بسمه تعالی

درس سیستم‌های نهفته مبتنی بر هسته

تکلیف کامپیوتری ۳ : یادگیری استفاده از ابزار TCE برای پیاده سازی روی پردازنده TTA

پردیس دانشکده های فنی دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

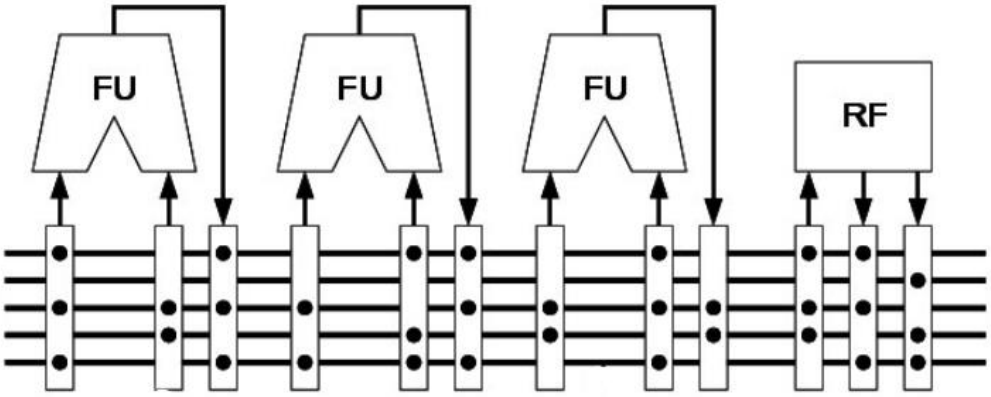
دکتر احمد شعبانی

نیم‌سال دوم سال تحصیلی ۱۴۰۲-۱۴۰۱

نگارش: امیرمهدی جودی - نگین سفاری

مقدمه:

هدف از این تمرین کامپیوتری، پیاده‌سازی یک نمونه پردازنده ASIP مبتنی بر ساختار TTA برای یک کاربرد خاص است.



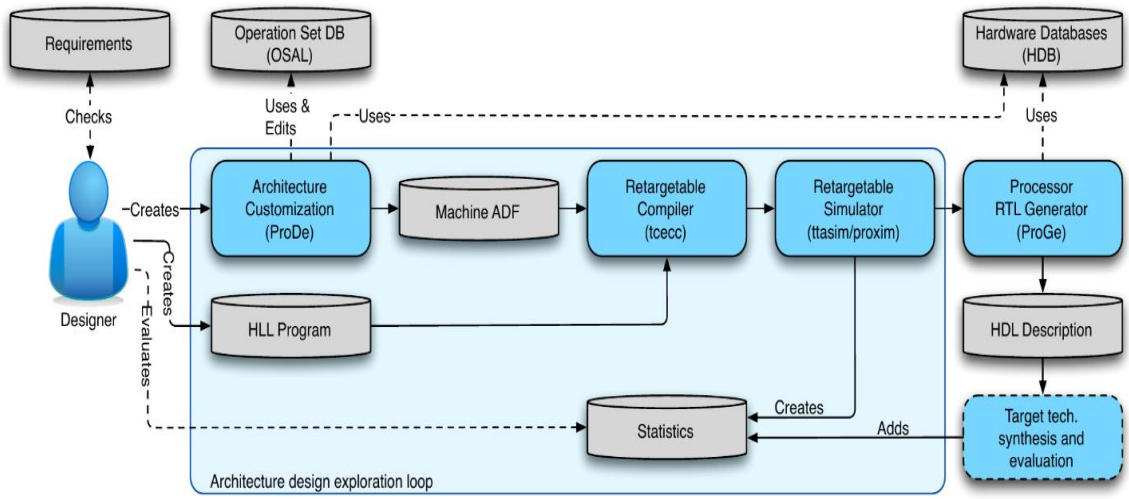
شکل ۱: ساختار کلی یک پردازنده ASIP

کاربرد خاص مورد نظر در این تمرین، پیاده‌سازی الگوریتم تبدیل گسسته کسینوسی 8×8 DCT است. تبدیل کسینوسی گسسته دنباله‌ای از اعداد را به صورت مجموع توابع کسینوسی با فرکانس‌های متفاوت نمایش میدهد. این تبدیل در پردازش تصویر و ویدئو و فشرده‌سازی داده کاربردهای فراوانی دارد. از آنجا که پیاده‌سازی سخت افزاری مستقیم این تبدیل به دلیل بالا بودن تعداد ضرب کننده‌های ممیز شناور، پیچیده و مستلزم زمان

