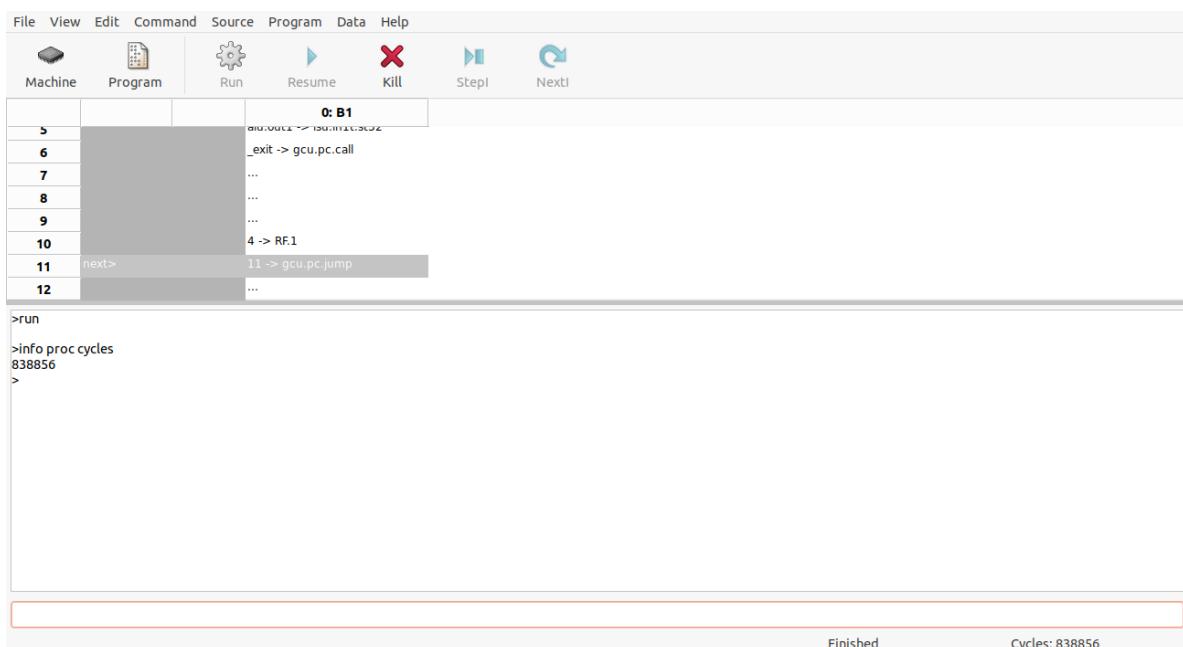


کامپایل کردن کد `bit_16_8x8_dct.c` در حالت `:minimal`

```
root@imangholi:/home/ali/Desktop/CA3-Ali-Imangholi-810197692/report/step-2/question-1/DCT# tcecc -O2 -a minimal.adf -o dct.tpef dct_8x8_16_bit.c
root@imangholi:/home/ali/Desktop/CA3-Ali-Imangholi-810197692/report/step-2/question-1/DCT#
```

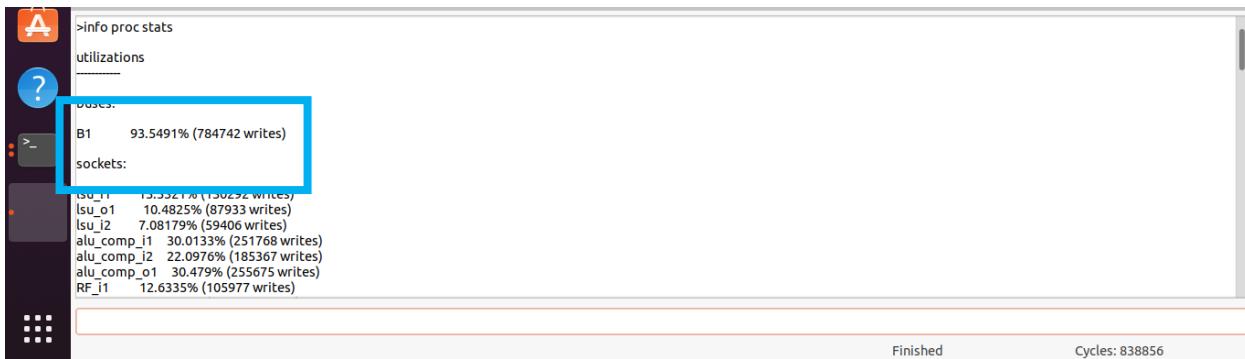
تعداد سیکلی که طول می کشد کد اجرا شود:

$$\text{clock cycle} = 838856$$



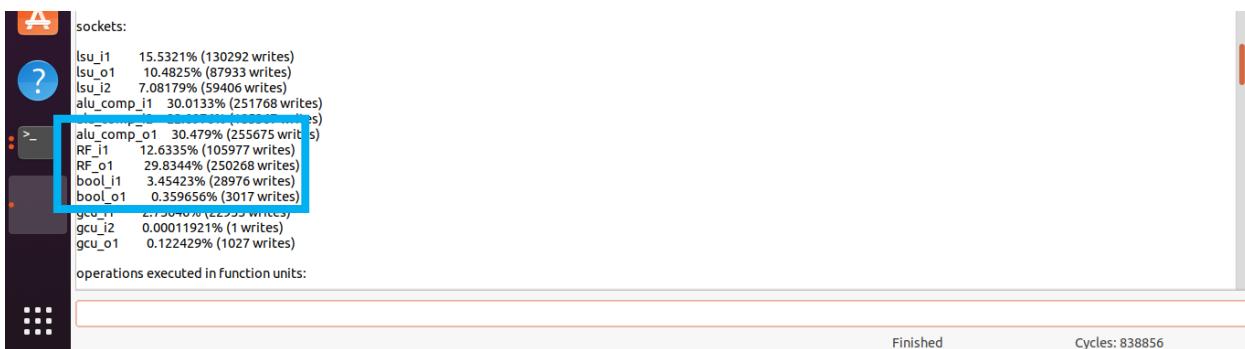
## اطلاعات bus

تنها یک bus وجود دارد و در 93% موقع مشغل به انجام پردازش می باشد.



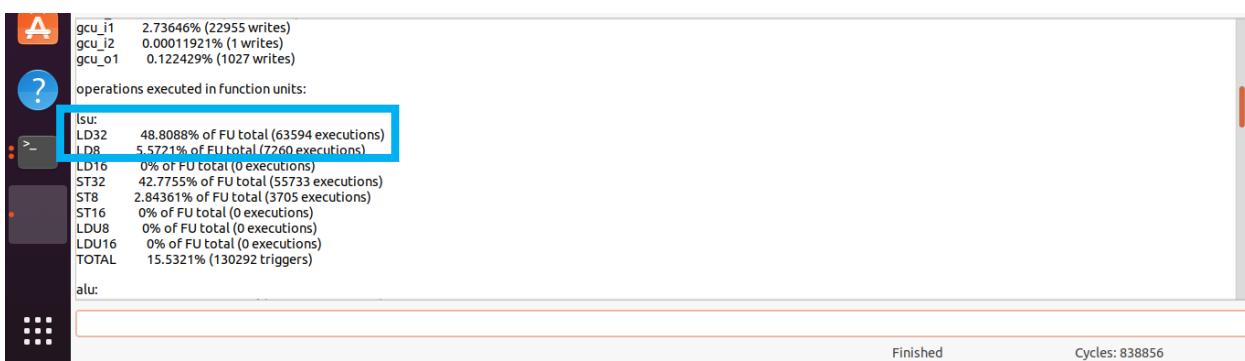
## اطلاعات مربوط با سوکت های موجود در سیستم مینimal:

بیشترین فعالیت مربوط به port های register file می باشد.  $RF_{i1}, RF_{o1} \rightarrow$  register file می باشد.



## دستورات اجرا شده:

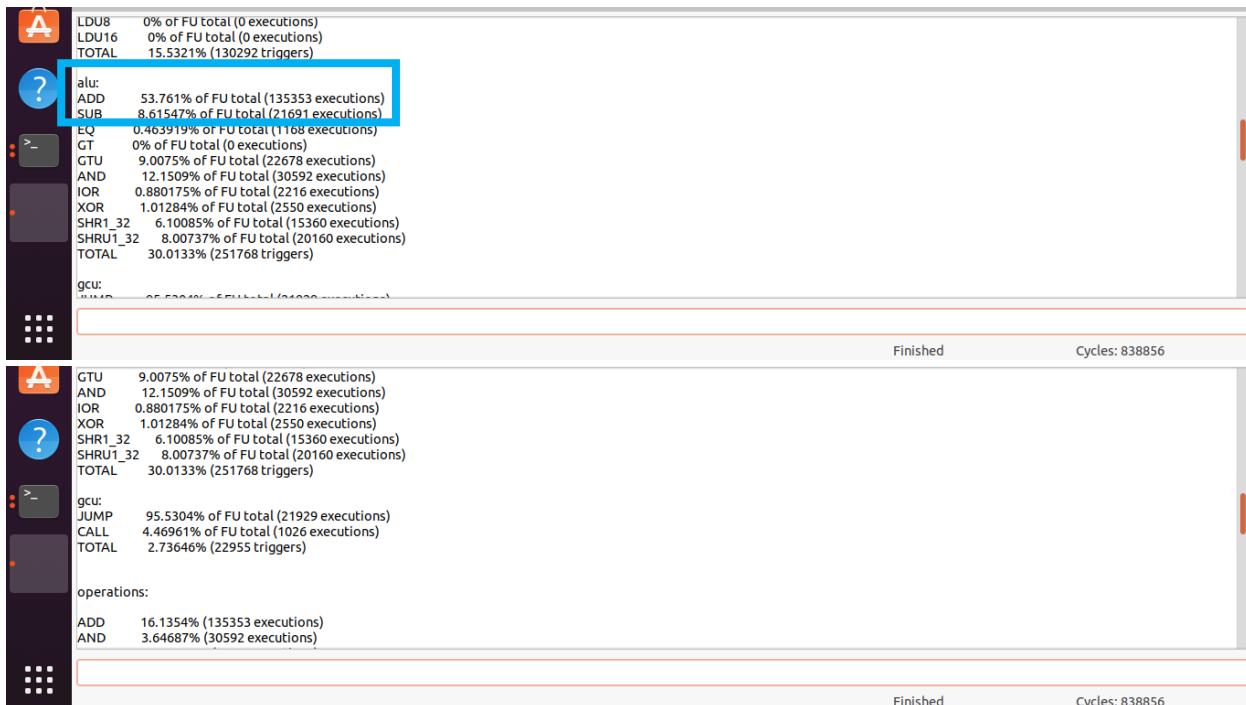
LD32 بیشترین مقدار را نسبت به باقی دستور ها داشته است.



## عملیات های انجام شده در ALU

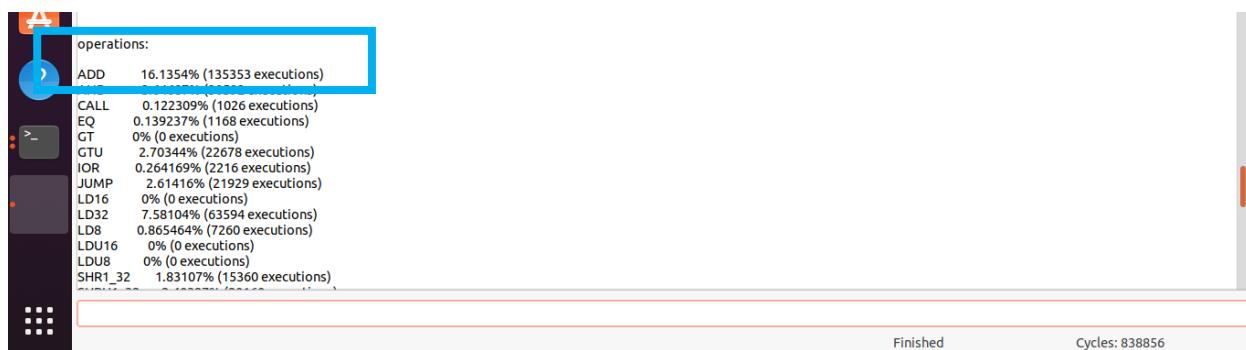
بیشترین عملیات مربوط به ADD می باشد.

این امر منطقی نیز می باشد، زیر کد اجرا شده ضرب و جمع های زیادی را انجام می دهد و به همین جهت قابل پیشگویی است که شمار تکرار ADD نسبت باقی عملیات ها بیشتر می باشد.



## عملیات های انجام شده (در حالت کلی):

در حالت کلی نیز عملیات ADD بیشترین استفاده را داشته است.



اطلاعات دسترسی به رجیستر ها:

register accesses:

RF:  
0 98772 reads, 0 guard reads, 2052 writes  
1 47936 reads, 0 guard reads, 17345 writes  
2 54375 reads, 0 guard reads, 38559 writes  
3 22470 reads, 0 guard reads, 22470 writes  
4 26715 reads, 0 guard reads, 25551 writes  
TOTAL 250268 reads, 0 guard reads, 105977 writes  
TOTAL 5 registers used

bool:  
TOTAL 0 registers used

immediate unit accesses:

>

Finished Cycles: 838856

## عملیات Boolean

immediate unit accesses:

A ? > - .

1 47936 reads, 0 guard reads, 17345 writes  
2 54375 reads, 0 guard reads, 38559 writes  
3 22470 reads, 0 guard reads, 22470 writes  
4 26715 reads, 0 guard reads, 25551 writes  
TOTAL 250268 reads, 0 guard reads, 105977 writes  
TOTAL 5 registers used

bool:  
0 1774 reads, 21102 guard reads, 25741 writes  
1 1243 reads, 5143 guard reads, 3235 writes  
TOTAL 3017 reads, 20245 guard reads, 28976 writes  
TOTAL 2 registers used

immediate unit accesses:

>

Finished Cycles: 838856

: Highlight top execution counts

Exec Count = 15296

File View Edit Command Source Program Data Help

Machine Program Run Resume Step Next

0: B1

Exec Count	Address range
15296	8596 - 8624
1024	8632 - 8640
1024	8626 - 8628
1024	8579 - 8595
64	8538 - 8549
64	8512 - 8523
64	8494 - 8503
64	8445 - 8454
64	8417 - 8432
64	8384 - 8406
64	8283 - 8292
64	8201 - 8251
64	8160 - 8184
64	8098 - 8147
64	7924 - 7968
64	7875 - 7911
64	7815 - 7862
64	7702 - 7712
64	7624 - 7677

>run

>info proc cycles  
838856

>info procsstats

utilizations

buses:

B1 93.5491% (784742 writes)

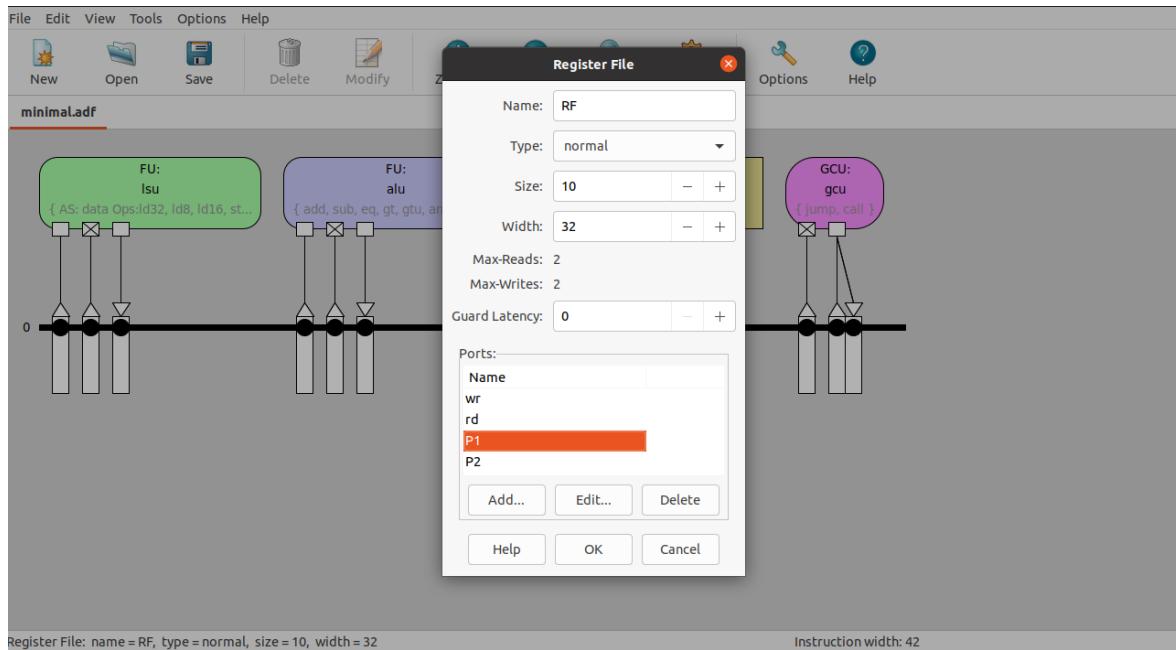
sockets:

lsu.i1 15.5321% (130292 writes)  
lsu.o1 10.4825% (87933 writes)  
lsu.j2 7.08179% (59406 writes)

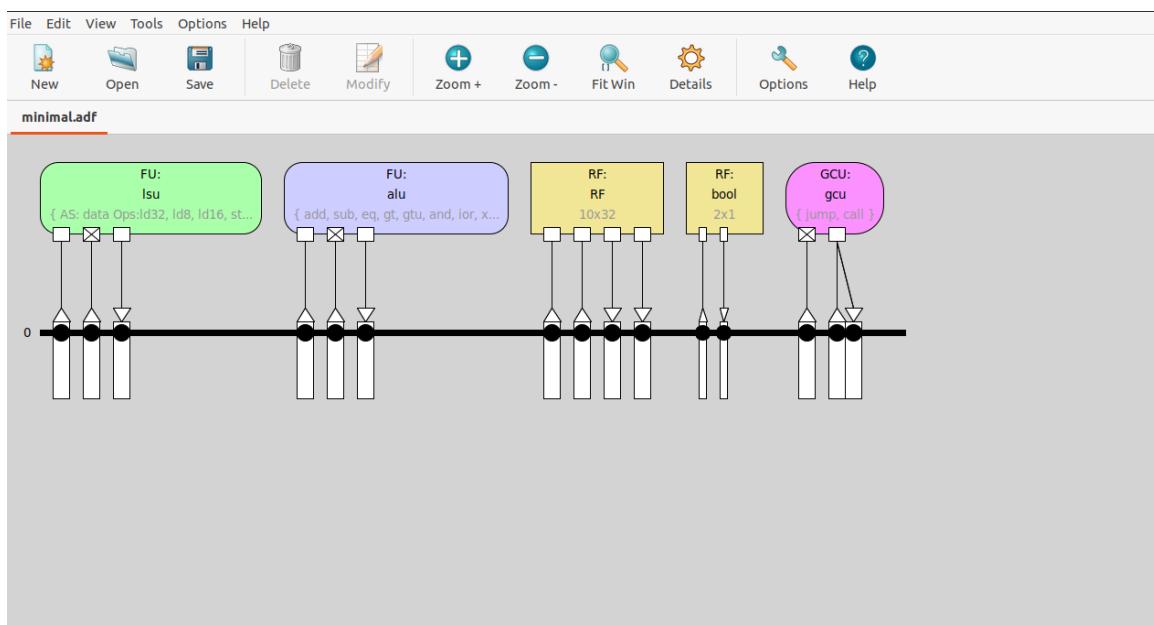
Finished Cycles: 838856

## Part 2 → Question2 & 3

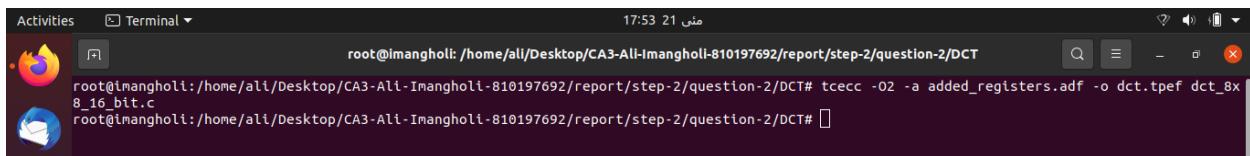
اضافه کردن سایز Register file از 5 به 10 و افزودن port به آن:



شماتیک کلی پس از اضافه کردن سایز Register file و افزودن port:



