یک متدولوژی مجتمع‌سازی هسته‌های IP در یک محیط طراحی شیءگرای سیستم‌های نهفته

شعله هاشمی نمین*، شاهین حسابی†

چکیده

امروزه استفاده‌ی مجدد از هسته‌های IP در طراحی و پیاده‌سازی سیستم های پیچیده کاربرد فراوانی دارد؛ زیرا استفاده‌ی مجدد از این بلوک های آماده، هم زمان طراحی را کاهش می دهد و هم سیستم پیاده سازی شده با استفاده از هسته های IP از نظر سرعت، اندازه و کارایی بهینه می باشد. در این مقاله یک متدولوژی استفاده از هسته‌های IP برای پیاده‌سازی متدهای سخت‌افزاری در چارچوب طراحی ادیسه که یک محیط طراحی سیستم های نهفته است، ارایه شده است. مجتمع‌سازی هسته‌های IP در این متدولوژی، خودکار شده است و این امر بسیاری از مشکلات استفاده از هسته‌های IP را برطرف می کند. برای ارزیابی متدولوژی پیشنهادی یک تابع IDCT دو بعدی (2-D Inverse Discrete Cosine Transform) در محیط طراحی ادیسه یک بار با استفاده از هسته‌ی IP و بار دیگر بدون استفاده از آن، طراحی و پیاده سازی شده و با یکدیگر مقایسه شده‌اند.

کلمات کلیدی

ادیسه، هسته‌ی IP، wrapper، ASIP

An IP Integration Methodology in an Object-Oriented Design Environment for Embedded Systems

Shoaleh Hashemi Namin, Shaahin Hessabi
Department of Computer Engineering Sharif University of Technology

Abstract

Nowadays, IP core reuse is popular for designing and implementing complex systems; because reuse of already provided blocks not only decreases design time but also systems implemented with these blocks are optimized in terms of speed, size, and performance. In this paper we propose a methodology for using IP cores in ODYSSEY design framework, an embedded system design environment, to implement hardware methods. In this methodology, we have automated IP integration process that eliminates many difficulties related to IP integration. Methodology evaluation has been carried out through the design and implementation of 2-D Inverse Discrete Cosine Transform function in ODYSSEY design environment with and without using IP core.

Keywords

ODYSSEY, IP Core, wrapper, ASIP

* دانشگاه صنعتی شریف، دانشکده مهندسی کامپیوتر، namin@ce.sharif.edu
† دانشگاه صنعتی شریف، دانشکده مهندسی کامپیوتر، hessabi@sharif.edu

۱- مقدمه

امروزه برای طراحی و پیاده سازی سیستم های پیچیده‌ی متشکل از سخت‌افزار و نرم‌افزار متدولوژی‌ها و ابزارهای طراحی موجود این امکان را به طراح می دهند که در مراحل اولیه‌ی طراحی معماری های مختلفی را برای پیاده‌سازی سیستم ارزیابی کند و در نهایت بهترین معماری را برای پیاده‌سازی انتخاب کند. در عین حال امکان استفاده از هسته‌هایIP برای پیاده‌سازی معماری انتخاب شده، قابلیت تولید[1]، قابلیت اطمینان و کارایی طرح را افزایش می‌دهد.

در این مقاله هدف ارایه‌ی یک متدولوژی برای استفاده از هسته‌هایIP به منظور پیاده‌سازی متدهای سخت‌افزاری در معماری ASIP ارایه شده توسط متدولوژی طراحی ادیسه می باشد. همچنین برای برطرف کردن مشکلات مربوط به استفاده از هسته‌هایIP متدولوژی ارایه شده خودکار شده است بدین معنی که در محیط طراحی ادیسه طراح فقط از هسته‌هایIP در طرح خود استفاده می‌کند و مسایل مربوط به مجتمع سازی آنها (طراحی wrapper و استفاده از آنها) خودکار صورت می‌گیرد.