اجرای زیاد است، بررسی روش‌های بهینه‌سازی و کاهش زمان اجرای این تبدیل برای یک کاربرد خاص منظوره مورد توجه است. در حالت کلی، میتوان تبدیل گسسته کسینوسی تک بعدی را برای یک بلوک با ابعاد N*N به شکل زیر بیان نمود که در این رابطه x_n نمونه‌های داده ورودی و X_K خروجی ضرایب تبدیل گسسته کسینوسی میباشد.

$$X_k = \sum_{n=0}^{N-1} x_n cos\left[\frac{\pi}{N}\left(n+\frac{1}{2}\right)k\right] \qquad for \;\; k = 0, 1, 2, \dots, N-1$$

ابزار TCE که در این تمرین با آن آشنا میشویم به ما کمک میکند که الگوریتم مورد نظر را روی پردازنده‌ی همه منظوره TTA پیاده‌سازی کنیم و در جهت بهبود پارامترهای طراحی و شخصی‌سازی پردازنده، تغییراتی را برای یک کاربرد خاص در این پردازنده انجام دهیم و نتایج مربوطه را استخراج نماییم. این ابزار علاوه بر فراهم ساختن امکاناتی برای مشاهده نحوه اجراء بر روی پردازنده مذکور، امکان اعمال تغییرات در سطح معماری را فراهم می‌سازد. ضمن اینکه، امکان پروفایل‌گیری به‌منظور مشخص کردن نقاط هات اسپات در الگوریتم و اضافه کردن یکسری شتاب‌دهنده سخت افزاری وجود دارد .



شکل ۲: مسیر طراحی در ابزار TCE

گام اول:

این ابزار روی سیستم عامل linux نصب می‌شود. مراحلی که در ادامه گفته می‌شوند برای نسخه‌های Ubuntu 20.04 یا بالاتر مناسب هستند. پیشنهاد می‌شود از این نسخه استفاده کنید اما در صورتی که قصد استفاده از نسخه‌های دیگر را دارید، می‌توانید از اینجا مراحل را مطالعه کنید.

به کمک دستورات زیر، پیش نیازهای مورد نیاز را نصب کنید.

```
sudo apt-get install libwxgtk3.0-gtk3-dev libboost-all-dev \
tcl8.6-dev libedit-dev libsqlite3-dev sqlite3 libxerces-c-dev g++ make \
latex2html libffi-dev autoconf automake libtool subversion git cmake
```

سپس openASIP که نسخه جدید TCE هست را به کمک دستور زیر clone کنید (openASIP که قبلا فقط از معماری TTA پشتیبانی می‌کرد، با نام TCE شناخته می‌شد، اما حالا از ISA های بیشتری پشتیبانی می‌کند):

```
git clone https://github.com/cpc/openasip.git openasip-devel
```

پس از آن به مسیر زیر رفته و با استفاده از کد زیر نسخه‌ی مناسب LLVM را نصب کنید.

```
cd openasip-devel/openasip
tools/scripts/install_llvm_15.sh $HOME/local
```

این بخش ممکن است زمان زیادی طول بکشد! پس از اتمام نصب LLVM در همان ترمینال دستور زیر را وارد کنید.

```
nano ~/.bashrc
```

فایلی باز می‌شود. به انتهای آن رفته و سه خط زیر را کپی کرده و سیو کنید. پس از آن ترمینال را بسته و دوباره در همان پوشه‌ی tce ترمینال را باز کنید.

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$HOME/local/lib
export PATH=$HOME/local/bin:$PATH
export LDFLAGS=-L$HOME/local/lib
```

اگر این سه خط را در فایل bashrc ذخیره نکنید با هر اجرای ترمینال باید آن‌ها را وارد کنید. حال که دوباره وارد ترمینال شدید (در همان مسیر قبل)، کد زیر را وارد کنید و منتظر اتمام عملیات نصب باشید.