## کامپایل کردن کد bit\_16\_8x8\_dct.c (تغییرات در حالت جدید :Register file

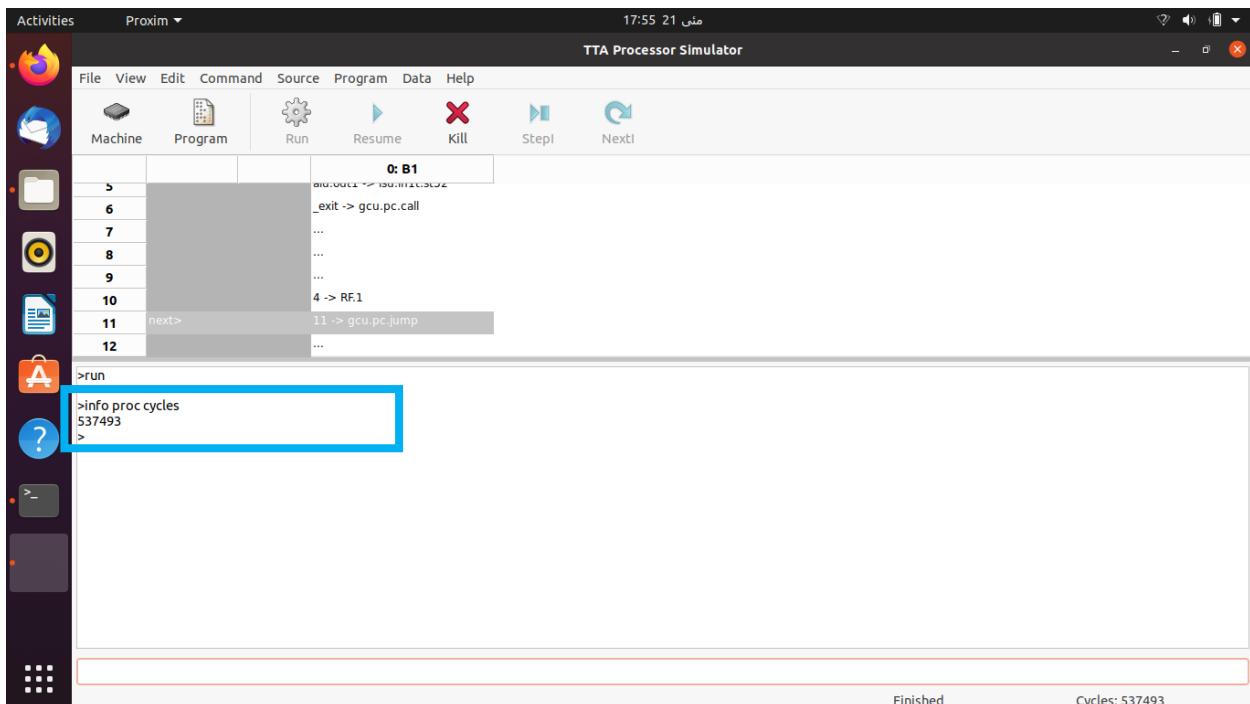


```
Activities Terminal 17:53 21
root@imangholi:/home/all/Desktop/CA3-Ali-Imangholi-810197692/report/step-2/question-2/DCT# tcc -O2 -a added_registers.adf -o dct.tpef dct_8x8_bit.c
root@imangholi:/home/all/Desktop/CA3-Ali-Imangholi-810197692/report/step-2/question-2/DCT#
```

تعداد سیکلی که طول می کشد کد اجرا شود:

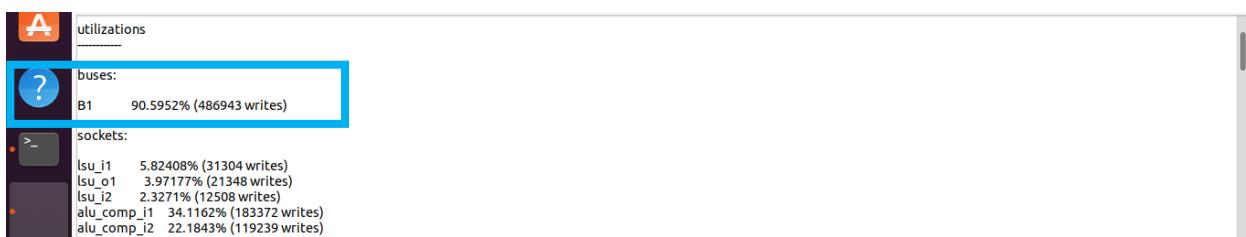
$$clock\ cycle = 537493$$

تغییرات اعمالی منجر شده است که clock cycle مورد نیاز برای اجرای برنامه کاهش بباید زیرا هم سایز رجیستر افزایش یافته است که این مورد باعث می شود تا مدام مجبور به حذف محتوای رجیستر و قرار دادن محتوای جدید نباشیم و از طرف دیگر هم برای نوشتن و هم برای خواندن از port 2 تا port 7 مجاز نباشد و اگر یک port مشغول باشد می توان از port دیگر برای دسترسی استفاده نمود.



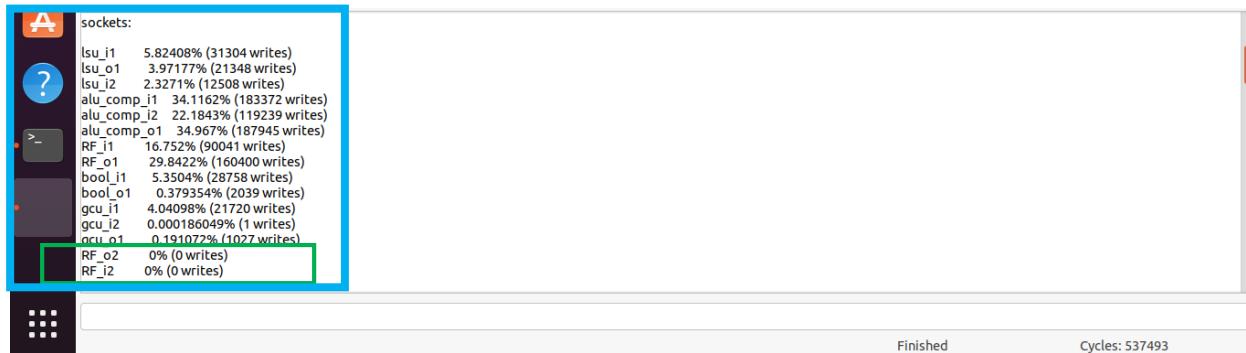
### اطلاعات bus

نهایا یک bus وجود دارد و در 90% موقع مشغول به انجام پردازش می باشد که نسبت به حالت قبل حدود 3 درصد بهبود یافته است.



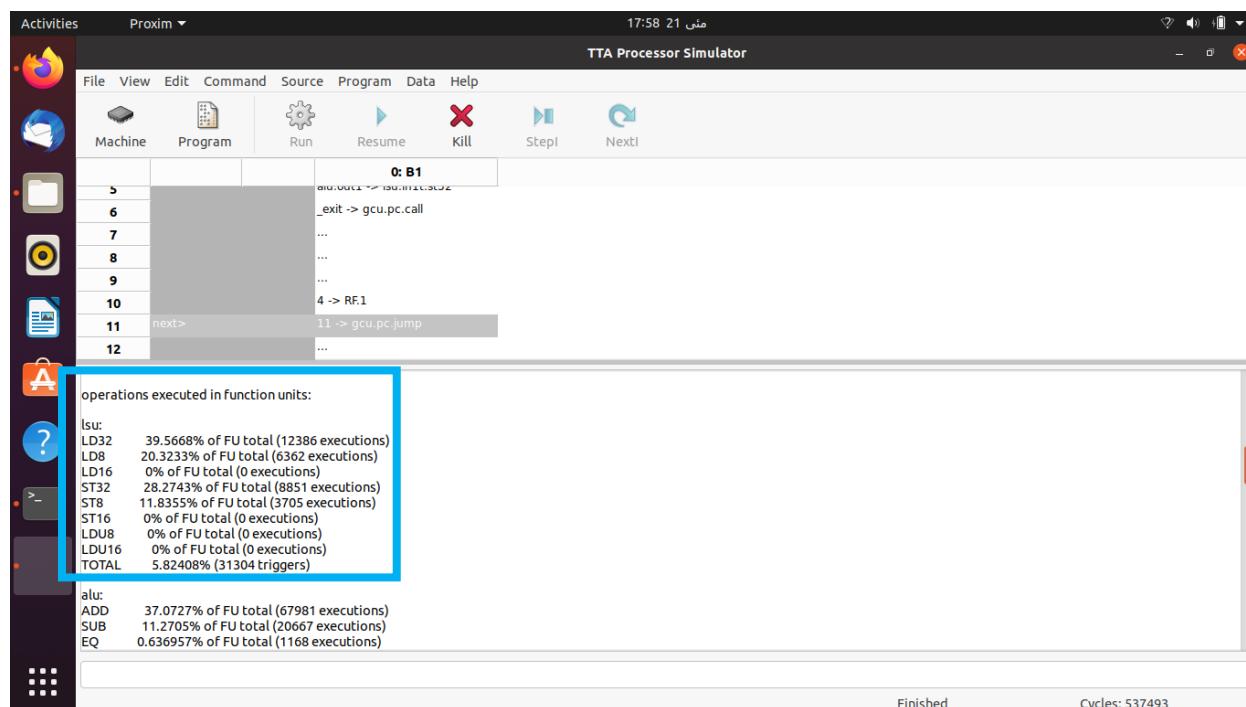
## اطلاعات مربوط با سوکت های موجو در سیستم جدید (تغییرات در Register file)

با توجه به خروجی گزارش port های اضافه ای که تعریف کردیم اصلا نیاز نشده اند و در تمامی موارد وجود یک port برای نوشتن و یک port برای خواندن کافی است و به همین جهت از port های اضافه ای که تعریف شده اند استفاده نشده است.



### دستورات اجرا شده:

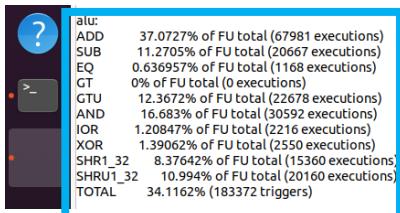
همانند حالت قبل *ld32* بیشترین مقدار را نسبت به باقی دستور ها داشته است.



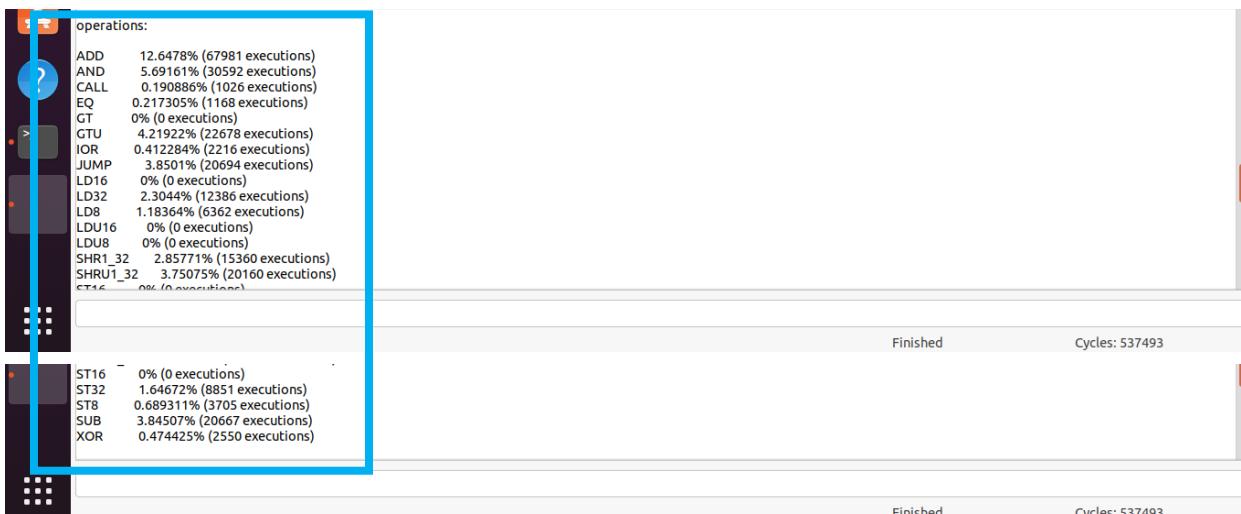
## عملیات های انجام شده در ALU

بیشترین عملیات مربوط به ADD می باشد.

همانطور که در قسمت قبل نیز توضیح داده شد این امر منطقی نیز می باشد، زیر کد اجرا شده، ضرب و جمع های زیادی را انجام می دهد و به همین جهت قابل پیش‌بینی نیز می باشد که شمار تکرار ADD نسبت باقی عملیات ها بیشتر می باشد.

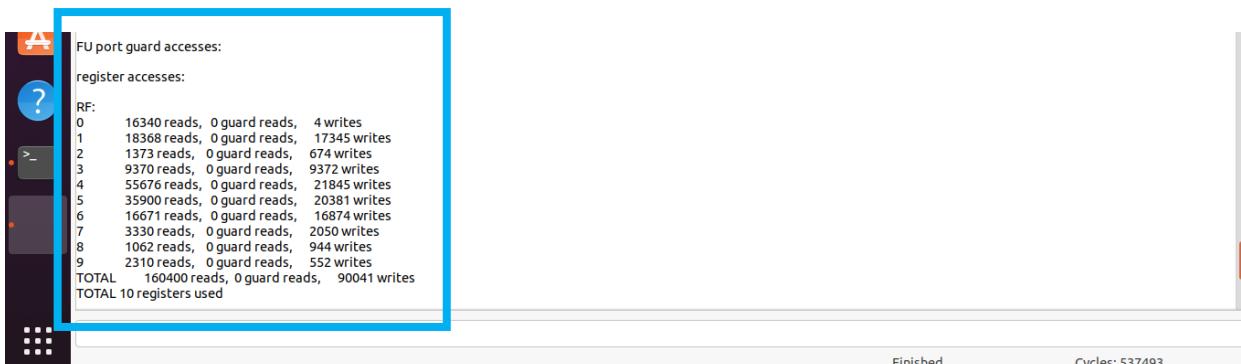


عملیات های انجام شده (در حالت کلی):

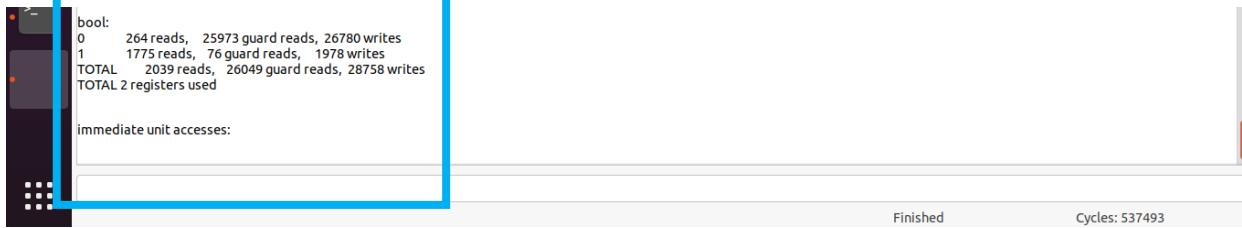


## اطلاعات دسترسی به رجیستر ها:

به علت افزایش سایز Register file دسترسی به رجیستر ها با حالت متعادل تری انجام شده است و نسبت به حالت قبل تعداد read و write کمتری به هر رجیستر مستقل انجام شده است زیرا در این حالت نیاز نیست که پی در پی مقدار رجیستر ها را برای داده های جدید خالی کنیم و مقدار داده های جدید را درون آن قرار داهیم، در این حالت فضای برای داده های جدید علاوه بر داده های قدیمی در اختیار داریم.



## عملیات Boolean



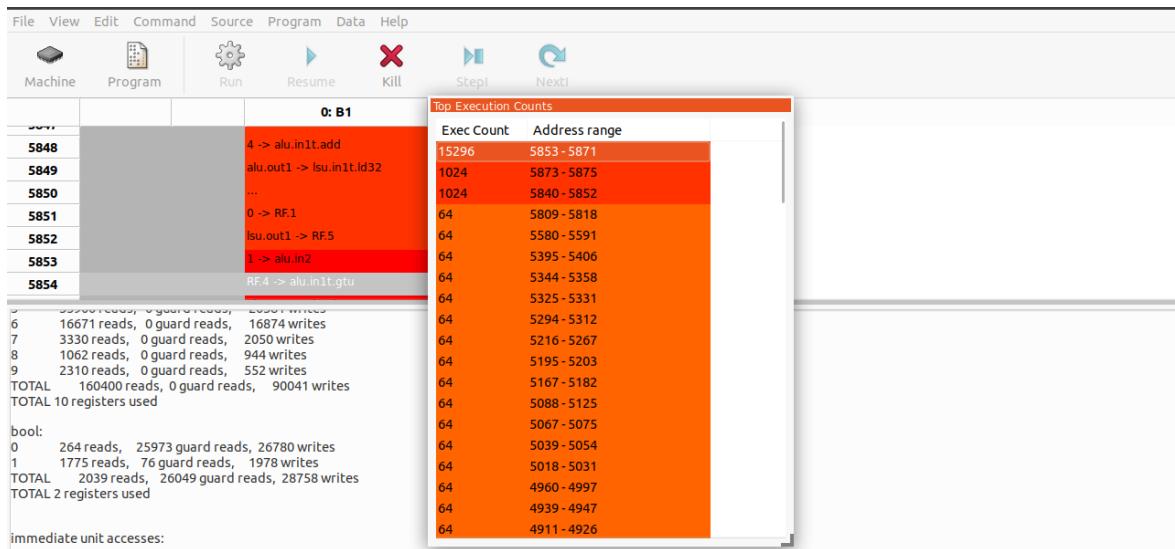
```
boot:  
0 264 reads, 25973 guard reads, 26780 writes  
1 1775 reads, 76 guard reads, 1978 writes  
TOTAL 2039 reads, 26049 guard reads, 28758 writes  
TOTAL 2 registers used  
  
immediate unit accesses:
```

Finished Cycles: 537493

### : Highlight top execution counts

Exec Count = 15296

این پارامتر نسبت به حالت قبل تغییری نکرده است و برابر با حالت قبلی می باشد زیرا در روند الگوریتم کد تغییری را انجام نداده ایم و یا اختصاصی ای را به پردازندهی base functional unit اضافه نکرده ایم.