در زمینه‌ی استفاده از هسته‌هایIP در محیط‌های طراحی سیستم یکی از کارهای انجام شده، مربوط به محیط هم‌طراحی و شبیه‌سازی توأمان POLIS [3][2] می باشد. در این محیط یک متدولوژی برای استفاده از کتابخانه‌ی هسته هایIP نرم که به فرمت VHDL در سطح انتقال ثبات[2] هستند، معرفی شده است. این متدولوژی فقط در یک سطحِ شبیه سازی، هسته‌هایIP را در سیستم قرار می‌دهد؛ همچنین مجتمع‌سازی هسته‌هایIP به صورت دستی برای یک کاربرد خاص صورت گرفته است. اما در متدولوژی ما اولاً هسته‌هایIP در بالاترین و پایین ترین سطوح شبیه‌سازی در سیستم قرار داده می‌شوند و همچنین مجتمع‌سازی این هسته‌ها در سیستم کاملاً خودکار می‌باشد.

یکی از کارهای انجام شده در زمینه ی طراحی در سطح سیستمِ سخت‌افزار که در [4] اشاره شده است، طراحی یک کامپایلر می باشد که کد MATLAB یک برنامه ی کاربردی را به عنوان ورودی دریافت می‌کند و یک کد VHDL در سطح انتقال ثبات ارایه می کند؛ برای بهبودِ طرحی که به وسیله‌ی این کامپایلر تولید می شود یکی از کارهای انجام شده استفاده از هسته های IP بهینه برای ایجاد طرح موردنظر می باشد. چگونگی استفاده از هسته‌یIP در این کامپایلر بدین صورت است که یک پایگاه داده از هسته‌هایIP شامل پیاده‌سازی HDL و واسطهای آنها وجود دارد و یک تولید کننده‌ی واسط نیز برای هر هسته‌یIP، طراحی و پیاده‌سازی شده است که پارامترهای هسته‌یIP و واسطهای آنها را تعیین می‌کند. تفاوت این کار با متدولوژی ما این است که اولاً در محیط طراحی ادیسه یک سیستم شامل سخت‌افزار و نرم‌افزار تولید می‌شود در حالیکه در [4] سیستم تولید شده کاملاً سخت‌افزاری است و ثانیاً برنامه‌ی کاربردیِ ورودیِ این کامپایلر به زبان MATLAB می‌باشد و بنابراین هسته‌هایIP موجود در این سیستم در سطح بالا توابع MATLAB می‌باشند در حالیکه در محیط طراحی ادیسه زبانِ برنامه‌ی کاربردی ورودی C++ می‌باشد، بنابراین مدل سطح بالای هسته‌هایIP موجود در سیستم ما نیز C++ می‌باشد و به طور کلی متدولوژی پیشنهادی ما با روش به کارگرفته شده در کامپایلر ذکر شده متفاوت می باشد.

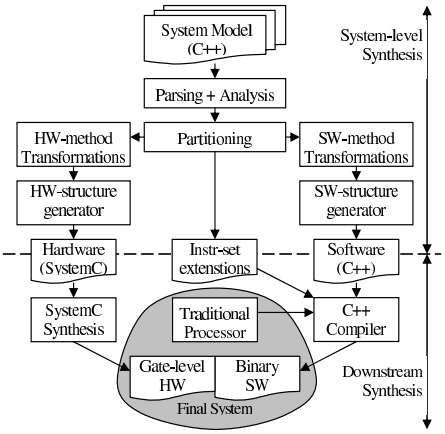
در پروژه ی BALBOA [5] محیطی برای مدل کردن سیستم های نهفته در سطح بالا ارایه شده است که در آن می توان با کنارهم قرار دادن عناصرIP موجود در کتابخانه، یک سیستم نهفته را مدل کرد. عناصرIP موجود در کتابخانه توسط طراحِ کتابخانه به زبان سیستم سی پیاده سازی می شوند. سپس طراحِ سیستم با استفاده از زبان CIL(Component Integration Language) که در این پروژه تعریف شده است، عناصر موجود در کتابخانه را به یکدیگر متصل می‌کند تا سیستم نهایی خود را بسازد. رویکرد این پروژه برای مدل کردن سیستم نهفته از پایین به بالا می‌باشد در حالیکه رویکرد ادیسه در طراحی از بالا به پایین می‌باشد. همچنین پروژه‌ی BALBOA فقط یک محیط برای مدل کردن سیستم‌های نهفته در سطح بالای طراحی ارایه کرده است و در مورد سنتز سیستم طراحی شده و پیاده‌سازی آن هیچ تلاشی صورت نگرفته است در حالیکه در متدولوژی ادیسه بستر پیاده‌سازی سیستم، طراحی و ایجاد شده و سیستم نهایتاً روی یک بُرد FPGA سنتز می شود.