```
./autogen.sh && ./configure --prefix=$HOME/local && make -j8 && make install
```

در صورتی که برای نصب پیش‌نیاز دیگری لازم بود آن را اضافه کرده و خط قبل را دوباره اجرا کنید. به عنوان مثال در صورتی که با خطای نداشتن ورژن مناسب پایتون مواجه شدید خط زیر را اجرا کرده و دوباره مرحله‌ای که دچار خطا شده بود را اجرا کنید.

```
sudo apt-get install python
```

در این مرحله باید نصب به صورت کامل انجام شده باشد. برای تست آن خط کد زیر را در ترمینال بنویسید.

ttasim --version

گام دوم: حالا از پوشه‌ی openasip وارد پوشه‌ی data و سپس mach شوید و فایل minimal.adf را کپی کنید. آن را در پوشهی DCT که از پیوست ذخیره کرده‌اید پیست کنید و در همان پوشه ترمینال جدیدی باز کنید. با اجرای خط کد زیر میتوانید سیستم اولیه ی مینیمالی که در اختیار داریم را مشاهده کنید.

prode minimal.adf

حال میخواهیم الگوریتم موجود در کدهایی که به صورت آماده در فایل dct_8x8_16_bit.c در اختیار شما قرار داده شده است را بر روی این سیستم مینیمال پیاده کنیم.

tcecc -O2 -a minimal.adf -o dct.tpef dct_8x8_16_bit.c
proxim minimal.adf dct.tpef

۲- دو خط بالا را اجرا کرده و در پنجره ی باز شده run را بزنید و در پایین صفحه دستور info proc cycles و سپس info proc stats را وارد کنید و تعداد سیکلی که طول میکشد تا کد اجرا شود و خلاصه ای از آمار داده شده را در گزارش خود بیاورید. از منوی source میتوانید profile data و سپس highlight top execution counts را انتخاب کنید تا بیشترین تعداد انجام شدن دستورات را مشاهده کنید. در تمام قسمتها نتایج را در گزارش آورده و فایلها را با نام مناسب ضمیمه کنید.

حال دوباره دستور prode minimal.adf را اجرا کنید و روی واحد RF:RF دبل کلیک کرده و مقدار size را از ۵ به ۸ تغییر دهید و در قسمت ports گزینه ی add را انتخاب کرده و دو پورت جدید اضافه کنید و save as را انتخاب کرده و با نام added_registers.adf ذخیره کنید.

۳- کدهای زیر را در ترمینال وارد کرده و مراحل سوال ۲ را برای سیستم جدید پیاده کرده و نتایج را مقایسه کنید.

tcecc -O2 -a added_registers.adf -o dct.tpef
dct_8x8_16_bit.c proxim added_registers.adf dct.tpef

سیستم مینیمالی که در اختیار داریم فقط یک باس برای اطلاعات دارد. prode added_registers.adf را اجرا کرده و از منوی edit گزینهی add و سپس transport bus انتخاب کنید. این کار را تکرار کرده تا زمانی که ۸ باس داشته باشیم. میتوانید هر سوکت را به هر باس متصل یا جدا کنید اما در این تمرین کافی است از منوی tools، گزینه ی fully connected IC را انتخاب کنید.

۴- سیستم جدید را با اسم multiple_bus.adf ذخیره کرده و مراحل قبل را تکرار و نتایج را مقایسه کنید .

۵- با توجه به نتایج و این که از هر باس چند درصد مواقع استفاده شده بگویید آیا به ۸ باس احتیاج داشتیم یا با تعداد باس کمتر نیز به نتایج نسبتا مشابه میرسیدیم؟ با ذکر دلیل توضیح دهید چند باس برای بهبود کار ما کافی بود؟ تعداد باس را به مقدار پیشنهادی تغییر دهید و تعداد سیکل را مقایسه کنید.

۶- برای این که مقایسه کنیم کدام دستورات پرتکرارترین هستند ابتدا دستوری که در ترمینال وارد میکنیم را به شکل زیر تغییر میدهیم و سپس highlight top execution counts را انتخاب میکنیم.