File View Edit Command Source Program Data Help

Machine Program Run Resume Kill Step Next

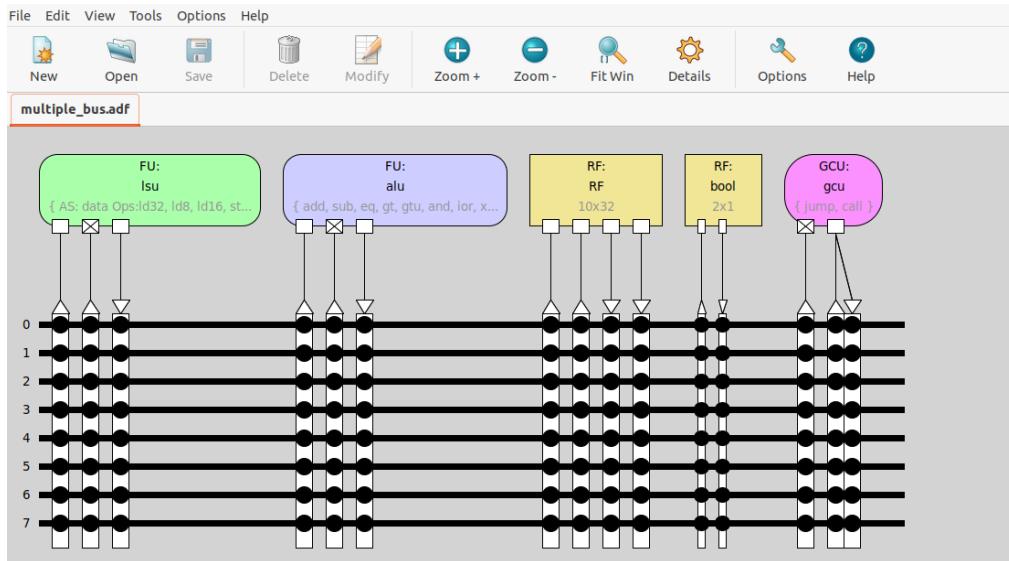
0: B1

Exec Count	Address range
15296	5853 - 5871
1024	5873 - 5875
1024	5840 - 5852
64	5809 - 5818
64	5580 - 5591
64	5395 - 5406
64	5344 - 5358
64	5325 - 5331
64	5294 - 5312
64	5216 - 5267
64	5195 - 5203
64	5167 - 5182
64	5088 - 5125
64	5067 - 5075
64	5039 - 5054
64	5018 - 5031
64	4960 - 4997
64	4939 - 4947
64	4911 - 4926

```
0 3360 reads, 0 guard reads, 25973 writes  
1 16671 reads, 0 guard reads, 16874 writes  
2 3330 reads, 0 guard reads, 2050 writes  
3 1062 reads, 0 guard reads, 944 writes  
4 2310 reads, 0 guard reads, 552 writes  
TOTAL 160400 reads, 0 guard reads, 90041 writes  
TOTAL 10 registers used  
  
bool:  
0 264 reads, 25973 guard reads, 26780 writes  
1 1775 reads, 76 guard reads, 1978 writes  
TOTAL 2039 reads, 26049 guard reads, 28758 writes  
TOTAL 2 registers used  
  
immediate unit accesses:
```

## Part 2 → Question3 & 4

تغییر ساختار bus به 8 عدد bus Fully connected



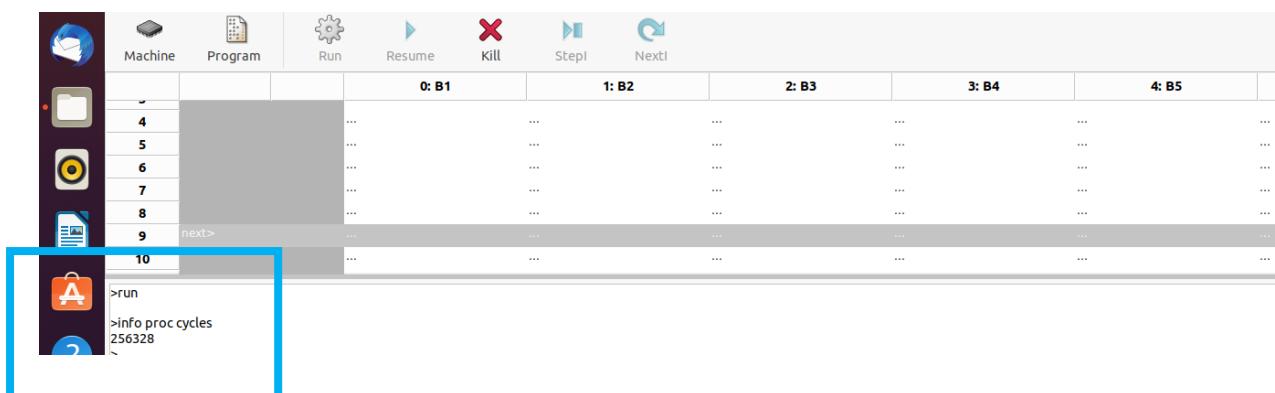
کامپایل کردن کد bit\_16\_8x8\_dct.c در حالت جدید (تغییرات در ساختار bus):

```
tcecc -02 -a multiple_bus.adf -o dct.tpef dct_8x8_1
```

تعداد سیکلی که طول می کشد کد اجرا شود:

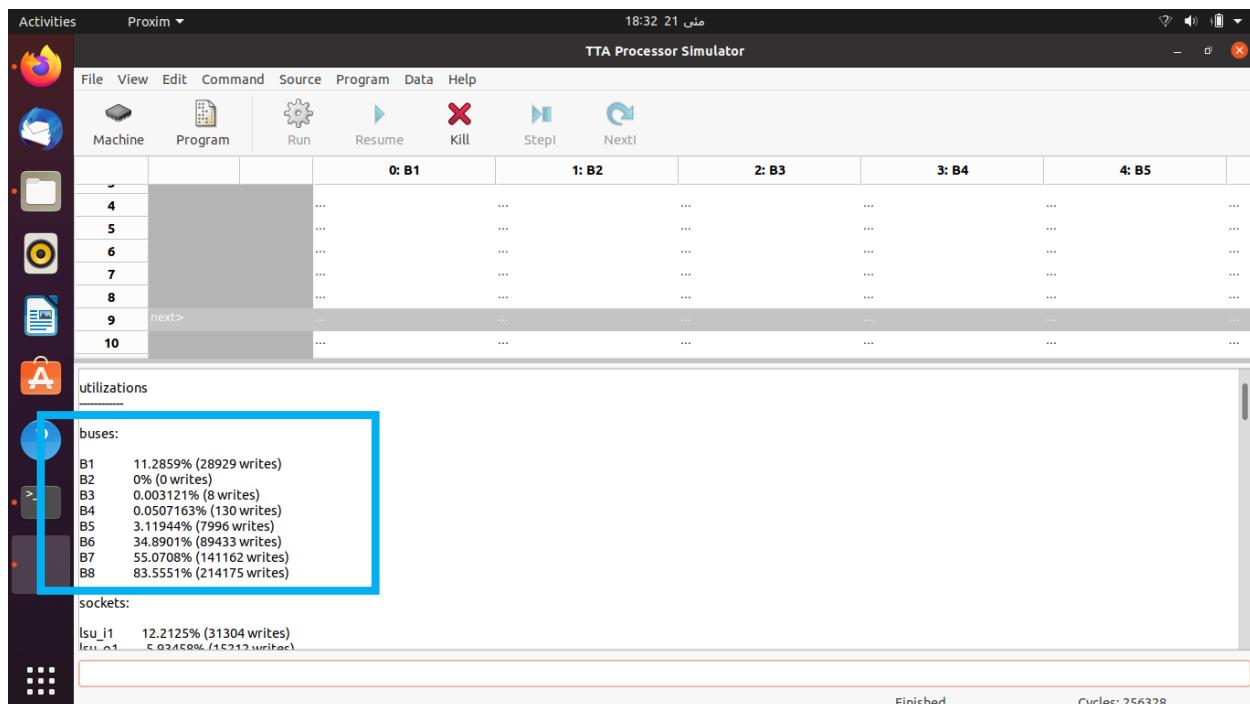
$$\text{clock cycle} = 256328$$

حدودا تعداد سیکل مورد نیاز برای اجرای برنامه نسبت به حالت قبل نصف شده است که این به علت این است که دیگر محدودیت بر روی نداریم و تمامی bus ها دسترسی bus fully connected به تمامی component دارند.



## اطلاعات bus

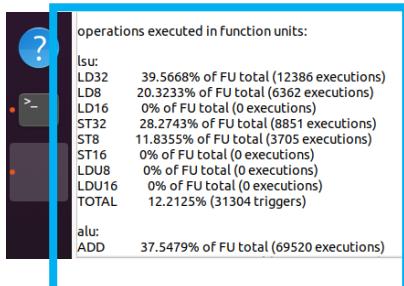
همانطور که از تصویر زیر مشخص است تعداد bus ها 8 عدد می باشد.  
بیشترین load بر روی bus شماره‌ی 8 قرار دارد. (B8)  
bus های شماره‌ی 2 و 3 و 4 عمل فعالیتی را انجام نمی دهند و تنها منجر به افزایش تعداد component و area مصرفی می باشند و برای سیستم performance ایجاد نمی کنند پس می توان از آن ها صرف نظر کرد.  
bus شماره‌ی 5 نیز مشارکت کمتر از 5 درصدی در اجرای برنامه دارد و می توان از این bus نیز صرف نظر کرد.



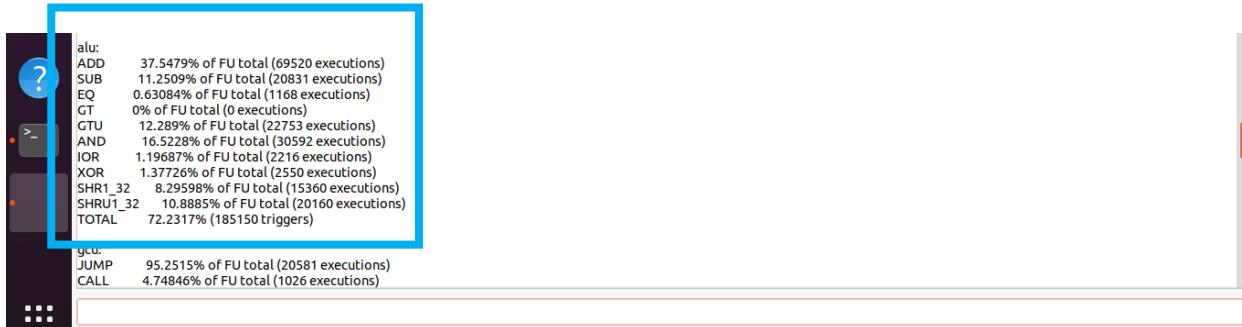
## اطلاعات مربوط با سوکت های موجود در سیستم جدید (تغییرات در ساختار bus):



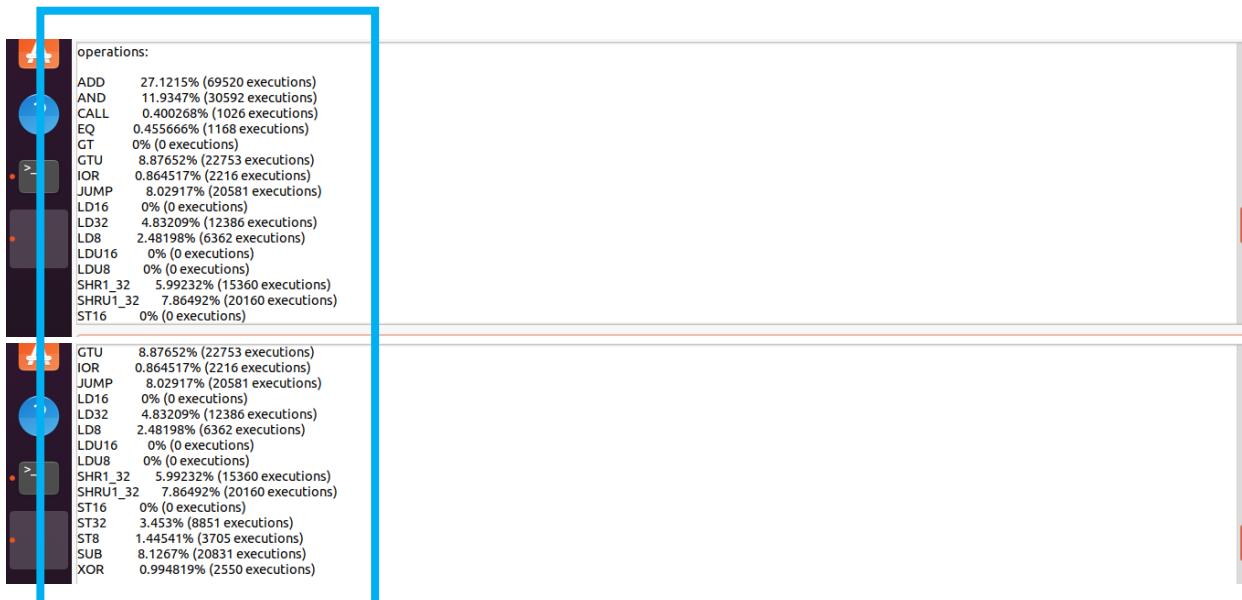
دستورات اجرا شده:



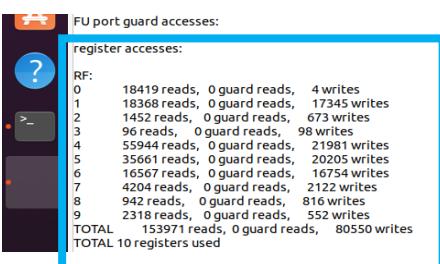
عملیات های انجام شده در ALU



عملیات های انجام شده (در حالت کلی):



اطلاعات دسترسی به رجیستر ها:



## Boolean Operations

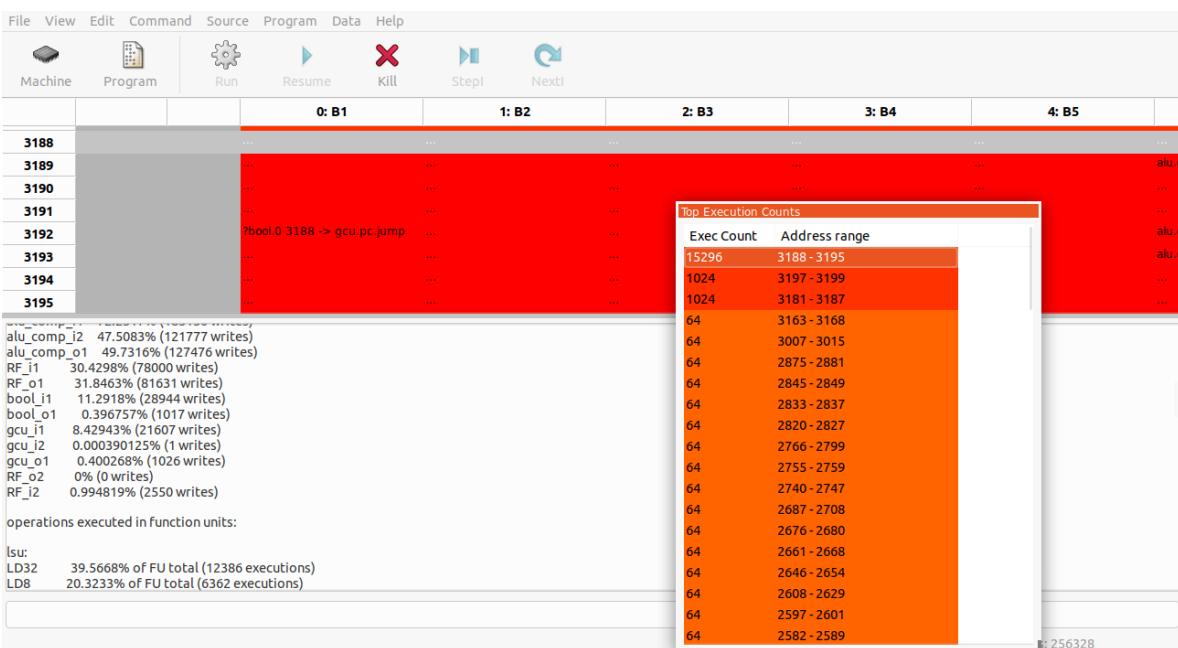
boot:  
0 1210 reads, 28777 guard reads, 27734 writes  
1 1138 reads, 152 guard reads, 1210 writes  
TOTAL 2348 reads, 28929 guard reads, 28944 writes  
TOTAL 2 registers used

immediate unit accesses:

Finished Cycles: 256328

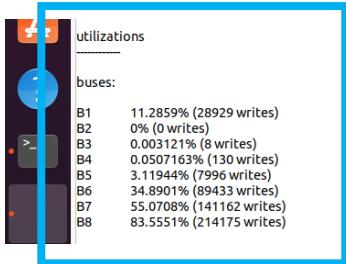
: Highlight top execution counts  
Exec Count = 15296

این پارامتر نسبت به حالت قبل تغییری نکرده است و برابر با حالت قبلی می باشد زیرا در روند الگوریتم کد تغییری را انجام نداده ایم و یا اختصاصی ای را به پردازنده‌ی base functional unit اضافه نکرده ایم.

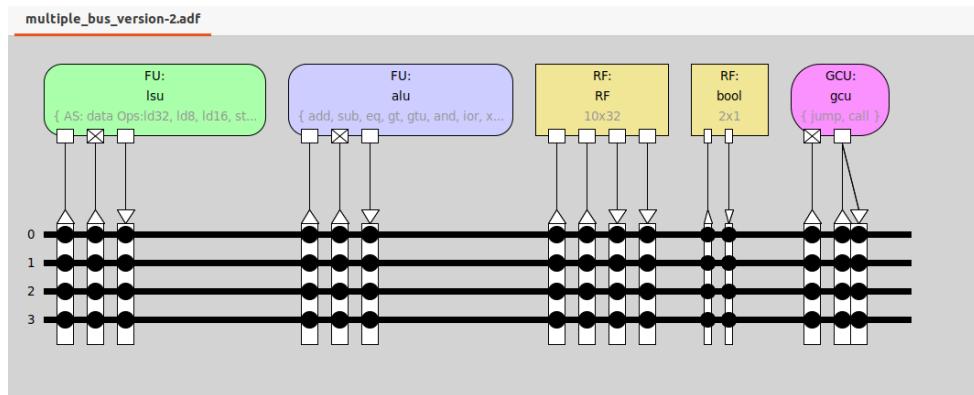


## Part 2 → Question5

bus های شماره‌ی 2 و 3 و 4 علاوه‌العملیتی را انجام نمی‌دهند و تنها منجر به افزایش تعداد component و area مصرفی می‌باشند و برای سیستم performance ایجاد نمی‌کنند پس می‌توان از آن‌ها صرف نظر کرد. bus شماره‌ی 5 نیز مشارکت کمتر از 5 درصدی در اجرای برنامه دارد و می‌توان از این bus نیز صرف نظر کرد.



بنابراین bus های شماره‌ی 2 و 3 و 4 و 5 را حذف می‌نماییم:



کامپایل کردن برنامه برای سیستم دارای 4 عدد bus fully connected متصل شده‌اند:

```
root@imangholi:/home/ali/Desktop/CA3-Ali-Imangholi-810197692/report/step-2/question-5/DCT# tcecc -O2 -a multiple_bus_version-2.adf -o dct.tpef  
root@imangholi:/home/ali/Desktop/CA3-Ali-Imangholi-810197692/report/step-2/question-5/DCT#
```

تعداد سیکلی که طول می کشد کد اجرا شود:

همانطور که دیده میشود تعداد سیکلی که برای اجرای برنامه در حالت 4 عدد bus وجود دارد تفاوت اندکی با حالت 8 عدد bus دارد.

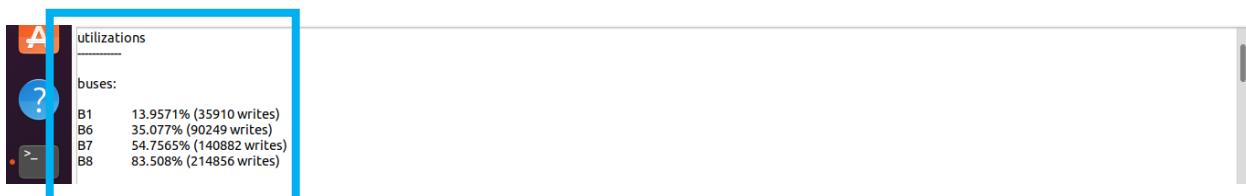
$$\text{clock cylce [8 bus]} = 256328$$

$$\text{clock cycle [4 bus]} = 257288$$

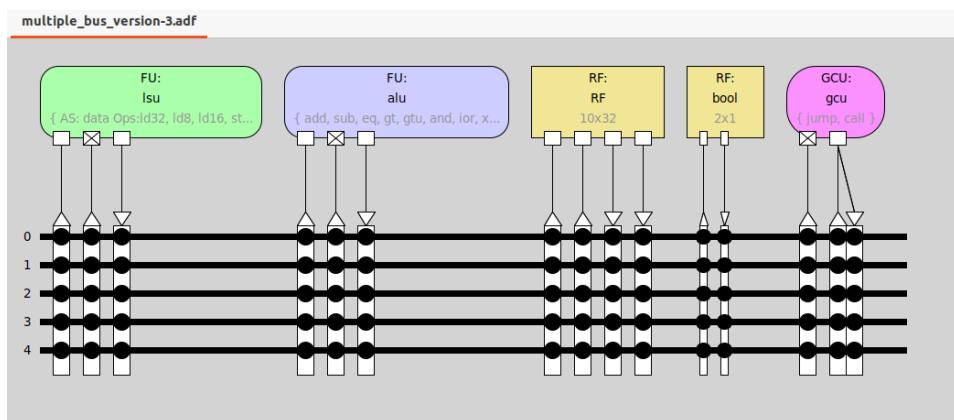
که این بین معنی است که حذف این 4 bus تاثیر زیادی بر clock cycle نداشته است ولیکن از لحاظ تعداد component مصرفی و area بهینه تر شده است.



در تصویر زیر نیز مشخص است که overhead bus بروی هر bus تغییر محسوسی نسبت به حالت قبل نداشته است و حذف این 4 عدد bus اضافه تاثیری بر overhead bus ها نداشته باشد.