در ادامه‌ی این مقاله، در بخش۲ متدولوژی و محیط طراحی سیستم‌های نهفته‌ی ادیسه و معماری پیشنهادی آن برای پیاده‌سازی سیستمِ طراحی شده معرفی می‌شود. در بخش۳ متدولوژی پیشنهادی برای استفاده از هسته‌هایIP به منظور پیاده‌سازی متدهای سخت‌افزاری در محیط طراحی ادیسه معرفی می شود. در بخش۴ پیاده‌سازیِ تابع IDCT دوبعدی را در محیط ادیسه با بکارگیری هسته‌یIP توضیح می‌دهیم و در بخش۵ نتایج به دست آمده را بررسی می‌کنیم.

۲- متدولوژی ادیسه

ODYSSEY (Object-oriented Design and sYntheSiS of Embedded sYstems) یک محیط طراحی، شبیه‌سازی و سنتز سیستم‌های نهفته می‌باشد[1]. متدولوژی ادیسه به دوبخش تقسیم می‌گردد: یک متدولوژی مدل‌سازی و یک متدولوژی پیاده‌سازی. در زمینه‌ی مدل‌سازی ODYSSEY شیءگرایی را پیشنهاد و دنبال می‌کند و برای پیاده‌سازی طرفدار ایده‌یASIP (application-specific instruction processors) است. دلایل ارایه شده برای این پیشنهاد عبارتند از هزینه‌ی بسیار بالاتر طراحی نرم‌افزار (در مقایسه با سخت‌افزار) در سیستمهای نهفته و مقبولیت شیءگرایی بعنوان یک متدولوژی قوی و موفق در دنیای نرم‌افزار . انگیزه های استفاده از ASIP در پیاده‌سازی عبارتند از هزینه و خطر بالای طراحی و تولید ASIC در فنآوریهای بسیار کوچکتر از میکرون امروزی[1].

مطابق شکل (۱) در محیط ادیسه کل فرآیند سنتز یک ASIP به دولایه تقسیم می‌گردد: لایه‌ی بالا که سنتز در سطح سیستم نامیده می‌شود، مدل سیستم را می‌گیرد و معماری سخت‌افزار و نرم‌افزار را به‌روش‌های رفتاری و ساختاری تولید می‌نماید، و لایه‌ی پایین که لایه‌ی سنتز پایین‌دستی نامیده می‌شود. این لایه بخشهای سخت‌افزاری و نرم‌افزاری تولید شده توسط لایه‌ی قبلی را گرفته و درنهایت یک توصیف در سطح گیت برای سخت‌افزار، و یک برنامه‌ی قابل اجرا برای نرم‌افزار تولید می‌نماید[1].



شکل(۱): روال سنتز تک پردازنده در ODYSSEY [1]

همانطور که در شکل (۱) مشاهده می‌شود کاربرد موردنظر به صورت شیء گرا و با زبان C++ مدل می‌شود و به عنوان ورودی به ابزار سنتز ادیسه داده می‌شود. پس از بخش‌بندی[۳] سیستم به بخش های سخت-افزاری و نرم‌افزاری توسط طراح، ابزار سنتز ادیسه این بخش‌ها را که هر بخش یک متد کلاس در مدل شیءگرا می‌باشد به بلوک‌های سخت‌افزاری و بخش‌های نرم‌افزاری بر روی معماریASIP نگاشت می‌کند بنابراین ASIP ایجادشده متناظر با کلاس‌های مدل شیءگرا می‌باشد. در این مرحله یک مدل شبیه سازی توأمان سیستم‌سی نیز ایجاد می شود. پس از این مرحله یک مدل شبیه‌سازی در سطح انتقال ثبات به فرمت VHDL نیز تولید می‌شود و در نهایت کل سیستم بر روی یک بُرد FPGA که شامل یک هسته‌ی پردازنده نیز می‌باشد با معماریASIP پیشنهادی ادیسه پیاده‌سازی می‌شود.