```
tcecc -O2 -a multiple_bus.adf -o dct.tpef dct_8x8_16_bit.c --disable-inlining proxim
multiple_bus.adf dct.tpef
```

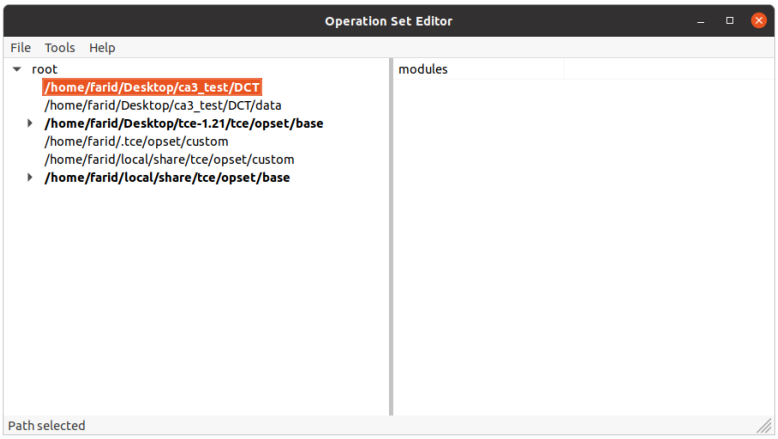
گام سوم: حال میخواهیم واحد محاسباتی جدیدی معرفی کنیم که به صورت سخت افزار اختصاصی بخشهای پرتکرار الگوریتم را انجام دهد. در کد dct داده شده سه تابع مشخص شده اند که ضرب و جمع و کسینوس را محاسبه میکنند.

جمع و ضرب به شکلی پیاده شده اند که با وجود این که نوع داده ها بدون علامت است محاسبات علامتدار باشند. عملیات ضرب، خروجی را ۱۵ بار به راست شیفت میدهد که دلیل آن مرتبط با این است که هرجا عملیات ضرب داریم یکی از اپرندهای آن کسینوس است، بنابراین؛ میتوان به جای کسینوس ۲۱۵ برابر آن را محاسبه کرد و بعد از ضرب تقسیم را انجام داد. بدین صورت مقادیر کسینوس به اندازه کافی بزرگ میشوند تا بتوانیم از نوع داده ی بدون علامت استفاده کنیم.

با توجه به رابطه‌ی dct میتوان مشاهده کرد که به تمام مقادیر کسینوس نیازی نداریم و فقط ضرایب فرد $\frac{\pi}{16}$ مورد نیاز هستند. پس کد را میتوان به گونه ای نوشت که خروجی کسینوس برای این مقادیر را به صورت بدون علامت بدهد.

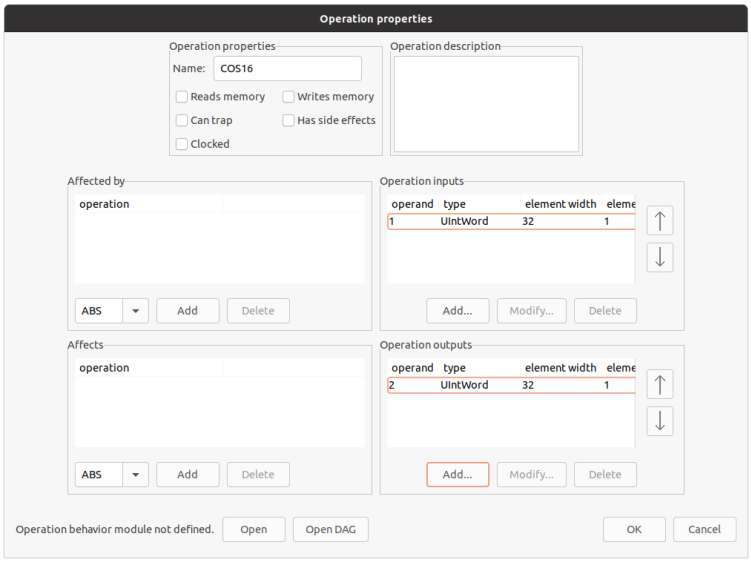
۷- با مطالعه‌ی کد dct_8x8_16_bit.c و سه تابع معرفی شده در انتهای آن توضیح دهید این سه تابع به چه شکل کار میکنند. (کافی است توضیح دهید عملکرد کلی این سه تابع با توجه به ورودیهای مختلفی که میگیرند به چه صورت است)

برای اضافه کردن توابع مورد نظر در ترمینال osed را وارد کنید. در صفحه ی باز شده root را باز کرده و مانند شکل زیر اولین خط آدرس محلی است که در آن قرار دارید. روی آن کلیک راست کنید و یک ماژول جدید به نام custom درست کنید.



شکل ۳: انتخاب مسیر مناسب در osed

سپس روی نام ماژول کلیک کرده و add operation را انتخاب کنید. نام عملگر را COS16 بگذارید و مانند شکل زیر یک ورودی و یک خروجی از نوع Uint word و با اندازه ۳۲ بیت برای آن اضافه کنید.



شکل ٤: اضافه کردن تابع جدید در osed

دو عملگر دیگر با نام های MUL_16_FIX و ADD_16_FIX اضافه کنید که هر کدام دو ورودی و یک خروجی مانند قبل داشته باشند. روی یکی از این عملگرها کلیک راست کرده و modify behavior را انتخاب کنید و محتویات فایل custom.c را در فایل custom.cc که باز شده کپی کنید.