برای اینکه مطمئن شویم که 4 عدد bus کافی است یا خیر و آیا با 1 عدد bus بالاتری نمی رسمیم، تعداد bus را به 5 مورد افزایش می دهیم:



مشاهده می شود که تغییر محسوس و مطلوبی را نداشته و همچنین overhead bus ها کاهش جشمگیری نداشته تا overhead همه‌ی آن‌ها متعادل شود و هنوز bus اضافه شده در موقع پردازش انجام می دهد.

$$clock cylce [8 bus] = 256328$$

$$clock cycle [5 bus] = 256344$$

$$clock cycle [4 bus] = 257288$$

```

>run
>info proc cycles
256344
>info proc stats
utilizations
buses:
B1 11.3235% (29027 writes)
B6 34.8879% (89433 writes)
B7 55.0737% (141178 writes)
B8 83.5561% (214191 writes)
B2 3.11925% (7996 writes)
sockets:
Finished Cycles: 256344
  
```

بنابراین با توجه به trade-off موجود بین area و performance بهتر است که از 4 عدد bus استفاده شود. ♦

## Part 2 → Question6

کامپایل کردن برنامه در حالت `--disable-inlining`

```

Activities Terminal ▾ 19:22
root@imangholi: /home/all/Desktop/CA3-Ali-Imangholi-810197692/report/step-2/question-6/DCT
root@imangholi: /home/all/Desktop/CA3-Ali-Imangholi-810197692/report/step-2/question-6/DCT# tcecc -O2 -a multiple_bus_version-2.adf -o dct.tpef
root@imangholi: /home/all/Desktop/CA3-Ali-Imangholi-810197692/report/step-2/question-6/DCT# 
  
```

تعداد سیکلی که طول می کشد کد اجرا شود:

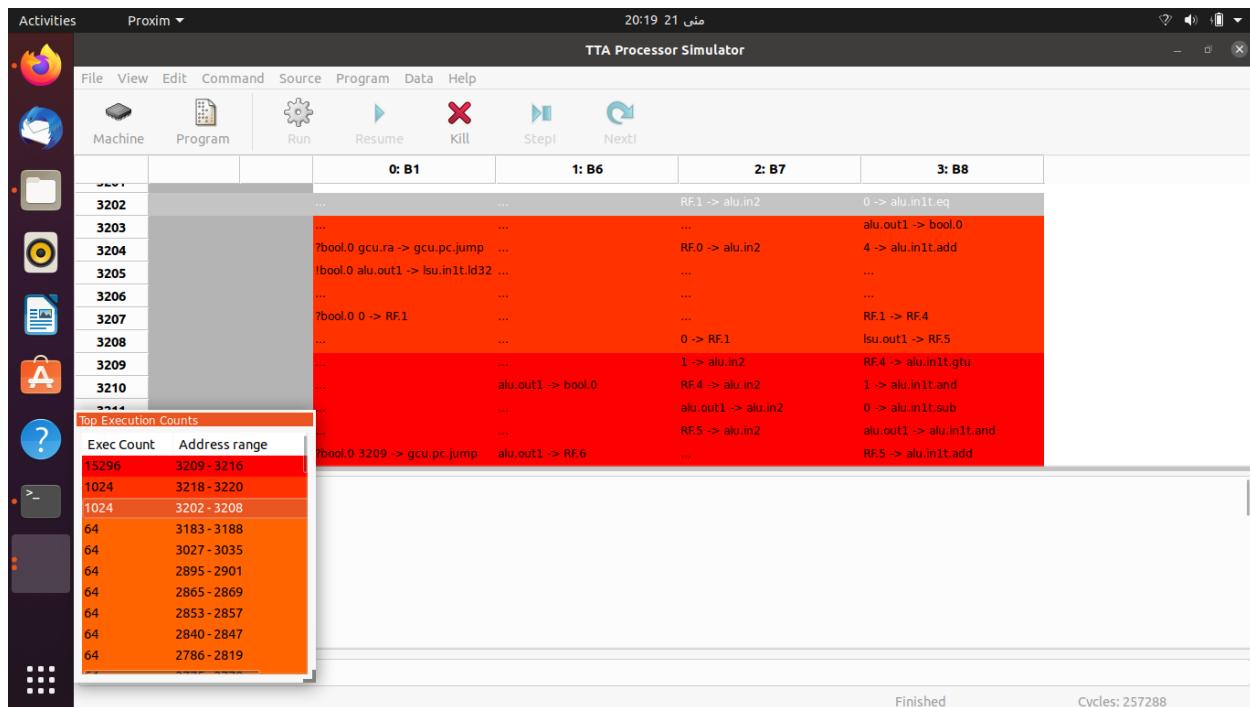
```

Activities Terminal ▾
>run
>info proc cycles
257288
>
  
```

تعداد سیکلی که طول می کشد تا برنامه اجرا شود در حالتی که inlining غیر فعال است باید بیشتر از حالتی که فعال است شود زیرا کامپایلر در این حالت حق ندارد optimization انجام دهد و باید برای دسترسی به هر قابع function call را انجام دهد ولیکن با توجه به

تصویر بالا clock cycle تغییری نسبت به حالت قبلی نکرده است ولی مطابق توضیحات باید بیشتر می شد.

( از TA محترم پرسیدم، فرمودند اشکالی نداره 😊 )



با کلیک کردن بر روی top execution counts مشخص نمی شود که کدام قسمت از کد مورد نظر و بیشترین اجرا را دارا می باشد(bug نرم افزار می باشد). به همین علت باتوجه به اطلاعات دستور کار برای قسمت های بعد پیش می رویم و ADD functional unit های و COS و MUL را ایجاد می نماییم.

---

## Part 3 → Question7

---

cos16:

```
unsigned int cos16(int input) {  
    unsigned int cosine_values[] = {32138,  
                                    27246,  
                                    18205,  
                                    6393};  
  
    unsigned int remainder = 0;  
    int address = 0;  
    int inverse = 0;  
    unsigned int value = 0;  
    remainder = input - input/8;  
  
    if (remainder == 1) {  
        address = 0;  
    } else if (remainder == 3) {  
        address = 1;  
    } else if (remainder == 5) {  
        address = 2;  
    } else if (remainder == 7) {  
        address = 3;  
    } else {  
        address = 0;  
    }  
  
    if (input >8)  
        inverse = 1;  
  
    if (inverse == 1) {  
        value = 65536-cosine_values[address];  
    } else {  
        value = cosine_values[address];  
    }  
  
/*#ifdef DEBUG_COS  
printf("COSINE:\n");  
printf("input : %3d\n",input);  
printf("address : %1d, neg : %1d\n",address,inverse);  
printf("cos : %d\n",value);  
#endif*/  
    return value;  
}
```

در تابع cos16 عملیات کسینوس را انجام می دهد.

بدین منظور آرگومان مورد نظر که به صورت int می باشد به تابع پاس داده می شود.

سپس روند کلی تابع بدین صورت است که همانند یک multiplexer عمل می کند و در بدنه ای تابع مقدار select این multiplexer تعیین می شود.

بدین منظور آرایه ای از مقادیر خروجی تابع تعریف می شود.

سپس متغیر remainder تعریف می شود که مقدار address بر اساس آن تعیین می شود.

سپس متغیر address تعریف می شود که همانند select برای multiplexer می باشد.

و در ادامه متغیر inverse تعریف می شود که برای مقادیر منفی مورد استفاده قرار می گیرد.

متغیر value از نوع unsigned integer تعیین می شود و مقدار نهایی تابع در آن قرار می گیرد و به بیرون از تابع پاس داده می شود.

برای انجام محاسبات در ابتدا مقدار remainder را مشخص می‌نماییم.

مقدار remainder از آن جهت حائز اهمیت است که مقدار  $(n+1)2$  را در رابطه  $\cos\left(\frac{(2n+1)\pi}{8}\right)$  محاسبه می‌نماید.

در این تابع این مقادیر تحت عنوان 1 و 3 و 5 و 7 محاسبه شده اند و باقی موارد don't care محسوب شده اند که این امر منجر شده است تا کد به درستی، یاسخ را برای موارد بالاتر محاسبه ننماید.

برای محاسبه  $remainder = input - input/8$  از رابطه  $remainder = input - input/8$  استفاده می‌نماییم.

براساس مقدار بدست آمده برای remainder مقدار address را تعیین می نماییم و همانطور که از کد مشخص است این کار به از طریق دستورات شرط، if – else if – else if – remainder امکان بذیر است.

اگر مقدار آگومان ورودیتابع بیشتر از 8 باشد مقدار flag مربوط به inverse کنیم که بدین معنی است که مقدار حاصل منفی می باشد و در محاسبات نهایی، نیز مقدار value از رابطه محذا محسوسه م، شود.

در انتهای نیز با توجه به اینکه مقدار inverse یک مم، باشد با خبر مقدار value، ا تعیین مم، نمایم.

اگر مقدار نهایی، منفی، یاشد پا به عبارتی، مقدار inverse باشد مقدار value را طبق رابطه‌ی

نهایی، مشت است و حاصل، نهایی، دا طق، اطهی، [value = cosine\_values[address]] محاسه م، نهایی.

د، ادامه کد آبديت شده ۱، آد، به و تفاوت آن با کد بالا و سطوف شدن اشکال، کد بالا (تا حدوده ۱)، ۱، سان، سه، گنجمه.

```

unsigned int cos16(int input) {
    unsigned int cosine_values[] = {32767,
                                    32138,
                                    30274,
                                    27246,
                                    23170,
                                    18205,
                                    12540,
                                    6393};

    unsigned int remainder = 0;
    int inverse = 0;
    int temp = 0;
    unsigned int value = 0;
    temp = input/8;
    remainder = input - input/16;
    if (remainder >8) {
        remainder = 16 - remainder;
    }

    while (temp>=4)
        temp -= 4;

    if (temp ==1 || temp ==2)
        inverse = 1;

    if (inverse == 1) {
        if (address == 0) {
            value = 65536-cosine_values[remainder]-1;
        }
        else {
            value = 65536-cosine_values[remainder];
        }
    } else {
        value = cosine_values[remainder];
    }

/*#ifdef DEBUG_COS
printf("COSINE: \n");
printf("input : %d\n",input);
printf("address : %d, neg : %d\n",address,inverse);
printf("cos : %d\n",value);
#endif*/
    return value;
}

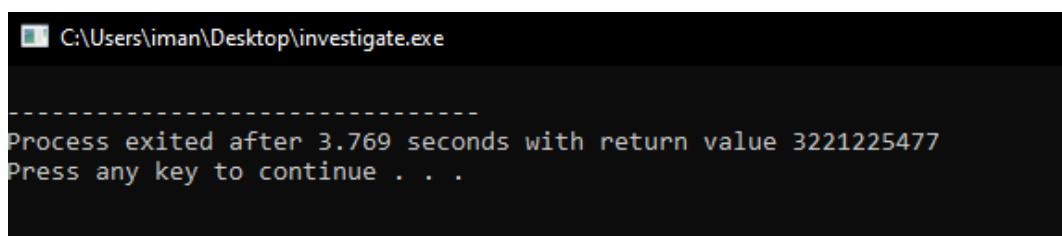
```

در کد update شده‌ی بالا مقادیر cos() از میان 8 مقدار گیزینش می‌شوند و در این کد سعی شده است تا برای مقادیر بالاتر نیز بتوان مقدار cos() را به درستی محاسبه نمود.

تغییراتی که کد update شده نسبت به کد قبلی داشته علاوه بر اینکه مقادیر cosine\_value به 8 مقدار افزایش یافته است، شیوه‌ی محاسبه‌ی remainder نیز تغییر یافته است و برای محاسبه‌ی remainder از رابطه‌ی دیگری بهره می‌بریم. از طرفی مقدار remainder را به وسیله‌ی یک شرط if کمتر از 8 نگه می‌داریم تا بتوان به وسیله‌ی remainder از 8 مقدار cosine\_value انتخاب نماییم. از طرف دیگر برای بررسی مشبт و منفی بودن مقدار نهایی، متغیر temp را تعریف می‌نماییم که مقدار آن را میان 4 ناحیه تقسیم می‌نماییم و به ازای ناحیه‌ی 1 و 2 مقدار cos() را منفی در نظر می‌گیریم.

در نهایت با توجه به اینکه مقدار cos() مشبт است یا منفی مقدار نهایی را محاسبه می‌نماییم. تغییرات این کد باعث شده است که علاوه بر مقادیر 1 و 3 و 5 و 7 و ... 15؛ مقادیر دیگری را نیز بتوان توسط این تابع cos() محاسبه نمود ولیکن ابراهیمی نیز دارد که به مواردی از آن در ذیل اشاره می‌کنیم:

با وجود بمبود‌های حاصل ولیکن هنوز هم این کد دارای ابراد می‌باشد، به طور مثال به ازای فراخوانی تابع به ازای ورودی 18، پس از انجام محاسبات مقدار remainder برابر با -1 می‌شود و از آنجاکه مقدار remainder را به عنوان select برای multiplexer انتخاب شده است و به وسیله‌ی remainder به داده‌های cosine\_value دسترسی داریم با قرار دادن remainder برابر با -1 نمی‌توان از آرایه فرخوانی کرد. زیرا خانه‌های آرایه از 0 شروع می‌شوند نه -1 با اجرا کردن این کد به ازای ورودی 18 در Dev-C++ نیز ارور زیر مشاهده می‌شود که ناشی از این دسترسی نادرست به آرایه است.  
(علاوه بر این مشکل کامپایل هم وجود دارد زیر متغیر address تعریف نشده است.)



```
C:\Users\iman\Desktop\investigate.exe

-----
Process exited after 3.769 seconds with return value 3221225477
Press any key to continue . . .
```

## fix\_mul\_16:

```
unsigned int fix_mul_16(unsigned int num1, unsigned int num2){
    unsigned int temp, prod;
    int temp1,temp2;
    int neg1= 0;
    int neg2 = 0;

    if (num1 > 32767){
        if (num1 > 65535) {
            num1 = 65535;
        }
        neg1 = 1;
        temp1 = 0-num1+65536;
    } else {
        temp1 = (int)(num1);
    }
    if (num2 > 32767){
        if (num2 > 65535) {
            num2 = 65535;
        }
        neg2 = 1;
        temp2 = 0-num2+65536;
    } else {
        temp2 = (int)(num2);
    }
    prod = ((int)(temp1) * (int)(temp2)) >> 15;

    if ((neg1 == 0 && neg2 == 1) ||
        (neg1 == 1 && neg2 == 0)) {
        prod = 65536 - prod;
    }

    return prod;
}
```

عملیات ضرب دو عدد 16 بیتی را انجام می دهد. به این صورت که هم اپرند اول و هم اپرند دوم به صورت `unsigned integer` در

نظر گرفته می شوند ولیکن محاسبات به نحوی انجام می شود که جنبه ی باینری داشته باشد.

در ابتدا دو عدد `unsigned int` به تابع پاس داده می شود.

در بدنه ی تابع تعریف 6 متغیر وجود دارد.

متغیر `prod` از نوع `unsigned int` و در ادامه ی کد تابع خواهیم دید که نتیجه ی نهایی در `prod` ذخیره می شود.

متغیر های `temp1` و `temp2` که برای هر یک از اپرند ها تعییه شده است تا عملیات را از جنبه علامت بتوان بر روی اپرند ها انجام داد.

متغیر های `neg1` و `neg2` که منفی بودن هر یک از اپرند ها را نشان می دهند.

در ادامه بررسی روند را برای یک اپرند بیان می کنیم و به طور مشابه برای اپرند دوم نیز این عملیات انجام می شود.

برای بررسی اپرند اول که آیا منفی است یا خیر از یک `if` استفاده شده است به این صورت که اگر مقدار اپرند اول از  $2^{15}$  بزرگ تر باشد به این معنی است که بیت 16 نیز پر می باشد و تابع بیت 16 را بیت علامت در نظر گرفته است بنابراین اگر مقدار عدد از 32767 بزرگتر باشد به این معنی است که این عدد منفی است.

در ادامه ی `if` دیگر نیز وجود دارد که بررسی می کند که عدد پاس داده شده با تابع از مقدار  $2^{16}$  بزرگتر نشود، در غیر این صورت اگر مقدار پاس داده شده از 65536 بزرگتر باشد مقدار اپرند اول را به صورت پیشفرض برابر با 65536 در نظر می گیرد.

در ادامه اگر `if` اول برقرار باشد ( به این معنی است که اپرند اول منفی است) تابع مقدار `neg1` را برابر با 1 قرار می دهد

و این مقدار منفی را به صورت  $temp1 = 0 - num1 + 65536$  محاسبه می کند که این `temp1` اندازه ی عدد منفی را در خود نگه می دارد و متغیر `neg1` علامت این عدد منفی را در خود نگه می دارد. اگر شرط `if` اولی برقرار نباشد، بدین معنی است که اپرند اول مثبت می باشد و بدون هیچ تغییری مقدار آن در `temp1` قرار می گیرد.

در ادامه به همین شکل محاسبات را برای اپرند دوم نیز انجام می دهد.

برای محاسبه‌ی ضرب دو عدد با استفاده از اپراتور \* عملیات ضرب را انجام می‌دهیم و از آن جایی که دو عدد 16 بیت در هم ضرب می‌شوند حاصل ضرب یک عدد 32 بیت می‌باشد که با استفاده از اپراتور >> 15 بیت اول آن را دور می‌زیم تا بیت‌های پر ارزش‌تر آن را در متغیر prod نگه داریم. سپس بررسی می‌کنیم که نتیجه‌ی نهایی مثبت است یا منفی؟ برای اینکه نتیجه‌ی نهایی منفی باشد باید تنها یکی از اپرنده‌ها منفی باشد بدین منظور شرط  $(\text{neg1} == 0 \&\& \text{neg2} == 1) || (\text{neg1} == 1 \&\& \text{neg2} == 0)$  را قرار می‌دهیم و در صورت برقرار بودن این شرط بدین معنی است که حاصل ضرب منفی می‌باشد. و مقدار آن برابر است با  $\text{prod} = 65536 - \text{prod}$  و این مقدار به عنوان خروجی تابع Fix\_mul\_16 به بیرون از تابع پاس داده می‌شود.

## fix\_add\_16:

```
unsigned int fix_add_16(unsigned int num1, unsigned int num2){
    int temp1,temp2;
    int neg1 = 0;
    int neg2 = 0;
    unsigned int sum;

    if (num1 > 32767){
        if (num1 > 65535) {
            num1 = 65535>>1;
        }
        neg1 = 1;
        temp1 = (0-num1+65536)>>1;
    } else {
        temp1 = (int)(num1)>>1;
    }
    if (num2 > 32767){
        if (num2 > 65535) {
            num2 = 65535>>1;
        }
        neg2 = 1;
        temp2 = (0-num2+65536)>>1;
    } else {
        temp2 = (int)(num2)>>1;
    }

    if (temp1 > temp2) {
        if (neg1 == 1 && neg2 == 0){
            sum = 65536-(temp1 - temp2);
        } else if (neg1 == 0 && neg2 == 1) {
            sum = temp1-temp2;
        } else if (neg1 == 1 && neg2 == 1){
            sum = 65536-(temp1+temp2);
        } else {
            sum = temp1+temp2;
        }
    } else {
        if (neg2 == 1 && neg1 == 0){
            sum = 65536-(temp2 - temp1);
        } else if (neg2 == 0 && neg1 == 1) {
            sum = temp2-temp1;
        } else if (neg1 == 1 && neg2 == 1){
            sum = 65536-(temp1+temp2);
        } else {
            sum = temp1+temp2;
        }
    }
    return sum;
}
```

در تابع fix\_add\_16 دو اپرنده unsigned integer دریافت می‌شود و سپس عملیات جمع با دیدگاه باینری و علامت دار انجام می‌شود. در ابتدا متغیرهای temp1 و temp2 برای انجام پردازش بروی اپرنده اول و اپرنده دوم تعریف می‌شود.

در ادامه متغیرهای neg1 و neg2 به عنوان یک flag تعریف می‌شود که این flag نشان دهنده‌ی منفی بودن و یا منفی نبودن اپرنده‌ها می‌باشد.

در ادامه عملیات را برای اپرنده اول نشان می‌دهیم، برای اپرنده دوم نیز عینتاً این عملیات تکرار می‌شود.

در خط اول توسط یک دستور شرطی if بررسی می کنیم که آیا بیت 16 اپرند اول یک می باشد یا خیر در صورت یک بودن یعنی اپرند اول منفی مبادله و در غیر این صورت اپرند اول عددی مثبت است. بنابراین این بیت 16 بیت علامت ما می باشد.  
برای بررسی این شرط باید مقدار اپرند اول را با  $2^{15}$  مقایسه نمود، در صورتی که بزرگ تر باشد یعنی بیت 16 ام برابر با 1 است و در غیر این صورت یعنی بیت 16 ام صفر می باشد.

در ادامه دستور شرطی دیگری وجود دارد که بررسی می کند مقدار اپرند اول از مقدار  $2^{16}$  بزرگتر نباشد، اگر این شرط برقرار باشد یعنی مقدار اپرند اول از  $2^{16}$  بزرگ تر است و تابع مقدار اپرند اول را یک بیت به سمت راست شیفت می دهد و این مقدار را به عنوان پیشفرض خود در نظر می گیرد.  
در ادامه با توجه به اینکه دستور شرطی اول برقرار بود و وارد بدنه if شده ایم و اپرند اول منفی می باشد پس مقدار neg1 را برابر با 1 می کنیم که نشان دهنده این است که اپرند اول منفی می باشد و مقدار خود اپرند را از طریق رابطه  $\text{temp1} = (0 - \text{num1} + 65536) \gg 1$  محاسبه می نماییم.