۲-۱- معماری ASIP

معماری داخلی ASIP در شکل (۲) نشان داده شده است. این معماری، متناظر با یک کتابخانه شامل دو کلاس A و B می باشد که کلاس B از کلاس A مشتق شده است و متدهای f() و g() کلاس A را دوباره تعریف کرده و یک متدh() نیز علاوه بر متدهای موجود تعریف کرده است. شبه کدی که در شکل (۳) آمده است آنچه گفته شد را نشان می‌دهد. توجه کنید که دو تعریف یک متد می توانند روی بخش‌های متفاوتی نگاشت شوند (به عنوان مثال A::g() یک متدِ نرم-افزاری است در حالیکه B::g() یک متد سخت‌افزاری است)[6].

داده‌های تمام شیءها در یک حافظه‌ی داده‌ی مرکزی ذخیره می‌شود و این حافظه از طریق واحد مدیریت شیءها(Object Management Unit) قابل دسترسی است.

در یک کاربرد متناظر با شکل (۲) سه شیء O_{A1}، O_{A2}، و O_{B1} تعریف شده‌اند و داده‌های این شیءها همانطور که در شکل (۲) آمده است در حافظه‌ی مرکزی داده ذخیره شده‌اند.

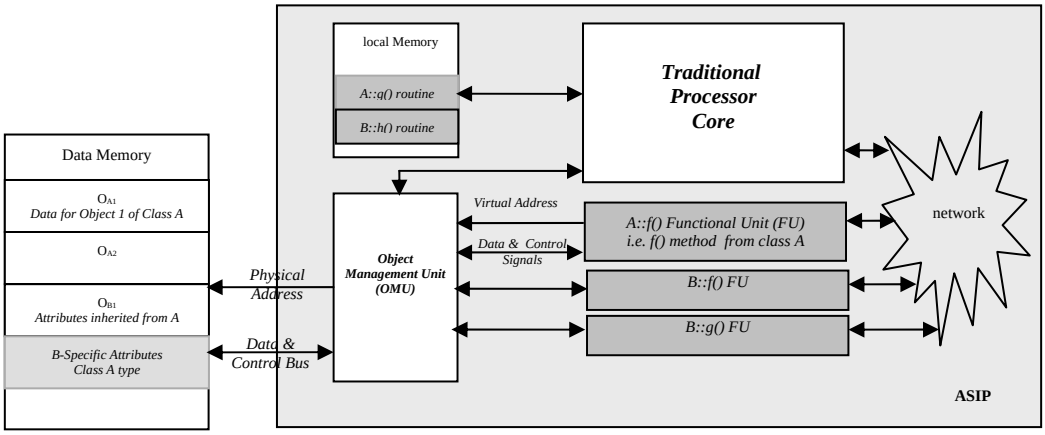
متدهایی از یک کلاس که به بخش‌های سخت‌افزاری نگاشت شده‌اند را متدهای سخت‌افزاری می نامیم و این متدها به صورت واحدهای عملیاتی(FU) پیاده‌سازی می‌شوند. متدهای نرم‌افزاری نیز به صورت روتین‌های نرم‌افزاری در حافظه‌ی محلی هسته‌ی پردازنده‌ی سنتی (گوشه‌ی بالا سمتِ چپِASIP در شکل(۲)) ذخیره می شوند[6].