۸- روی ماژول custom کلیک راست کرده و build را بزنید و سپس روی هریک از عملگرها کلیک راست کنید و simulate بزنید و درستی آنها را بررسی کنید. برای این کار باید توجه کنید هر عملگر چه ورودی هایی میتواند داشته باشد و خروجی آن به چه صورت تعبیر میشود.

بعد از اطمینان از درستی عملگرها osed را ببندید و با دستور prode آخرین سیستمی که ساختید را باز کنید. از منوی edit گزینهی add و سپس function unit را انتخاب کنید. نام آن را CUSTOM گذاشته و در قسمت سمت راست (پورت ها) گزینهی add را انتخاب کنید و گزینه ی trigger را انتخاب کنید. دو پورت دیگر نیز اضافه کنید بدون این که تریگر آنها را فعال کنید.

در قسمت operations گزینه ی add from Opset را انتخاب کنید و از لیست ADD_16_FIX را با latency 3 انتخاب کنید. فرض میکنیم پس از ۳ سیکل خروجی جمع کننده آماده میشود که در واقعیت احتمالا مقدار کمتری خواهد بود. همین کار را برای MUL_16_FIX و 16COS نیز تکرار میکنیم. کسینوس تنها یک ورودی دارد بنابراین از دو پورت استفاده میکند. هنگام اضافه کردن این عملگر اپرند ۲ را به پورت ۳ وصل میکنیم تا همه ی خروجی ها به پورت یکسان وصل باشند. سیستم جدید را با نام custom.adf ذخیره کنید.

حال سراغ کد dct_8x8_16_bit.c می‌رویم. باید توابع قبلی با توابع ساخته شده جایگزین شوند. در این مرحله به شکل زیر کدها را جایگزین کنید. در حالت اولیه داریم:

```
a = cos16(b)
a = fix_add_16(b,c)
a = fix_mul_16(b,c)
```

در حالت جدید این توابع به شکل زیر در میآیند. توجه کنید خروجی‌ها آخر می‌آیند.

```
_TCE_COS16(b,a)
_TCE_ADD_16_FIX(b,c,a)
_TCE_MUL_16_FIX(b,c,a)
```

با توجه به این که نوع آرگومان گرفتن توابع متفاوت است و خروجی در آرگومانها می‌آید باید به دقت کد را اصلاح کنید و در صورت نیاز متغیرهای میانی جدید تعریف کنید.

۹- پس از ساختن واحد محاسباتی جدید و اضافه کردن آن کد را اصلاح کنید. یک بار فقط cos16 و بار دیگر فقط ضرب کننده و یک بار فقط جمع کننده را جایگزین کنید. یک بار نیز هرسه را با شتاب دهنده سخت افزاری ساخته شده جایگزین کنید. برای هر حالت مراحل قبل را تکرار کنید و نتایج را مقایسه کنید و در گزارش بیاورید. کدام تابع تاثیر بیشتری در تعداد سیکل داشت؟

تاریخ تحویل: ساعت 23:59 روز 1402/3/6

موفق و سلامت باشید.