اگر دستور شرطی اول برقرار نباشد و مقدار اپرند اول از  $2^{15}$  بزرگ تر نباشد بدین معنی است که مقدار اپرند اول مثبت است و neg1 برابر با صفر می باشد و مقدار اپرند اول برابر با 1  $\text{temp1} = (\text{int})(\text{num1}) \gg 1$  می باشد.

مراحل بالا برای اپرند دوم نیز تکرار می شود.

در ادامه ی کد عملیات sum انجام می شود، در ابتدا بررسی می شود که مقدار اپرند اول بیشتر است یا مقدار اپرند دوم؟  
این عملیات از آن جهت انجام می شود که ممکن است یکی یا جفت اپرند ها منفی باشند و باید با توجه به این مورد عملیات sum را انجام داد.

بدین منظور حالتی را شرح می دهیم که مقدار اپرند اول از مقدار اپرند دوم بزرگتر باشد:  
به 4 حالت تقسیم می شود و هر حالت محاسبات مختص به خود را انجام می دهد:

(الف) اپرند اول منفی باشد و اپرند دوم مثبت باشد:

$$\text{sum} = 65536 - (\text{temp1} - \text{temp2})$$

(ب) اپرند اول مثبت باشد و اپرند دوم منفی باشد:

$$\text{sum} = \text{temp1} - \text{temp2}$$

(ج) هر دو اپرند منفی باشند:

$$\text{sum} = 65536 - (\text{temp1} + \text{temp2})$$

(د) هر دو اپرند مثبت باشند:

$$\text{sum} = \text{temp1} + \text{temp2}$$

و به همین ترتیب برای حالتی که مقدار اپرند دوم از اپرند اول بزرگ تر باشد نیز محاسبات مشابهی را داریم.

در نهاین مقدار sum به عنوان خروجی تابع در نظر گرفته می شود و به عنوان خروجی به بیرون از تابع پاس داده می شود.

## Part 3 → Question7 & 8

اضافه کردن functional unit

**Operation properties**

<b>Operation properties</b>	<b>Operation description</b>
Name: COS16	
<input type="checkbox"/> Reads memory <input type="checkbox"/> Writes memory	
<input type="checkbox"/> Can trap <input type="checkbox"/> Has side effects	
<input type="checkbox"/> Clocked	

**Affected by**

operation
-----------

**Operation inputs**

operand	type	element width	element
1	UIntWord	32	1

**Affects**

operation
-----------

**Operation outputs**

operand	type	element width	element
2	UIntWord	32	1

Operation behavior module not defined. [Open](#) [Open DAG](#) [OK](#) [Cancel](#)

**Operation properties**

<b>Operation properties</b>	<b>Operation description</b>
Name: MUL_16_FIX	
<input type="checkbox"/> Reads memory <input type="checkbox"/> Writes memory	
<input type="checkbox"/> Can trap <input type="checkbox"/> Has side effects	
<input type="checkbox"/> Clocked	

**Affected by**

operation
-----------

**Operation inputs**

operand	type	element width	element
1	UIntWord	32	1
2	UIntWord	32	1

**Affects**

operation
-----------

**Operation outputs**

operand	type	element width	element
3	UIntWord	32	1

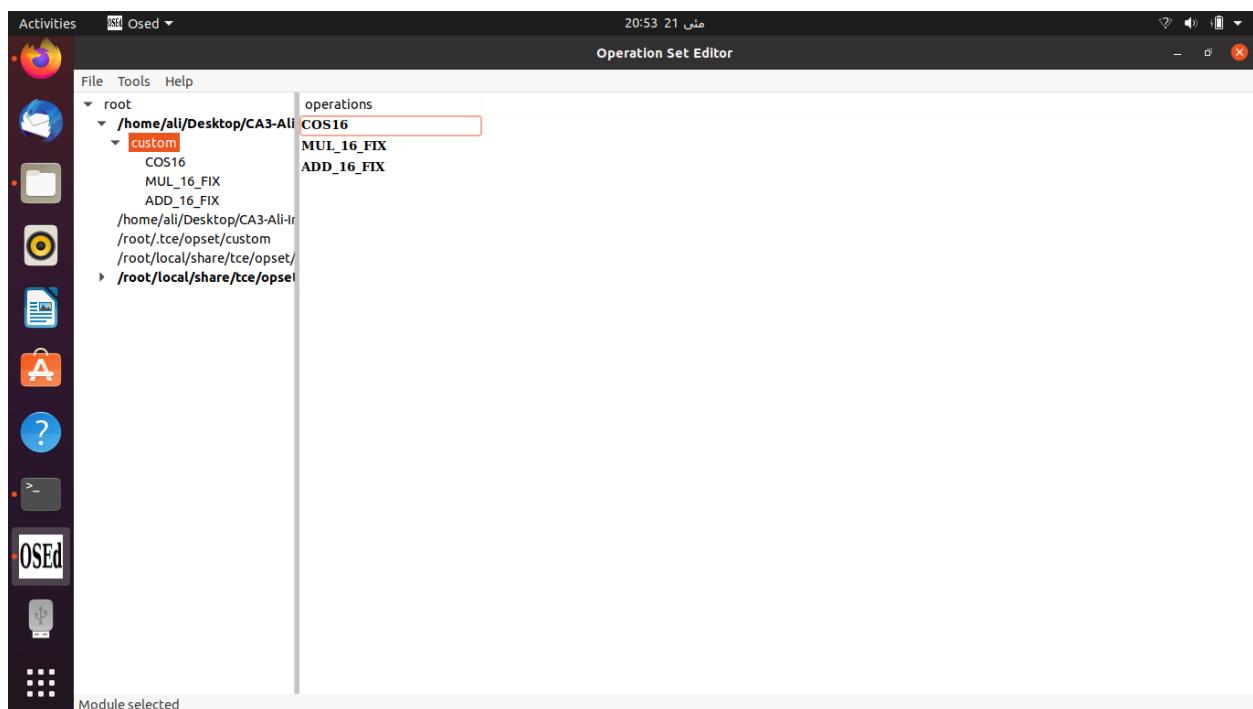
Operation behavior module not defined. [Open](#) [Open DAG](#) [OK](#) [Cancel](#)

### Operation properties

<b>Operation properties</b> Name: <input type="text" value="ADD_16_FIX"/> <input type="checkbox"/> Reads memory <input type="checkbox"/> Writes memory <input type="checkbox"/> Can trap <input type="checkbox"/> Has side effects <input type="checkbox"/> Clocked	<b>Operation description</b> <div style="border: 1px solid black; height: 100px; width: 100%;"></div>												
<b>Affected by</b> operation <div style="border: 1px solid black; height: 100px; width: 100%;"></div> <p style="margin-top: 10px;"> <a href="#">ABS</a> <a href="#">Add</a> <a href="#">Delete</a> </p>													
<b>Operation inputs</b> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th>operand</th> <th>type</th> <th>element width</th> <th>element</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>UIntWord</td> <td>32</td> <td>1</td> </tr> <tr> <td>2</td> <td>UIntWord</td> <td>32</td> <td>1</td> </tr> </tbody> </table> <p style="margin-top: 10px;"> <a href="#">Add...</a> <a href="#">Modify...</a> <a href="#">Delete</a> </p>		operand	type	element width	element	1	UIntWord	32	1	2	UIntWord	32	1
operand	type	element width	element										
1	UIntWord	32	1										
2	UIntWord	32	1										
<b>Affects</b> operation <div style="border: 1px solid black; height: 100px; width: 100%;"></div> <p style="margin-top: 10px;"> <a href="#">ABS</a> <a href="#">Add</a> <a href="#">Delete</a> </p>													
<b>Operation outputs</b> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th>operand</th> <th>type</th> <th>element width</th> <th>element</th> </tr> </thead> <tbody> <tr> <td>3</td> <td>UIntWord</td> <td>32</td> <td>1</td> </tr> </tbody> </table> <p style="margin-top: 10px;"> <a href="#">Add...</a> <a href="#">Modify...</a> <a href="#">Delete</a> </p>		operand	type	element width	element	3	UIntWord	32	1				
operand	type	element width	element										
3	UIntWord	32	1										

Operation behavior module not defined. [Open](#) [Open DAG](#)

[OK](#) [Cancel](#)



## بررسی صحت عملکرد مازول cos16

$$input = 0 \rightarrow address = 0 \rightarrow value = 32138 \rightarrow \cos(0) \approx \frac{32138}{2^{15}} = 0.9808$$

Simulate Operation Behavior: COS16

Input values		Output values		
operand	value	operand	value	Clock count: 0
1	0	2	32138	
<input type="text" value="0"/>	<input type="button" value="int"/>	<input type="button" value="Update"/>		
Format: <input type="button" value="unsigned int"/>				
<input type="button" value="Reset"/> <input type="button" value="Trigger"/> <input type="button" value="Advance Clock"/> <input type="button" value="Show registers"/>				<input type="button" value="OK"/>

$$input = 1 \rightarrow address = 0 \rightarrow value = 32138 \rightarrow \cos\left(\frac{\pi}{16}\right) = \frac{32138}{2^{15}} = 0.9808$$

Simulate Operation Behavior: COS16

Input values		Output values		
operand	value	operand	value	Clock count: 0
1	1	2	32138	
<input type="text" value="1"/>	<input type="button" value="int"/>	<input type="button" value="Update"/>		
Format: <input type="button" value="unsigned int"/>				
<input type="button" value="Reset"/> <input type="button" value="Trigger"/> <input type="button" value="Advance Clock"/> <input type="button" value="Show registers"/>				<input type="button" value="OK"/>

$$input = 3 \rightarrow address = 1 \rightarrow value = 27246 \rightarrow \cos\left(\frac{3\pi}{16}\right) = \frac{27246}{2^{15}} = 0.8315$$

Simulate Operation Behavior: COS16

Input values		Output values		
operand	value	operand	value	Clock count: 0
1	3	2	27246	
<input type="text" value="3"/>	<input type="button" value="int"/>	<input type="button" value="Update"/>		
Format: <input type="button" value="unsigned int"/>				
<input type="button" value="Reset"/> <input type="button" value="Trigger"/> <input type="button" value="Advance Clock"/> <input type="button" value="Show registers"/>				<input type="button" value="OK"/>

$$input = 5 \rightarrow address = 2 \rightarrow value = 18205 \rightarrow \cos\left(\frac{5\pi}{16}\right) = \frac{18205}{2^{15}} = 0.5556$$

Simulate Operation Behavior: COS16

Input values		Output values		
operand	value	operand	value	Clock count:
1	5	2	18205	0

Format: unsigned int

Reset Trigger Advance Clock Show registers OK

$$input = 7 \rightarrow address = 3 \rightarrow value = 18205 \rightarrow \cos\left(\frac{5\pi}{16}\right) = \frac{18205}{2^{15}} = 0.5556$$

Simulate Operation Behavior: COS16

Input values		Output values		
operand	value	operand	value	Clock count:
1	7	2	6393	0

Format: unsigned int

Reset Trigger Advance Clock Show registers OK

$$input = 9 \rightarrow address = 0 \rightarrow value = 65536 - 32138 = 33398 \rightarrow \cos\left(\frac{9\pi}{16}\right) \neq \frac{33398}{2^{15}} = 1.0192$$

inverse = 1

Simulate Operation Behavior: COS16

Input values		Output values		
operand	value	operand	value	Clock count:
1	9	2	33398	0

Format: unsigned int

Reset Trigger Advance Clock Show registers OK

$$input = 11 \rightarrow address = 0 \rightarrow value = 65536 - 32138 = 33398 \rightarrow \cos\left(\frac{11\pi}{16}\right) \neq \frac{33398}{2^{15}} = 1.0192$$

*inverse = 1*

Simulate Operation Behavior: COS16

Input values		Output values		Clock count:
operand	value	operand	value	0
1	11	2	33398	

Format: unsigned int

Reset Trigger Advance Clock Show registers OK

$$input = 13 \rightarrow address = 0 \rightarrow value = 65536 - 32138 = 33398 \rightarrow \cos\left(\frac{13\pi}{16}\right) \neq \frac{33398}{2^{15}} = 1.0192$$

*inverse = 1*

Simulate Operation Behavior: COS16

Input values		Output values		Clock count:
operand	value	operand	value	0
1	13	2	33398	

Format: unsigned int

Reset Trigger Advance Clock Show registers OK

$$input = 15 \rightarrow address = 0 \rightarrow value = 65536 - 32138 = 33398 \rightarrow \cos\left(\frac{15\pi}{16}\right) \approx \frac{33398}{2^{15}} = 1.0192$$

*inverse = 1*

Simulate Operation Behavior: COS16

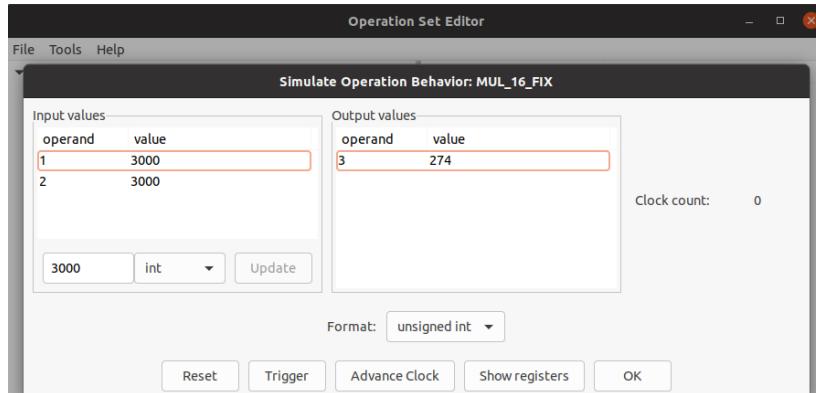
Input values		Output values		Clock count:
operand	value	operand	value	0
1	15	2	33398	

Format: unsigned int

Reset Trigger Advance Clock Show registers OK

## بررسی صحت عملکرد مازول mul\_fix\_16

$$in_1 = 3000, in_2 = 3000 \rightarrow in_1 * in_2 = 9000000 \gg 15 = 274$$

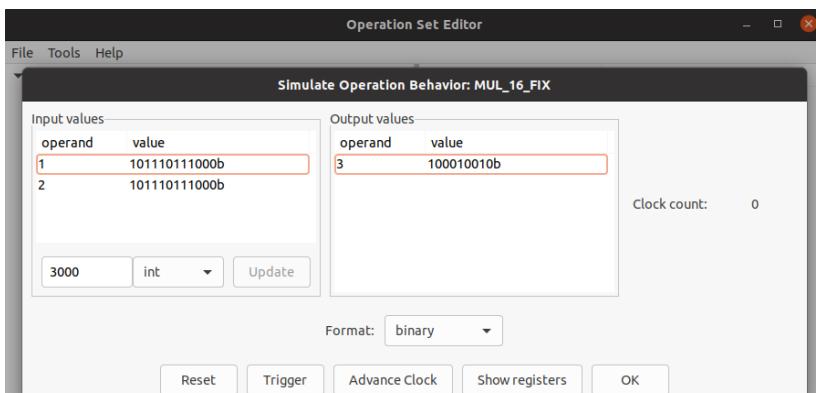


## Bit Shift Calculator

Perform bit shift operations with decimal, hexadecimal, binary and octal numbers

The screenshot shows the 'Bit Shift Calculator' interface. On the left, under 'Number Base', 'Decimal (10)' is selected. The 'Number' field contains '9000000'. Under 'Shift Direction', 'Right' is selected. The 'Bits to Shift' field contains '15'. On the right, the results are displayed in four formats: Decimal (274), Binary (100010010), Hexadecimal (112), and Octal (422).

به علت اینکه هر رو عدد مثبت می باشند، نتیجه‌ی نهایی نیز مثبت است و در نمایش باینری بیت 16 ام که بیت sign می باشد صفر شده است. (9 بیت نمایش داده شده است و 7 بیت lsb آن صفر می باشند.)



$in_1 = 33000, in_2 = 3000 \rightarrow in_1 * in_2 = 65536 - ((65536 - 33000) * (3000) \gg 15) = 62558$

Simulate Operation Behavior: MUL\_16\_FIX

Input values		Output values	
operand	value	operand	value
1	33000	3	62558
2	3000		

Clock count: 0

Format: unsigned int

Buttons: Reset, Trigger, Advance Clock, Show registers, OK

# Bit Shift Calculator

Perform bit shift operations with decimal, hexadecimal, binary and octal numbers

Number Base		
Decimal (10)	▼	2978
Number		101110100010
97608000		ba2
Shift Direction		Octal
>> Right	▼	5642
Bits to Shift		
15		

اپنند اول منفی و اپنند دوم مثبت می باشد، در نتیجه حاصل نهایی منفی باید باشد که در نمایش باینری زیر نیز بیت 16 ام که بیت sign است برای با 1 شده است.

Simulate Operation Behavior: MUL\_16\_FIX

Input values		Output values	
operand	value	operand	value
1	1000000011101000b	3	1111010001011110b
2	101110111000b		

Clock count: 0

Format: binary

3000 int Update

Reset Trigger Advance Clock Show registers OK

$$in_1 = 33000, in_2 = 33000 \rightarrow in_1 * in_2 = 65536 - ((65536 - 33000) * (65536 - 33000) \gg 15) = 32305$$

**Simulate Operation Behavior: MUL\_16\_FIX**

Input values		Output values		Clock count:
operand	value	operand	value	
1	33000	3	32305	0
2	33000			

Format: **unsigned int**

**Buttons:** Reset, Trigger, Advance Clock, Show registers, OK

## Bit Shift Calculator

Perform bit shift operations with decimal, hexadecimal, binary and octal numbers

Number Base			
Decimal (10)		32305 11111000110001 7e31 77061	
Number		1058591296	
Shift Direction		>> Right	
Bits to Shift		15	

هر دو اپرند منفی می باشند بنابراین حاصل ضرب نهایی باید مثبت باشد که همنظور که از نمایش باینری زیر مشخص است بیت علامت که بیت 16 می باشد 0 می باشد که بدین معنی است که حاصل نهایی مثبت است.