۳- متدولوژی استفاده از هسته‌هایIP

استفاده از هسته‌هایIP در طراحی سیستم‌های پیچیده‌ی امروزی دو هدف را دنبال می‌کند: یکی زمان کوتاهتر رسیدن محصول به بازار و دیگری بهینه بودن سیستم ایجاد شده در مقایسه با سیستم حاصل از سنتز سطح سیستم خودکار.

متدهای سخت‌افزاری در معماری ASIP همانطور که در فرآیند سنتز این معماری توضیح داده شد در مرحله‌ی سنتز پایین دستی توسط یک ابزار سنتز خودکار(سیستم سی کامپایلرِ شرکت synopsys در ابزار سنتز کنونی ادیسه) به توصیف در سطحِ گیت تبدیل شده و نهایتاً بر روی یک بُرد FPGA سنتز می‌شود. هدف ما، استفاده از هسته-هایIP در سطح گیت یا انتقال ثبات برای پیاده‌سازی متدهای سخت-افزاری به جای سنتز خودکار توسط ابزار سیستم سی کامپایلر می‌باشد. بدین منظور ابتدا کتابخانه‌ای از هسته‌هایIP با توصیف در سطح انتقال ثبات به فرمتVHDL ایجاد می‌کنیم؛ اما با توجه به اینکه این هسته-هایIP توسط روال سنتز ادیسه ایجاد نمی‌شوند برای اینکه با پروتکل موجود در معماریASIP سازگار شوند برای هر کدام از آنها یک wrapper نیز طراحی می‌کنیم. بدین ترتیب این هسته‌ها به همراه wrapperهایشان در سطح انتقال ثبات و با فرمتVHDL، یک کتابخانه را تشکیل می‌دهند.

از آنجایی که سیستم به صورت شیءگرا و با زبان C++ طراحی می-شود و به عنوان ورودی به ابزار سنتز ادیسه داده می‌شود برای هر هسته‌یIP که در واقع نماینده‌ی یک متد سخت‌افزاری است باید متدی متناظر با آن در توصیف C++ سیستم موجود باشد؛ بنابراین برای هر هسته‌یIP یک مدل C++ نیز ایجاد شد تا طراح سیستم نهفته در سطح بالای طراحی نیز از قابلیت استفاده‌ی مجددِ IP بهره بگیرد. در نتیجه مطابق شکل (۴) دو کتابخانه از هسته‌هایIP و wrapperهای متناظر آنها معرفی می‌شود که یک کتابخانه مربوط به توصیف C++ هسته‌ها می‌باشد و یک کتابخانه مربوط به توصیفِ در سطحِ انتقال ثباتِ هسته‌ها با فرمتVHDL می باشد.



شکل(۲): معماری داخلی ASIP متناظر با کلاس A شامل متدهای f() و g() و کلاس B مشتق شده از کلاس A شامل متدهای دوباره تعریف شده‌ی f() و g() و متد جدید h() [6]

```
class A {
  void f();
  void g();
  …; // other member-functions and attributes
};

class B extends A {
  void f(); // A::f() is overridden here
  void g(); // A::g() is overridden here
  void h();
  ...; // other member-functions and attributes
};
```

شکل(۳): شبه کد متناظر با معماری ASIP در شکل (۲)

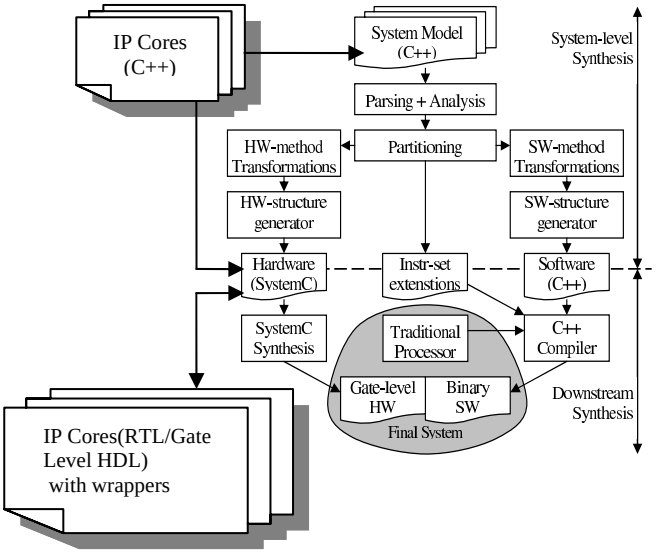
بین عناصر این دو کتابخانه تناظر یک به یک وجود دارد یعنی برای هر هسته با توصیف VHDL یک هسته نیز متناظر با آن و با توصیف C++ وجود دارد. این متدولوژی این امکان را ایجاد کرده است که حتی با وجود هسته‌های IP بتوان در تمام سطوح شبیه‌سازی توأمان موجود در محیط ادیسه، سیستم را شبیه‌سازی کرد.