**Simulate Operation Behavior: MUL\_16\_FIX**

Input values		Output values		Clock count:
operand	value	operand	value	
1	1000000011101000b	3	11111000110001b	0
2	1000000011101000b			

Format: **binary**

**Buttons:** Reset, Trigger, Advance Clock, Show registers, OK

## بررسی صحت عملکرد مازول add\_fix\_16

$$in_1 = 3000, in_2 = 3000 \rightarrow value = (3000 \gg 1) + (3000 \gg 1) = 1500 + 1500 = 3000$$

Simulate Operation Behavior: ADD\_16\_FIX

Input values		Output values	
operand	value	operand	value
1	3000	3	3000
2	3000		

Clock count: 0

Format: signed int

Reset Trigger Advance Clock Show registers OK

## Bit Shift Calculator

Perform bit shift operations with decimal, hexadecimal, binary and octal numbers

Number Base	
Decimal (10)	1500
Binary	10111011100
Hexadecimal	5dc
Octal	2734

Number: 3000

Shift Direction: >> Right

Bits to Shift: 1

هر دو اپرند مثبت می باشند پس بیت علامت (بیت 16 ام) باید برابر با 0 باشد:

Simulate Operation Behavior: ADD\_16\_FIX

Input values		Output values	
operand	value	operand	value
1	10111011100b	3	10111011100b
2	10111011100b		

Clock count: 0

Format: binary

Reset Trigger Advance Clock Show registers OK

$$in_1 = 3000, in_2 = 33000 \rightarrow value = 65536 - [((-33000 + 65536) \gg 1) - (3000 \gg 1)] \\ \rightarrow 65536 - (16268 - 1500) = 50768$$

Simulate Operation Behavior: ADD\_16\_FIX

Input values		Output values				
operand	value	operand	value	Clock count:		
1	3000	3	50768	0		
2	33000					
<input type="button" value="33000"/> <input type="button" value="int"/> <input type="button" value="Update"/>						
		<input type="button" value="Format: unsigned int"/>				
		<input type="button" value="Reset"/>	<input type="button" value="Trigger"/>	<input type="button" value="Advance Clock"/>	<input type="button" value="Show registers"/>	<input type="button" value="OK"/>

## Bit Shift Calculator

Perform bit shift operations with decimal, hexadecimal, binary and octal numbers

<p>Number Base</p> <p>Decimal (10)</p> <p>Number</p> <p>32536</p> <p>Shift Direction</p> <p>&gt;&gt; Right</p> <p>Bits to Shift</p> <p>1</p>	<p>Decimal</p> <p>16268</p> <p>Binary</p> <p>11111110001100</p> <p>Hexadecimal</p> <p>3f8c</p> <p>Octal</p> <p>37614</p>
--	--

حاصل نهایی منفی است و بیت 16 ام برابر با 1 می باشد:

Simulate Operation Behavior: ADD\_16\_FIX

Input values		Output values				
operand	value	operand	value	Clock count:		
1	101110111000b	3	1100011001010000b	0		
2	1000000011101000b					
<input type="button" value="33000"/> <input type="button" value="int"/> <input type="button" value="Update"/>						
		<input type="button" value="Format: binary"/>				
		<input type="button" value="Reset"/>	<input type="button" value="Trigger"/>	<input type="button" value="Advance Clock"/>	<input type="button" value="Show registers"/>	<input type="button" value="OK"/>

$$in_1 = 50000, in_2 = 33000 \rightarrow value = 65536 - [((-50000 + 65536) \gg 1) - (3000 \gg 1)] \\ \rightarrow 65536 - (7768 - 1500) = 59268$$

Simulate Operation Behavior: ADD\_16\_FIX

Input values		Output values		Clock count:
operand	value	operand	value	
1	50000	3	59268	
2	3000			

Format: unsigned int

Buttons: Reset, Trigger, Advance Clock, Show registers, OK

## Bit Shift Calculator

Perform bit shift operations with decimal, hexadecimal, binary and octal numbers

Number Base		Decimal	
Decimal (10)	7768	Binary	1111001011000
Number	1e58	Hexadecimal	1e58
15536	17130	Octal	17130

Shift Direction: >> Right

Bits to Shift: 1

حاصل نهایی منفی است پس باید بیت 16 ام برابر با 1 باشد:

Simulate Operation Behavior: ADD\_16\_FIX

Input values		Output values		Clock count:
operand	value	operand	value	
1	1100001101010000b	3	111001110000100b	
2	101110111000b			

Format: binary

Buttons: Reset, Trigger, Advance Clock, Show registers, OK

$$\begin{aligned}in_1 = 33000, in_2 = 33000 \rightarrow value &= 65536 - [((-33000 + 65536) \gg 1) + ((-33000 + 65536) \gg 1)] \\&\rightarrow 65536 - (2 * 16268) = 33000\end{aligned}$$

Simulate Operation Behavior: ADD\_16\_FIX

Input values		Output values	
operand	value	operand	value
1	3000	3	50768
2	33000		

Clock count: 0

Format: unsigned int

Buttons: Reset, Trigger, Advance Clock, Show registers, OK

# Bit Shift Calculator

Perform bit shift operations with decimal, hexadecimal, binary and octal numbers

Number Base		
Decimal (10)		16268
Number		11111110001100
32536		3f8c
Shift Direction		Octal
>> Right		37614
Bits to Shift		
1		

حاصل نهایی منفی است پس باید بیت 16 ام برابر با 1 باشد:

Simulate Operation Behavior: ADD\_16\_FIX

Input values		Output values	
operand	value	operand	value
1	10110111000b	3	1100011001010000b
2	100000011101000b		

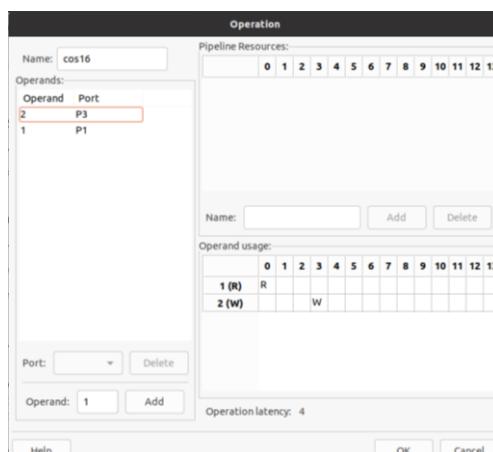
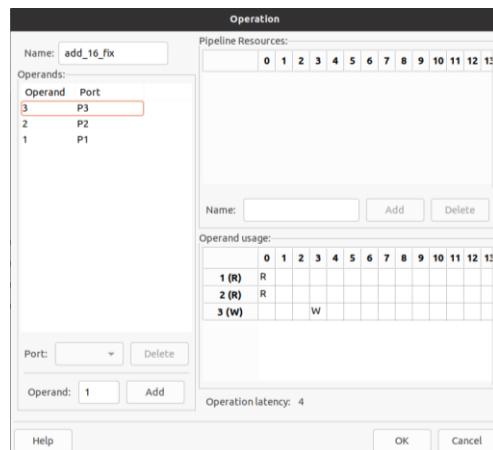
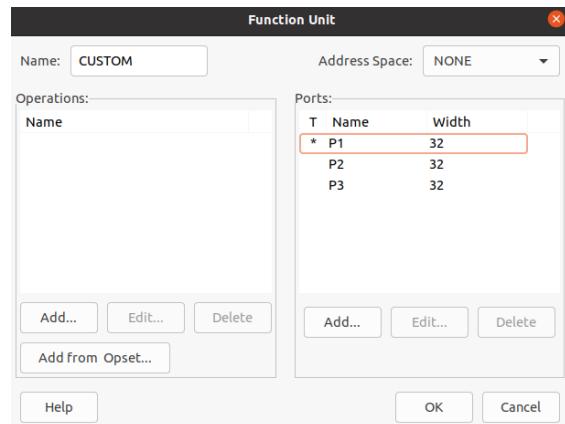
Clock count: 0

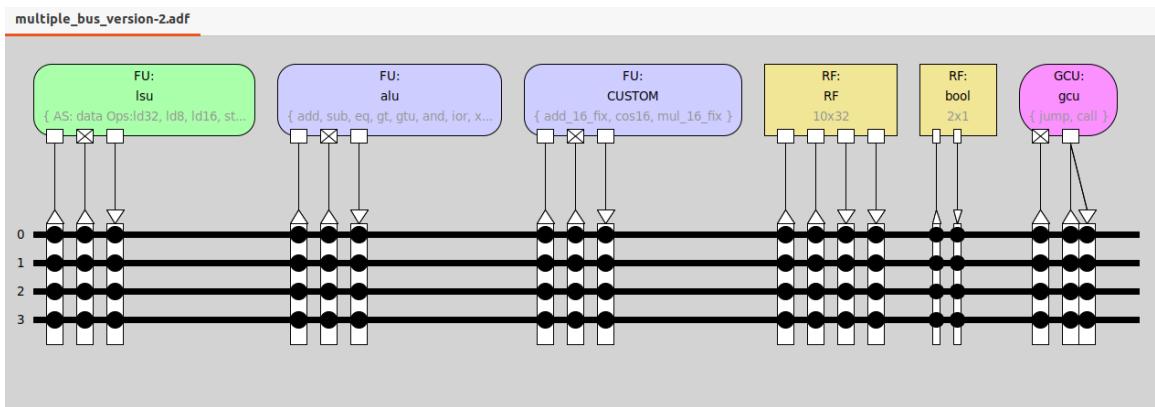
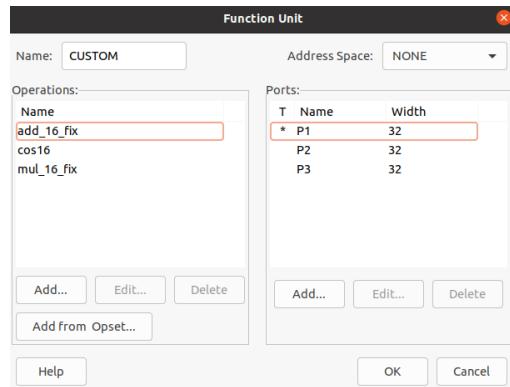
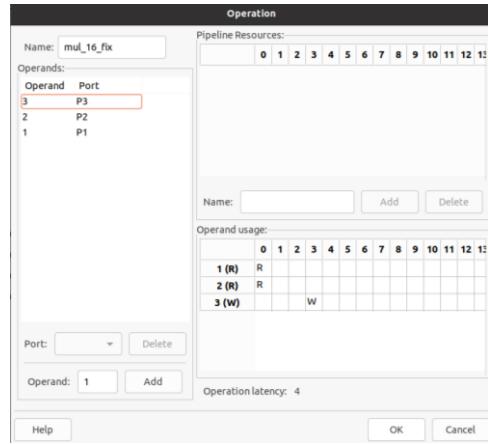
33000 int ▾ Update

Format: binary ▾

Reset Trigger Advance Clock Show registers OK

اضافه کردن مازول های ساخته شده با adf. قبلی:





## Part 3 → Question9

استفاده از مازول cos16 ساخت شده:

تغییرات کد به نحوی که از مازول cos\_16 ساخته شده استفاده گردد:

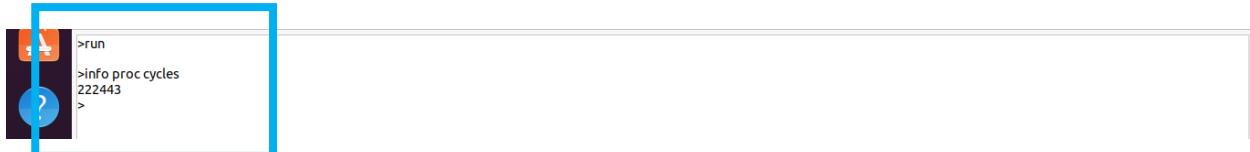
```
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?  
dct_8x8_16_bit.c x  
79  
80 first calculate rows */  
81  
82     unsigned int cos_x_ans;  
83     unsigned int cos_3_x_ans;  
84     unsigned int cos_5_x_ans;  
85     unsigned int cos_7_x_ans;  
86     unsigned int cos_9_x_ans;  
87     unsigned int cos_11_x_ans;  
88     unsigned int cos_13_x_ans;  
89     unsigned int cos_15_x_ans;  
90  
91     for (y = 0; y < 8; y++) {  
92     /*#ifndef DEBUG_SINGLE_MEMORY_LOCATION  
93     printf("ROW %ld\n", y);  
94     #endif */  
95     for (x = 0; x < 8; x++) {  
96         _TCE_COS16(x, cos_x_ans);  
97         _TCE_COS16(3 * x, cos_3_x_ans);  
98         _TCE_COS16(5 * x, cos_5_x_ans);  
99         _TCE_COS16(7 * x, cos_7_x_ans);  
100        _TCE_COS16(9 * x, cos_9_x_ans);  
101        _TCE_COS16(11 * x, cos_11_x_ans);  
102        _TCE_COS16(13 * x, cos_13_x_ans);  
103        _TCE_COS16(15 * x, cos_15_x_ans);  
104  
105        a = fix_mul_16(fy[0], cos_x_ans);  
106        b = fix_mul_16(fy[1], cos_3_x_ans);  
107        c = fix_mul_16(fy[2], cos_5_x_ans);  
108        d = fix_mul_16(fy[3], cos_7_x_ans);  
109        e = fix_mul_16(fy[4], cos_9_x_ans);  
110        f = fix_mul_16(fy[5], cos_11_x_ans);  
111        g = fix_mul_16(fy[6], cos_13_x_ans);  
112        h = fix_mul_16(fy[7], cos_15_x_ans);  
113        j = fix_mul_16(fy[1], cos_15_x_ans);  
114  
115     /*#ifndef DEBUG_MULTIPLIER  
116     printf(" a: %d . f: %d . cos: %dn", a,fy[0].cos16(x));  
117     printf(" b: %d . f: %d . cos: %dn", b,fy[1].cos16(3*x));  
118     printf(" c: %d . f: %d . cos: %dn", c,fy[2].cos16(5*x));  
119     printf(" d: %d . f: %d . cos: %dn", d,fy[3].cos16(7*x));  
120     printf(" e: %d . f: %d . cos: %dn", e,fy[4].cos16(9*x));  
121     printf(" f: %d . f: %d . cos: %dn", f,fy[5].cos16(11*x));  
122     printf(" g: %d . f: %d . cos: %dn", g,fy[6].cos16(13*x));  
123     printf(" h: %d . f: %d . cos: %dn", h,fy[7].cos16(15*x));  
124     #endif*/  
125  
C source file length : 11,367 lines : 387 Ln : 42 Col : 62 Pos : 1,090 Windows (CR LF) UTF-8 INS x  
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?  
dct_8x8_16_bit.c x  
166     printf(" memory at [%ld][%ld] : %dn", y,x,F[y][x]);  
167 }  
168  
169 #endif  
170 /*/  
171     /* second columns  
172     */  
173  
174     for (y = 0; y < 8; y++) {  
175     /*#ifndef DEBUG_SINGLE_MEMORY_LOCATION  
176     printf("COLUMN %ld\n", y);  
177     #endif*/  
178     for (x = 0; x < 8; x++) {  
179         _TCE_COS16(x, cos_x_ans);  
180         _TCE_COS16(3 * x, cos_3_x_ans);  
181         _TCE_COS16(5 * x, cos_5_x_ans);  
182         _TCE_COS16(7 * x, cos_7_x_ans);  
183         _TCE_COS16(9 * x, cos_9_x_ans);  
184         _TCE_COS16(11 * x, cos_11_x_ans);  
185         _TCE_COS16(13 * x, cos_13_x_ans);  
186         _TCE_COS16(15 * x, cos_15_x_ans);  
187  
188         a = fix_mul_16(fy[0], cos_x_ans);  
189         b = fix_mul_16(fy[1], cos_3_x_ans);  
190         c = fix_mul_16(fy[2], cos_5_x_ans);  
191         d = fix_mul_16(fy[3], cos_7_x_ans);  
192         e = fix_mul_16(fy[4], cos_9_x_ans);  
193         f = fix_mul_16(fy[5], cos_11_x_ans);  
194         g = fix_mul_16(fy[6], cos_13_x_ans);  
195         h = fix_mul_16(fy[7], cos_15_x_ans);  
196         j = fix_mul_16(fy[1], cos_15_x_ans);  
197     /*/  
198     #ifdef DEBUG_MULTIPLIER  
199     printf(" a: %d . f: %d . cos: %dn", a,fy[0].cos16(x));  
200     printf(" b: %d . f: %d . cos: %dn", b,fy[1].cos16(3*x));  
201     printf(" c: %d . f: %d . cos: %dn", c,fy[2].cos16(5*x));  
202     printf(" d: %d . f: %d . cos: %dn", d,fy[3].cos16(7*x));  
203     printf(" e: %d . f: %d . cos: %dn", e,fy[4].cos16(9*x));  
204     printf(" f: %d . f: %d . cos: %dn", f,fy[5].cos16(11*x));  
205     printf(" g: %d . f: %d . cos: %dn", g,fy[6].cos16(13*x));  
206     #endif*/  
207  
C source file length : 11,367 lines : 387 Ln : 42 Col : 62 Pos : 1,090 Windows (CR LF) UTF-8 INS x
```

کامپایل کردن برنامه:



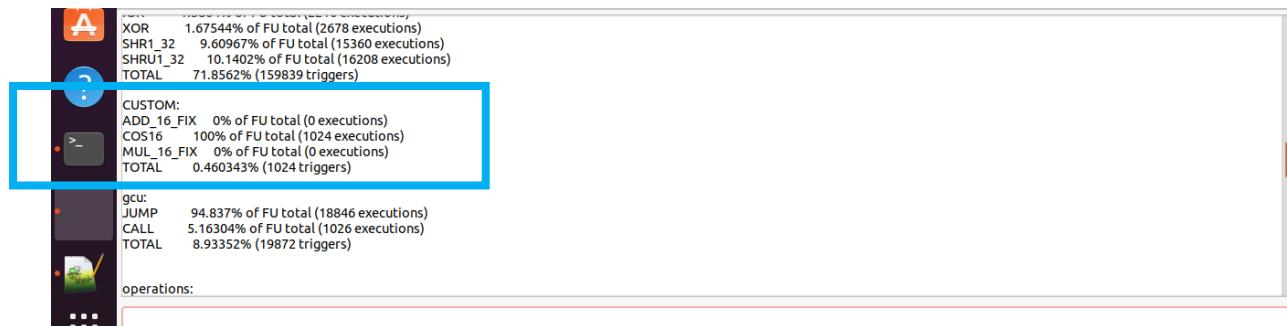
```
root@imangholi:/home/ali/Desktop/CA3-Ali-Imangholi-810197692/report/step-3/question-9/just-modify-cos# tcecc -O2 -a custom.adf -o dct.tpef dct
root@imangholi:/home/ali/Desktop/CA3-Ali-Imangholi-810197692/report/step-3/question-9/just-modify-cos#
```

تعداد سیکلی که طول می کشد کد اجرا شود:



```
>run
>info proc cycles
222443
>
```

.custom functional units اطلاعات



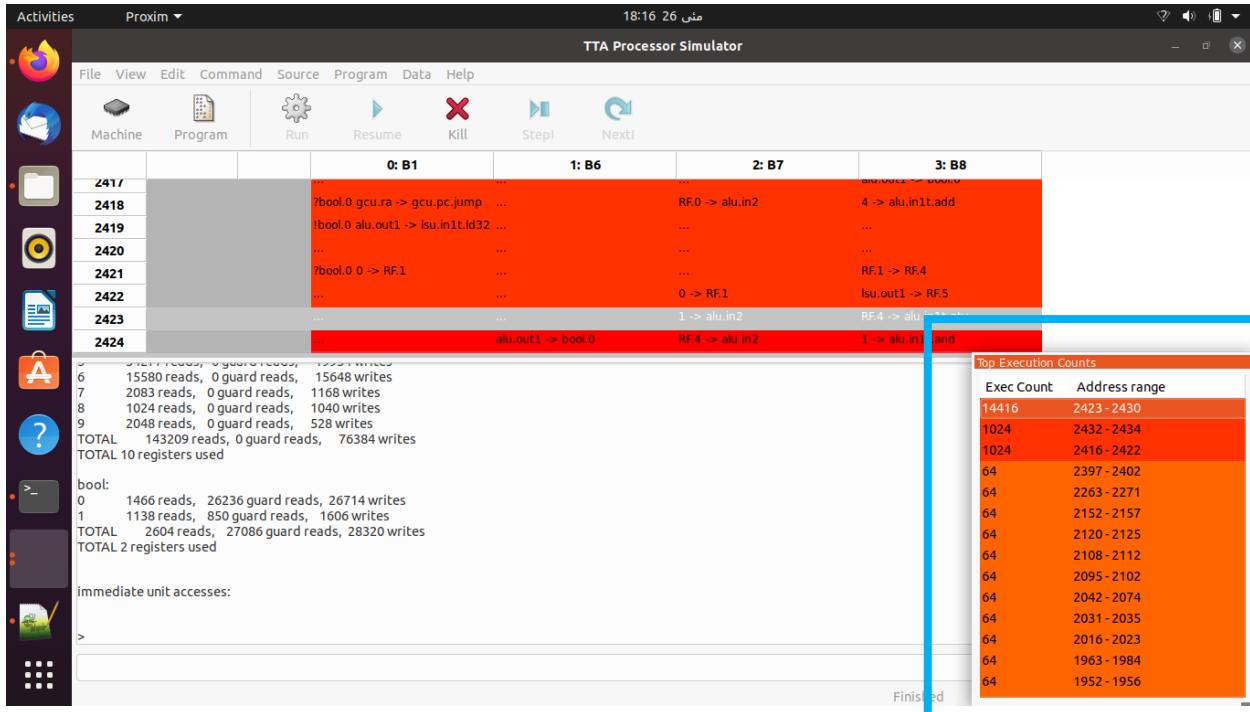
```
XOR 1.67544% of FU total (2678 executions)
SHR1_32 9.60967% of FU total (15360 executions)
SHRU1_32 10.1402% of FU total (16208 executions)
TOTAL 71.8562% (159839 triggers)

CUSTOM:
ADD_16_FIX 0% of FU total (0 executions)
COS16 100% of FU total (1024 executions)
MUL_16_FIX 0% of FU total (0 executions)
TOTAL 0.460343% (1024 triggers)

gcu:
JUMP 94.837% of FU total (18846 executions)
CALL 5.16304% of FU total (1026 executions)
TOTAL 8.93352% (19872 triggers)

operations:
```

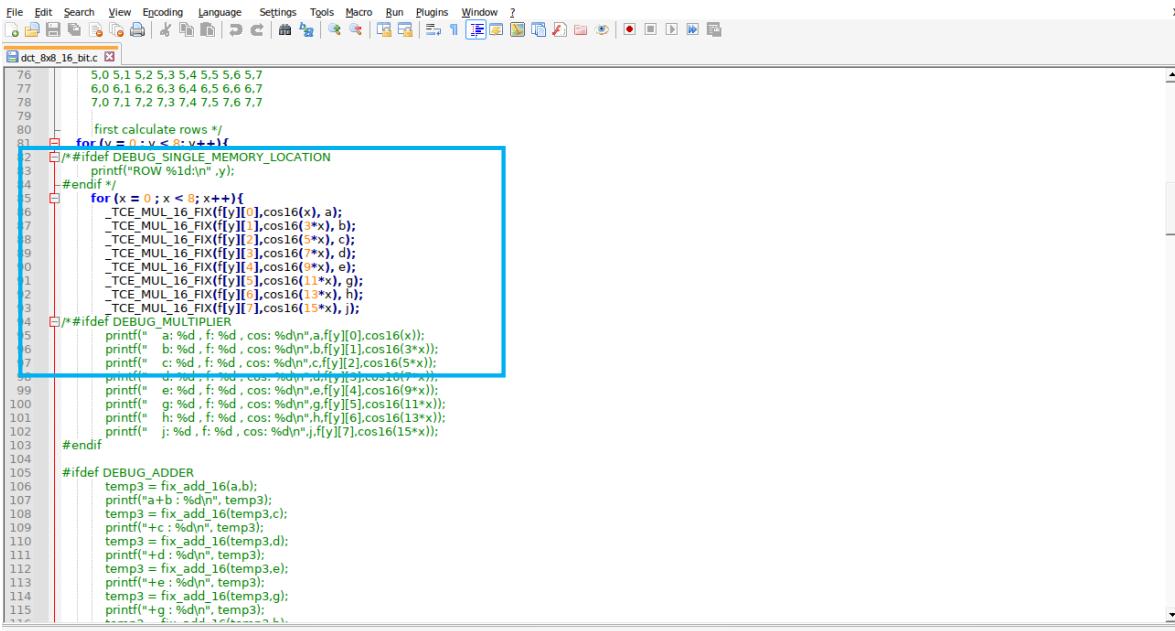
: Highlight top execution counts



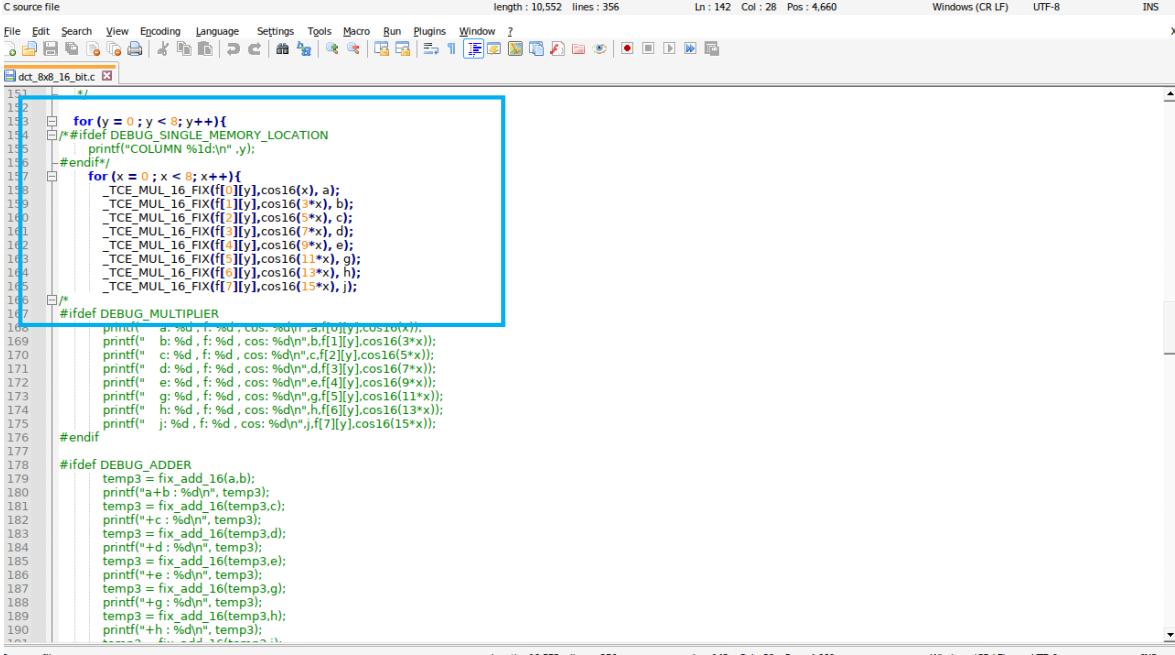
Exec Count	Address range
14416	2423 - 2430
1024	2432 - 2434
1024	2416 - 2422
64	2397 - 2402
64	2263 - 2271
64	2152 - 2157
64	2120 - 2125
64	2108 - 2112
64	2095 - 2102
64	2042 - 2074
64	2031 - 2035
64	2016 - 2023
64	1963 - 1984
64	1952 - 1956

## استفاده از مازول ساخت شده: MUL\_16\_FIX

تغییرات کد به نحوی که از مازول mul\_16\_fix ساخته شده استفاده گردد:



```
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?  
dct_8x8_16_bit.c  
76 5.0 5.1 5.2 5.3 5.4 5.5 5.6 5.7  
77 6.0 6.1 6.2 6.3 6.4 6.5 6.6 6.7  
78 7.0 7.1 7.2 7.3 7.4 7.5 7.6 7.7  
79  
80     first calculate rows */  
81     for (y = 0; y < 8; y++) {  
82         /*#ifdef DEBUG_SINGLE_MEMORY_LOCATION  
83             printf("ROW %d\n", y);  
84         #endif */  
85         for (x = 0; x < 8; x++) {  
86             TCE_MUL_16_FIX(fy[0], cos16(x), a);  
87             TCE_MUL_16_FIX(fy[1], cos16(3*x), b);  
88             TCE_MUL_16_FIX(fy[2], cos16(5*x), c);  
89             TCE_MUL_16_FIX(fy[3], cos16(7*x), d);  
90             TCE_MUL_16_FIX(fy[4], cos16(9*x), e);  
91             TCE_MUL_16_FIX(fy[5], cos16(11*x), g);  
92             TCE_MUL_16_FIX(fy[6], cos16(13*x), h);  
93             TCE_MUL_16_FIX(fy[7], cos16(15*x), i);  
94         /*#ifdef DEBUG_MULTIPLIER  
95             printf(" a: %d, f: %d, cos: %dn", a, fy[0], cos16(x));  
96             printf(" b: %d, f: %d, cos: %dn", b, fy[1], cos16(3*x));  
97             printf(" c: %d, f: %d, cos: %dn", c, fy[2], cos16(5*x));  
98             printf(" d: %d, f: %d, cos: %dn", d, fy[3], cos16(7*x));  
99             printf(" e: %d, f: %d, cos: %dn", e, fy[4], cos16(9*x));  
100            printf(" g: %d, f: %d, cos: %dn", g, fy[5], cos16(11*x));  
101            printf(" h: %d, f: %d, cos: %dn", h, fy[6], cos16(13*x));  
102            printf(" i: %d, f: %d, cos: %dn", j, fy[7], cos16(15*x));  
103        #endif  
104  
105        #ifdef DEBUG_ADDER  
106            temp3 = fix_add_16(a,b);  
107            printf("a+b : %dn", temp3);  
108            temp3 = fix_add_16(temp3,c);  
109            printf(" +c : %dn", temp3);  
110            temp3 = fix_add_16(temp3,d);  
111            printf(" +d : %dn", temp3);  
112            temp3 = fix_add_16(temp3,e);  
113            printf(" +e : %dn", temp3);  
114            temp3 = fix_add_16(temp3,g);  
115            printf(" +g : %dn", temp3);  
116        #endif  
117  
118        #ifdef DEBUG_MULTIPLIER  
119            printf(" a: %d, f: %d, cos: %dn", a, fy[0], cos16(x));  
120            printf(" b: %d, f: %d, cos: %dn", b, fy[1], cos16(3*x));  
121            printf(" c: %d, f: %d, cos: %dn", c, fy[2], cos16(5*x));  
122            printf(" d: %d, f: %d, cos: %dn", d, fy[3], cos16(7*x));  
123            printf(" e: %d, f: %d, cos: %dn", e, fy[4], cos16(9*x));  
124            printf(" g: %d, f: %d, cos: %dn", g, fy[5], cos16(11*x));  
125            printf(" h: %d, f: %d, cos: %dn", h, fy[6], cos16(13*x));  
126            printf(" i: %d, f: %d, cos: %dn", j, fy[7], cos16(15*x));  
127        #endif  
128  
129        #ifdef DEBUG_ADDER  
130            temp3 = fix_add_16(a,b);  
131            printf("a+b : %dn", temp3);  
132            temp3 = fix_add_16(temp3,c);  
133            printf(" +c : %dn", temp3);  
134            temp3 = fix_add_16(temp3,d);  
135            printf(" +d : %dn", temp3);  
136            temp3 = fix_add_16(temp3,e);  
137            printf(" +e : %dn", temp3);  
138            temp3 = fix_add_16(temp3,g);  
139            printf(" +g : %dn", temp3);  
140            temp3 = fix_add_16(temp3,h);  
141            printf(" +h : %dn", temp3);  
142        #endif  
143    }  
144  
145    /*
```



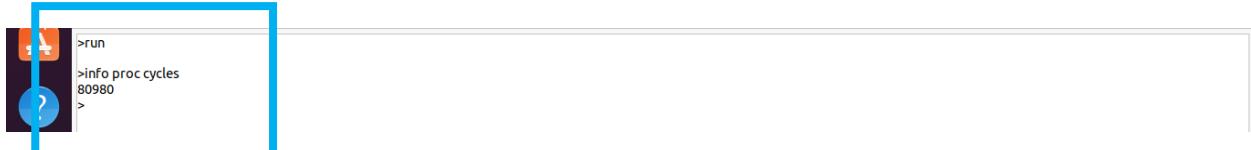
```
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?  
dct_8x8_16_bit.c  
151 */  
152     for (y = 0; y < 8; y++) {  
153         /*#ifdef DEBUG_SINGLE_MEMORY_LOCATION  
154             printf("COLUMN %d\n", y);  
155         #endif */  
156         for (x = 0; x < 8; x++) {  
157             TCE_MUL_16_FIX(fy[0], cos16(x), a);  
158             TCE_MUL_16_FIX(fy[1], cos16(3*x), b);  
159             TCE_MUL_16_FIX(fy[2], cos16(5*x), c);  
160             TCE_MUL_16_FIX(fy[3], cos16(7*x), d);  
161             TCE_MUL_16_FIX(fy[4], cos16(9*x), e);  
162             TCE_MUL_16_FIX(fy[5], cos16(11*x), g);  
163             TCE_MUL_16_FIX(fy[6], cos16(13*x), h);  
164             TCE_MUL_16_FIX(fy[7], cos16(15*x), i);  
165         /*#ifdef DEBUG_MULTIPLIER  
166             printf(" a: %d, f: %d, cos: %dn", a, fy[0], cos16(x));  
167             printf(" b: %d, f: %d, cos: %dn", b, fy[1], cos16(3*x));  
168             printf(" c: %d, f: %d, cos: %dn", c, fy[2], cos16(5*x));  
169             printf(" d: %d, f: %d, cos: %dn", d, fy[3], cos16(7*x));  
170             printf(" e: %d, f: %d, cos: %dn", e, fy[4], cos16(9*x));  
171             printf(" g: %d, f: %d, cos: %dn", g, fy[5], cos16(11*x));  
172             printf(" h: %d, f: %d, cos: %dn", h, fy[6], cos16(13*x));  
173             printf(" i: %d, f: %d, cos: %dn", j, fy[7], cos16(15*x));  
174         #endif  
175     #endif  
176  
177     #ifdef DEBUG_ADDER  
178         temp3 = fix_add_16(a,b);  
179         printf("a+b : %dn", temp3);  
180         temp3 = fix_add_16(temp3,c);  
181         printf(" +c : %dn", temp3);  
182         temp3 = fix_add_16(temp3,d);  
183         printf(" +d : %dn", temp3);  
184         temp3 = fix_add_16(temp3,e);  
185         printf(" +e : %dn", temp3);  
186         temp3 = fix_add_16(temp3,g);  
187         printf(" +g : %dn", temp3);  
188         temp3 = fix_add_16(temp3,h);  
189         printf(" +h : %dn", temp3);  
190     #endif  
191 }
```

کامپایل کردن برنامه:



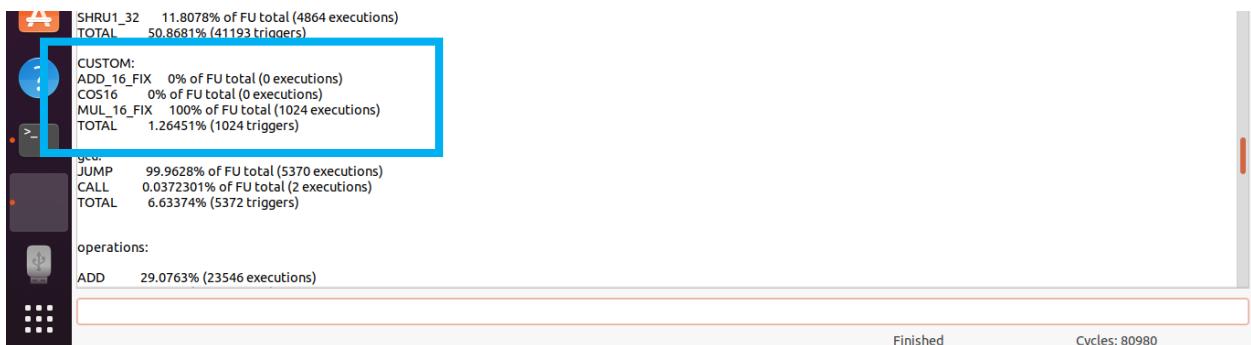
```
root@imangholi:/home/ali/Desktop/CA3-Ali-Imangholi-810197692/report/step-3/question-9/just-modify-mul# tcecc -O2 -a custom.adf -o dct.tpef dct
```

تعداد سیکلی که طول می کشد کد اجرا شود:



```
>run
>info proc cycles
80980
>
```

:custom functional units اطلاعات



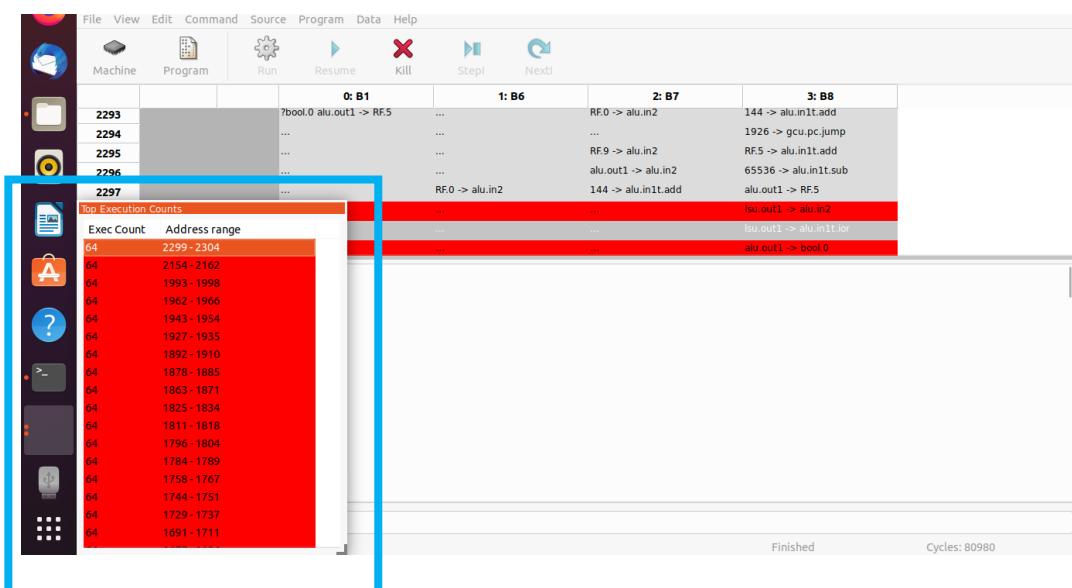
SHRU1\_32 11.8078% of FU total (4864 executions)  
TOTAL 50.8681% (41193 triggers)

CUSTOM:  
ADD\_16\_FIX 0% of FU total (0 executions)  
COS16 0% of FU total (0 executions)  
MUL\_16\_FIX 100% of FU total (1024 executions)  
TOTAL 1.26451% (1024 triggers)

JKR:  
JUMP 99.9628% of FU total (5370 executions)  
CALL 0.0372301% of FU total (2 executions)  
TOTAL 6.63374% (5372 triggers)

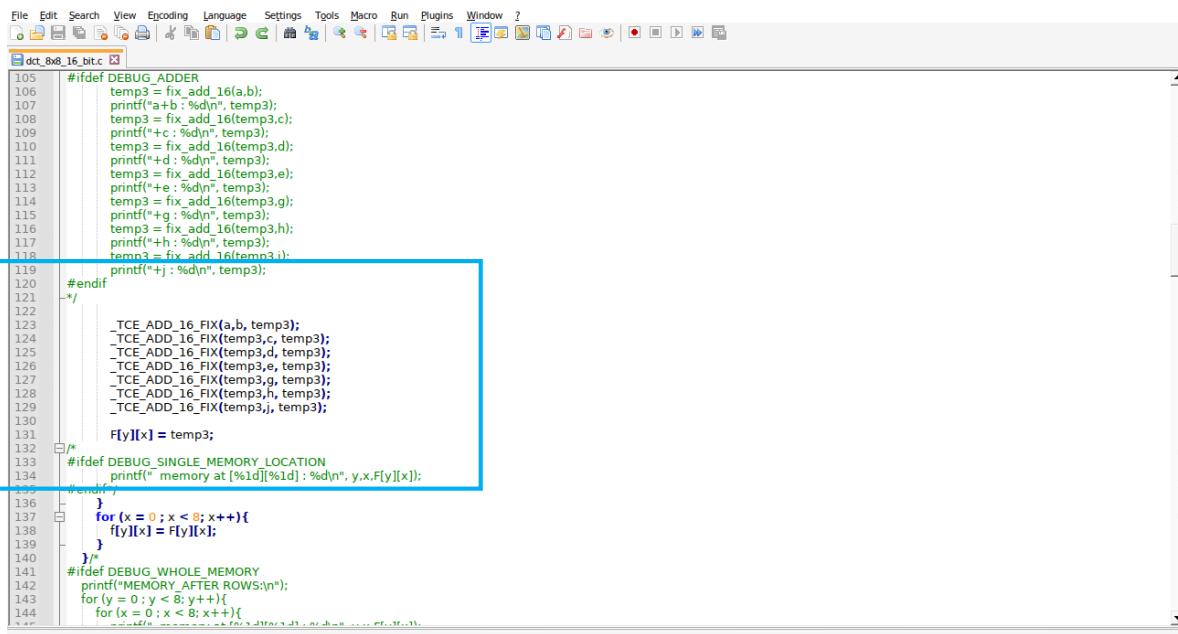
operations:  
ADD 29.0763% (23546 executions)