با وجود کتابخانه‌ای از هسته‌های مشخص و wrapperهای آنها تغییراتی در ابزار سنتز ادیسه ایجاد کردیم تا مجتمع‌سازی هسته‌های IP بطور خودکار انجام شود. بدین ترتیب فقط کافی است طراح یکی از هسته‌های IP کتابخانه را در مدل C++ طرحِ خود استفاده کند، پس از آن ابزار به طور خودکار هسته‌ی IP را تشخیص داده و اعمالِ مربوط به قرار دادن آن در ASIP و سازگار کردن آن را با مابقیِ سیستم انجام می‌دهد.

جزییات دو کتابخانه‌ی معرفی شده در بخش ۳-۱ توضیح داده می‌شود.

۳-۱- کتابخانه‌های هسته‌های IP

توصیف VHDL هسته‌های IP توسط ابزار Core Generator[7] شرکت Xilinx انجام گرفته و برای هر هسته‌ی تولید شده، یک wrapper طراحی شده است که آن را با پروتکل ارتباطی موجود در معماری ASIP سازگار می‌سازد.



شکل(۴): روال سنتز تک پردازنده با قابلیت استفاده از هسته‌های IP برای پیاده سازی متدهای سخت افزاری

برای ایجاد مدل C++ هسته‌های IP آنچه در مورد یک هسته اهمیت دارد، روتین هسته و داده هایی می باشد که این روتین روی آنها اعمال می شود. بنابراین یک هسته‌ی IP را می توان با یک کلاس در C++ به صورتی که در شکل (۵) آمده است تعریف کرد.

Data-in و Data-out به ترتیب داده های ورودی و خروجی هسته می باشند و عملکرد هسته در متد Run( ) پیاده سازی شده و توسط این متد اجرا می‌شود. برای حفظ امنیت کد منبع هسته در این توصیف (دسترسی نداشتن کاربر به الگوریتم عملکرد هسته) یک dll (Dynamic Link Library) در C++ تعریف می شود و کلاسی که برای توصیف سطح سیستم هسته ذکر شد از این کتابخانه بیرون فرستاده می شود. از این به بعد کلاس تعریف شده را "کلاس هسته" می نامیم.

```
Class  Core-Name{
Public:
Type Data-in;
Type Data-out;
Type Run();
}
```

شکل(۵): مدل C++ یک هسته‌ی IP

برای استفاده از کلاس‌های هسته باید wrapperهایی طراحی شود. بنابراین کلاسی مطابق با آنچه در شکل (۶) آمده است تعریف می‌شود. این کلاس، که از این بعد آن را "کلاس wrapper" می نامیم، کلاس هسته را با ابزار ادیسه سازگار ساخته است. کاربر برای استفاده از هسته در سطح سیستم باید شیء هایی از کلاس wrapper نمونه سازی کند. کاربر مستقیماً با کلاس هسته در ارتباط نمی‌باشد.

```
Class Core-Name-Wrapper{
Public:
Type Wrapper-in;
Type Wrapper-out;
Type Wrapper-Run();
}
```

```
Type Core-Name-Wrapper::Wrapper-Run()
{
Core-Name  object;
object.Data-in = Wrapper-in;
object.Run();
Wrapper-out = object.Data-out;
}
```

شکل(۶): کلاسِ wrapper و عملکرد آن

پس از ایجاد این مدل‌ها ابزار ادیسه را تغییر دادیم تا توصیفِ سیستم سی هسته‌های IP بگونه‌ای تولید شود که مدل شبیه‌سازی توأمان ایجاد شده توسط ابزار سنتز ادیسه صحیح باشد.