: Highlight top execution counts

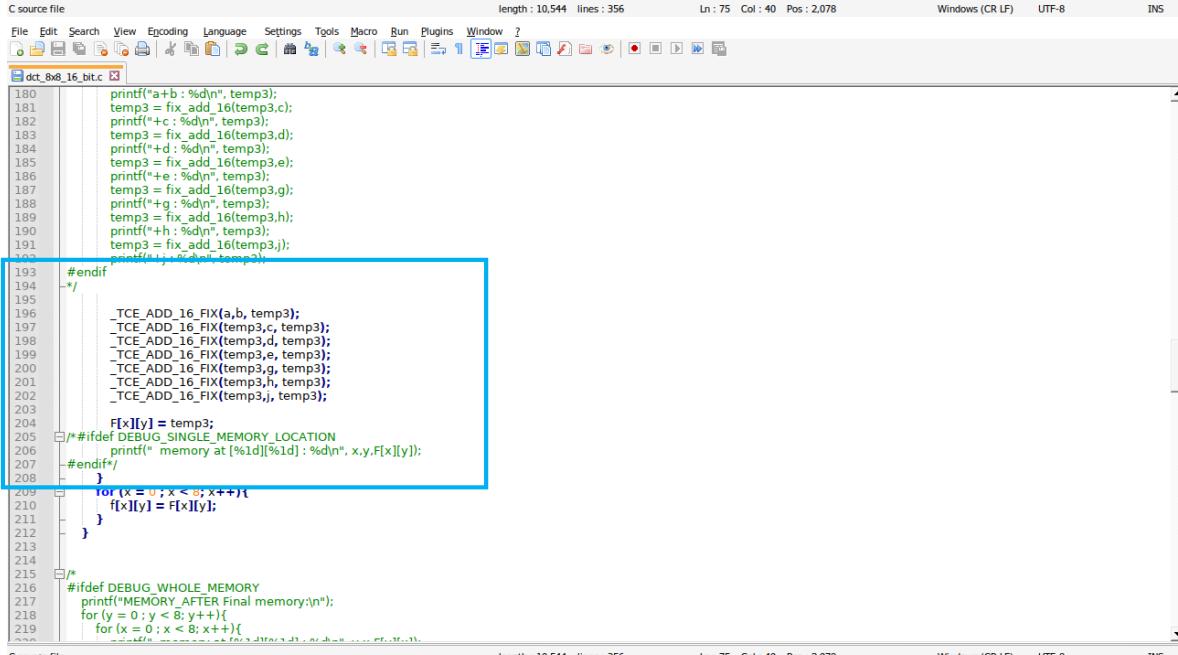


## استفاده از مازول ADD\_16\_FIX ساخت شده:

تغییرات کد به نحوی که از مازول add\_16\_fix ساخته شده استفاده گردد:



```
105 #ifdef DEBUG_ADDER
106     temp3 = fix_add_16(a,b);
107     printf("a+b : %dn",temp3);
108     temp3 = fix_add_16(temp3,c);
109     printf("c : %dn",temp3);
110     temp3 = fix_add_16(temp3,d);
111     printf("d : %dn",temp3);
112     temp3 = fix_add_16(temp3,e);
113     printf("e : %dn",temp3);
114     temp3 = fix_add_16(temp3,g);
115     printf("g : %dn",temp3);
116     temp3 = fix_add_16(temp3,h);
117     printf("h : %dn",temp3);
118     temp3 = fix_add_16(temp3,i);
119     printf("i+j : %dn",temp3);
120
121 #endif
122 */
123 -TCE_ADD_16_FIX(a,b,temp3);
124 -TCE_ADD_16_FIX(temp3,c,temp3);
125 -TCE_ADD_16_FIX(temp3,d,temp3);
126 -TCE_ADD_16_FIX(temp3,e,temp3);
127 -TCE_ADD_16_FIX(temp3,g,temp3);
128 -TCE_ADD_16_FIX(temp3,h,temp3);
129 -TCE_ADD_16_FIX(temp3,i), temp3;
130
131 F[y][x] = temp3;
132 */
133 #ifdef DEBUG_SINGLE_MEMORY_LOCATION
134     printf(" memory at [%ld][%ld] : %dn", y,x,F[y][x]);
135 #endif
136 */
137 for (x = 0 ; x < 8; x++){
138     F[y][x] = F[y][x];
139 }
140 */
141 #ifdef DEBUG_WHOLE_MEMORY
142     printf("MEMORY_AFTER ROWS:\n");
143     for (y = 0 ; y < 8; y++){
144         for (x = 0 ; x < 8; x++){
145             printf("%d ", F[x][y]);
146         }
147     }
148 */
149 #endif
```



```
180     printf("a+b : %dn",temp3);
181     temp3 = fix_add_16(temp3,c);
182     printf("c : %dn",temp3);
183     temp3 = fix_add_16(temp3,d);
184     printf("d : %dn",temp3);
185     temp3 = fix_add_16(temp3,e);
186     printf("e : %dn",temp3);
187     temp3 = fix_add_16(temp3,g);
188     printf("g : %dn",temp3);
189     temp3 = fix_add_16(temp3,h);
190     printf("h : %dn",temp3);
191     temp3 = fix_add_16(temp3,i);
192     printf("i+j : %dn",temp3);
193
194 #endif
195 */
196 -TCE_ADD_16_FIX(a,b,temp3);
197 -TCE_ADD_16_FIX(temp3,c,temp3);
198 -TCE_ADD_16_FIX(temp3,d,temp3);
199 -TCE_ADD_16_FIX(temp3,e,temp3);
200 -TCE_ADD_16_FIX(temp3,g,temp3);
201 -TCE_ADD_16_FIX(temp3,h,temp3);
202 -TCE_ADD_16_FIX(temp3,i), temp3;
203
204 F[x][y] = temp3;
205 */
206 #ifdef DEBUG_SINGLE_MEMORY_LOCATION
207     printf(" memory at [%ld][%ld] : %dn", x,y,F[x][y]);
208 #endif
209 */
210 for (x = 0 ; x < 8; x++){
211     F[x][y] = F[x][y];
212 }
213 */
214 /*
215 #ifdef DEBUG_WHOLE_MEMORY
216     printf("MEMORY_AFTER Final memory:\n");
217     for (y = 0 ; y < 8; y++){
218         for (x = 0 ; x < 8; x++){
219             printf("%d ", F[x][y]);
220         }
221     }
222 */
223 #endif
```

کامپایل کردن برنامه:



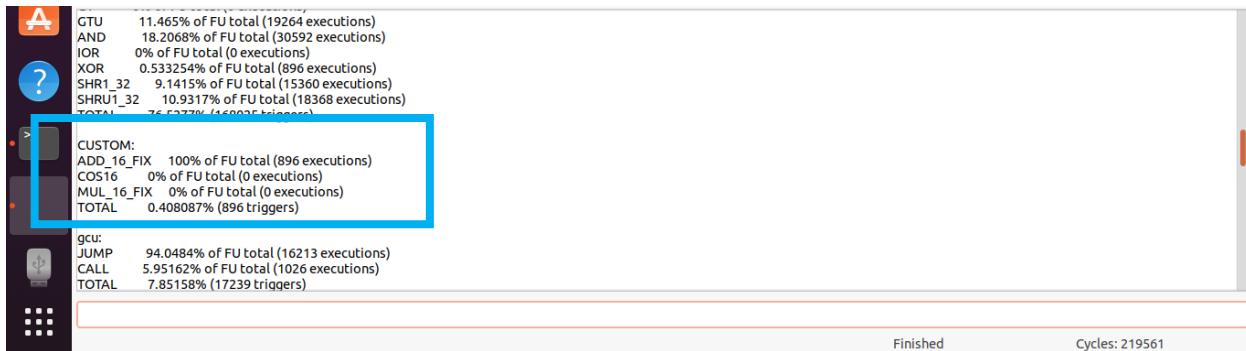
```
Activities Terminal 19:11 26 منی
root@imangholi:/home/ali/Desktop/CA3-Ali-Imangholi-810197692/report/step-3/question-9/just-modify-add# tcc -O2 -a custom.adf -o dct.tpef dct_8x8_16_bit.c
root@imangholi:/home/ali/Desktop/CA3-Ali-Imangholi-810197692/report/step-3/question-9/just-modify-add#
```

تعداد سیکلی که طول می کشد کد اجرا شود:



```
>run  
>info proc cycles  
219561  
>
```

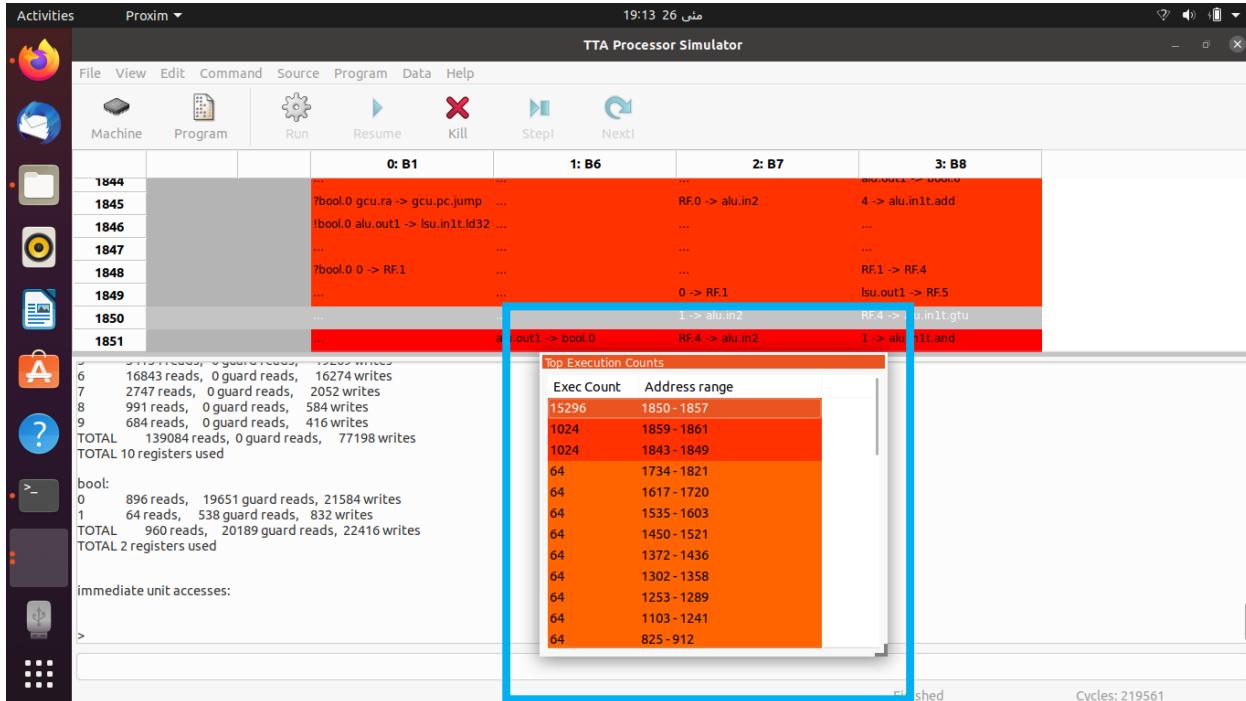
### :custom functional units اطلاعات



GTU	11.465% of FU total (19264 executions)	
AND	18.2068% of FU total (30592 executions)	
IOR	0% of FU total (0 executions)	
XOR	0.533254% of FU total (896 executions)	
SHR1_32	9.1415% of FU total (15360 executions)	
SHRU1_32	10.9317% of FU total (18368 executions)	
TOTAL	75.2000% of FU total (219561 executions)	
CUSTOM:		
ADD_16_FIX	100% of FU total (896 executions)	
COST16	0% of FU total (0 executions)	
MUL_16_FIX	0% of FU total (0 executions)	
TOTAL	0.408087% (896 triggers)	
gcu:		
JUMP	94.0484% of FU total (16213 executions)	
CALL	5.95162% of FU total (1026 executions)	
TOTAL	7.85158% (17239 triggers)	

Finished Cycles: 219561

### : Highlight top execution counts



Exec Count	Address range
15296	1850 - 1857
1024	1859 - 1861
1024	1843 - 1849
64	1734 - 1821
64	1617 - 1720
64	1535 - 1603
64	1450 - 1521
64	1372 - 1436
64	1302 - 1358
64	1253 - 1289
64	1103 - 1241
64	825 - 912

Finished Cycles: 219561

## استفاده از تمامی مازول های ساخت شده:

تغییرات کد به نحوی که از تمامی مازول های ساخته شده استفاده گردد:

The image shows three vertically stacked code editors, all displaying the same C code for a matrix multiplication function. The code is annotated with various compiler directives (#ifdef) to enable different optimization levels or specific memory locations.

**Top Editor:** Shows code for `MEMORY LOCATION`. It includes sections for `DEBUG_SINGLE_MEMORY_LOCATION`, `DEBUG_ADDER`, and `DEBUG_MULTIPLIER`. It uses `printf` statements to output intermediate values like `a[i][j]`, `b[i][j]`, and `c[i][j]`.

**Middle Editor:** Shows code for `MEMORY LOCATION`. It includes sections for `DEBUG_SINGLE_MEMORY_LOCATION`, `DEBUG_ADDER`, and `DEBUG_MULTIPLIER`. It uses `printf` statements to output intermediate values like `a[i][j]`, `b[i][j]`, and `c[i][j]`.

**Bottom Editor:** Shows code for `MEMORY LOCATION`. It includes sections for `DEBUG_SINGLE_MEMORY_LOCATION`, `DEBUG_ADDER`, and `DEBUG_MULTIPLIER`. It uses `printf` statements to output intermediate values like `a[i][j]`, `b[i][j]`, and `c[i][j]`.

The code itself is a standard C implementation of matrix multiplication, utilizing nested loops and various mathematical operations (addition, subtraction, multiplication) on 16-bit integers.

```

dct_8x8_16_bit.c
214     temp3 = fix_add_16(a,b);
215     printf("%a+b : %d\n",temp3);
216     temp3 = fix_add_16(temp3,c);
217     printf("%a+c : %d\n",temp3);
218     temp3 = fix_add_16(temp3,d);
219     printf("%a+d : %d\n",temp3);
220     temp3 = fix_add_16(temp3,e);
221     printf("%a+e : %d\n",temp3);
222     temp3 = fix_add_16(temp3,g);
223     printf("%a+g : %d\n",temp3);
224     temp3 = fix_add_16(temp3,h);
225     printf("%a+h : %d\n",temp3);
226     temp3 = fix_add_16(temp3,i);
227     printf("%a+i : %d\n",temp3);
228     #endif
229 }

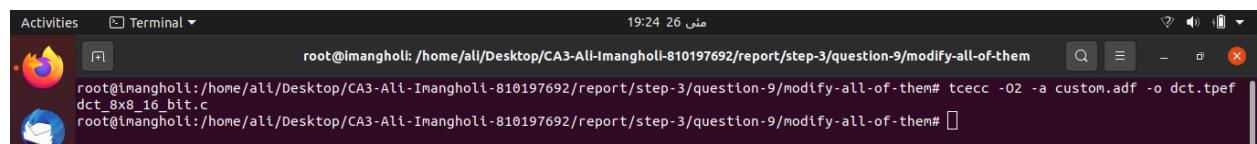
31     TCE_ADD_16_FIX(a,b, temp3);
32     TCE_ADD_16_FIX(temp3,c, temp3);
33     TCE_ADD_16_FIX(temp3,d, temp3);
34     TCE_ADD_16_FIX(temp3,e, temp3);
35     TCE_ADD_16_FIX(temp3,g, temp3);
36     TCE_ADD_16_FIX(temp3,h, temp3);
37     TCE_ADD_16_FIX(temp3,i, temp3);
38
39 F[x][y] = temp3;
40
41     #endif
42     printf(" memory [%d][%d] : %d\n", x,y,F[x][y]);
43 }
44
45 for(x = 0 ; x < 8 ; x++){
46     F[x][y] = F[x][y];
47 }
48
49
50 #if DEBUG_WHOLE_MEMORY
51 printf(MEMORY_AFTER|memory)\n";
52 for(y = 0 ; y < 8 ; y++){
53     for(x = 0 ; x < 8 ; x++)
54         F[x][y] = 0;
55 }
56
57 #endif

```

source file

length: 11,507 lines: 391 Ln: 77 Col: 40 Pos: 2,160 Windows (CR/LF) UTF-8 INS

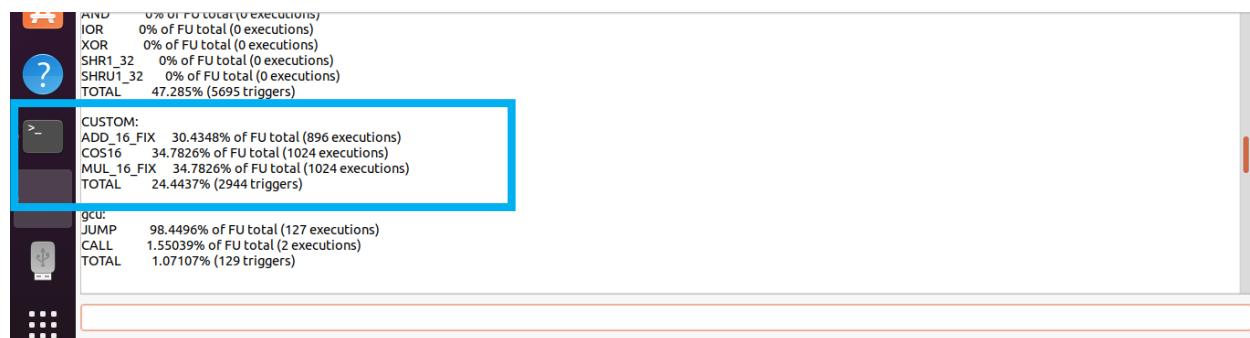
کامپایل کردن برنامه:



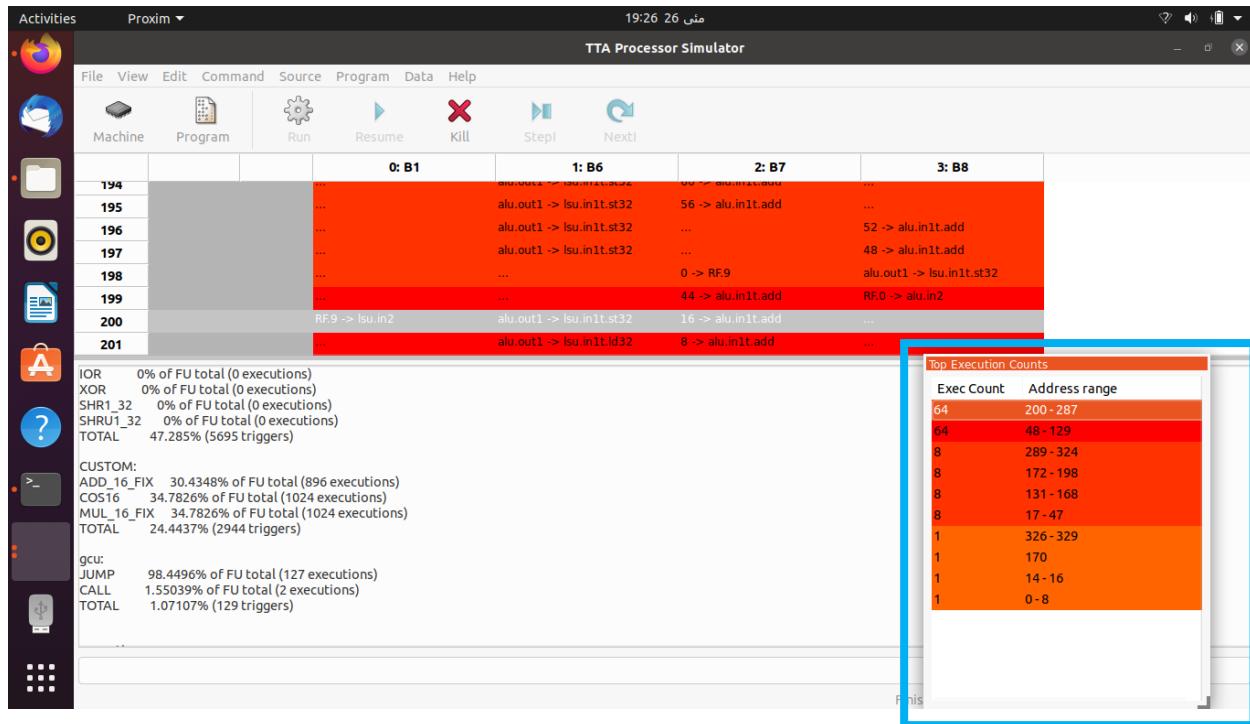
تعداد سیکلی که طول می کشد کد اجرا شود:



.اطلاعات custom functional units



### : Highlight top execution counts



### مقایسه میان تمامی حالت ها:

	Clock cycle	Top Execution counts
functional unit بودن اضافه کردن هیچ	257288	15296
اضافه کردن COS_16	222443	14416
اضافه کردن ADD_16_FIX	219561	15296
اضافه کردن MUL_16_FIX	80980	64
اضافه کردن تمامی functional unit های MUL ، COS و ADD	12044	64

همانطور که از جدول بالا مشخص است، با افزودن functional unit ها مقدار Top Execution counts و clock cycle کاهش می یابد و درنتیجه performance سیستم افزایش می یابد.

اوج این مورد هنگامی رخ می دهد که برای ضرب کننده یک functional unit مجزا قرار دمی دهیم و همانطور که در قسمت های قبل نیز بررسی شد دیدیم که در کد DCT عملیات ضرب زیادی انجام می شود و این امر منطقی است که اضافه کردن یک functional unit مجزا برای اجرای کد باعث کاهش Top Execution counts و clock cycle شود.

پس از ضرب کننده، جمع کننده مورد بعدی ای است که باعث کاهش Top Execution counts و clock cycle می شود زیرا در قسمت های از کد عملیات جمع نیز وجود دارد و این عملیات جمع را می توان با ای که برای آن اضافه می کنیم، انجام دهیم. در نهایت ماژول cos می باشد که باعث کاهش Top Execution counts و clock cycle و نسبت به 2 مورد دیگر تاثیر کمتری می گذارد (ولیکن با این حال تاثیر چشمگیری دارد). و علت آن این است که عملیات کسینوس گیری کمتر نسبت به صرب و جمع مورد استفاده قرار گرفته است.

در انتهای تاثیر اضافه کردن هر 3 مورد را مشاهد می کنیم که به شدت باعث کاهش Top Execution counts و clock cycle و افزایش performance نهایی شده است.

سیستم نهایی با تغییراتی که در Register file و bus و افزودن functional unit داشته است مناسب کاربرد هایی شده است که از عملیات DCT بهره می برند، همانند فشرده سازی و ... .

با تشکر از توجه شما