پس از شبیه‌سازی توأمانِ سطح بالا که به زبان سیستم سی است، یک مدل شبیه‌سازی دیگر نیز در سطح انتقال ثبات داریم که معماری ASIP با فرمت VHDL توصیف شده است. برای شبیه‌سازی در این سطح نیز هسته‌های IP با توصیف VHDL به همراه wrapperهایشان به طور خودکار در سیستم قرار می‌گیرند و بدین ترتیب شبیه‌سازی در این سطح نیز با وجود هسته‌های IP امکان‌پذیر می‌شود.

۴- پیاده سازی IDCT دوبعدی با استفاده از هسته‌ی IP

برای ارزیابی متدولوژی ارایه شده یک تابعِ IDCT دو بعدی با استفاده از این متدولوژی در محیط ادیسه طراحی و پیاده‌سازی شد. کد C++ که در شکل (۷) آمده است مدل شیءگرای سیستم را نشان می‌دهد.

```
class idct{
public:
short data [64];
void run();
};
class CoreIdct{
public:
short cdata [64];
void CoreRun();
};
void CoreIdct::CoreRun(){
idct core;
int i;
for(i=0;i<64;i++)
core.data[i]=cdata[i];
core.run();
for(i=0;i<64;i++)
cdata[i]=core.data[i];
};
void main ()
{
  int j;
  CoreIdct a;
  for(j=0;j<64;j++)
  a.cdata[j]=j;
  a.CoreRun();
  for(j=0;j<64;j++)
  printf("\n %d",a.cdata[j]);
}
```

شکل(۷): مدل شیءگرای سیستم

در این کد کلاس idct، کلاسِ هسته است و کلاس CoreIdct، کلاسِ wrapper آن می‌باشد. در ()main یک شیء از نوع کلاسِ wrapper نمونه‌سازی شده است که بر روی آرایه‌ی data[64] عمل می‌کند. این برنامه‌ی کاربردی به ابزار سنتز ادیسه‌ی تغییر یافته داده شد و مدل شبیه‌سازی توأمان به درستی ایجاد شد.

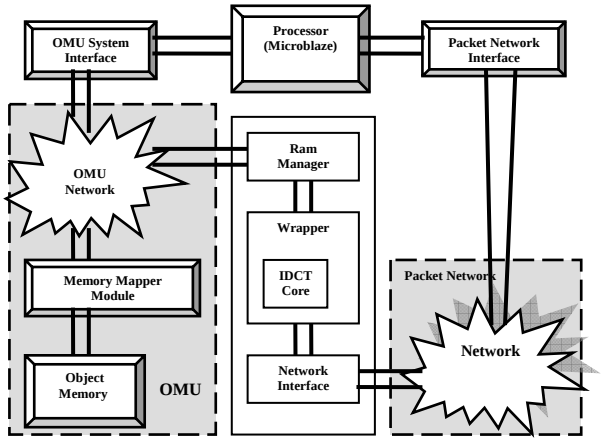
ساختار ASIP تولید شده در سطح انتقال ثبات نیز مطابق شکل (۸) است که کلاس هسته‌ی idct با هسته‌ی IDCT دو بعدی Xilinx که در سطح گیت است و توسط ابزار CORE Generator ایجاد شده، جایگزین شده است و یک wrapper نیز در سطح انتقال ثبات با فرمت VHDL در این ساختار وجود دارد که هسته‌ی IDCT دو بعدی Xilinx در آن قرار می‌گیرد تا با پروتکل موجود در ساختار ASIP سازگار شود.

ساختار ASIP در سطح انتقال ثبات توسط ابزار EDK7.1 [8] (embedded system development tool) ایجاد شده است و این سیستم توسط Modelsim شبیه‌سازی و درستی یابی شده است بدین صورت که خروجی تابع IDCT در این سطح با خروجی مدل شیءگرای این تابع مقایسه شد و خروجی‌ها یکسان بودند. نهایتاً سیستم توسط ISE7.1 [9] سنتز و پیاده‌سازی شده است.

۵- نتایج شبیه سازی و پیاده‌سازی

تابع IDCT دو بعدی طراحی شده، بر روی یک بُرد FPGA شرکت Xilinx از نوع Virtex4sx35 پیاده‌سازی شد. نتایج به دست آمده از شبیه‌سازی این سیستم، که در محیط ادیسه یک بار با استفاده از هسته‌ی IP و بار دیگر بدون استفاده از هسته‌ی IP طراحی شده است،

در جدول (۱) و نتایج به دست آمده از پیاده‌سازی نیز در جدول (۲) آمده است. با مشاهده‌ی مقادیرجدول (۱) و جدول (۲) می‌توان فهمید که طرح ایجاد شده توسط هسته‌ی IP بسیار بهینه‌تر می‌باشد.



شکل(۸): ساختار ASIP تولید شده برای کاربرد موردنظر

جدول(۱): نتایج حاصل از شبیه‌سازی سیستم

2-D IDCT Function Implementation	Execution Clock Cycles
IP Core IDCT	175
ODYSSEY IDCT	516

جدول(۲): نتایج حاصل از پیاده‌سازی سیستم

2-D IDCT Function Implementation	Number of Slices	Total gate count	Maximum Clock Frequency
IP Core IDCT	2406 out of 15360(15%)	59,132	167.497MHz
ODYSSEY IDCT	5176 out of 15360(33%)	65,757	26.130MHz

علاوه بر نتایج بدست آمده با وجود تابع IDCT در کتابخانه‌ی C++ دیگر نیازی به طراحی تابع IDCT دو بعدی در این سطح نیست و با استفاده‌ی مجدد از تابع موجود زمان طراحی کاهش می‌یابد.

۶- نتیجه

در این مقاله هدف ارایه‌ی یک متدولوژی برای استفاده از هسته‌های IP و مجتمع‌سازی آنها در یک محیط طراحی سطح سیستم سیستم‌های نهفته بود.

یکی از نکات مهم موجود در کارِ ارایه شده، خودکارسازی مجتمع کردن هسته‌های IP در سیستم طراحی شده است. خودکارسازی این فرآیند مشکلاتی مانند طراحی wrapper و نیاز به آشناییِ کامل با جزییات زمانی و عملکرد هسته‌ی IP به منظور طراحی wrapper مناسب آن را از مجموعه‌ی وظایف طراح خارج می‌سازد.

استفاده از هسته‌های IP در طراحی سیستم‌های پیچیده‌ی امروزی زمان طراحی و در نتیجه زمان رسیدن محصول به بازار را کاهش می‌دهد. همانطور که نتایج بدست آمده در این مقاله نشان داد طرح پیاده‌سازی شده با استفاده از هسته‌های IP بهتر از طرح ایجاد شده توسط سنتز خودکار است.

در مورد استفاده و مجتمع‌سازی هسته‌های IP در محیط‌های طراحی سطحِ سیستمِ سیستم‌های نهفته کارهای چندانی صورت نگرفته است زیرا موضوع طراحی سطحِ سیستم، خود موضوعی جدید است.

کارهایی که در آینده می‌تواند در محیط ادیسه برای استفاده از هسته‌های IP و مجتمع‌سازی آنها انجام شود عبارتند از: تولید خودکار wrapperها که این امر توسعه‌ی کتابخانه‌ی هسته را راحت‌تر می‌سازد؛ طراحی wrapperهایی برای تبدیل پروتکل‌های استاندارد OCP و VCI به پروتکل ارتباطی موجود در ASIP تا بدین ترتیب بتوان از هسته‌های IP با واسط‌های استاندارد نیز در محیط طراحی ادیسه استفاده کرد.

سپاسگزاری

این پروژه با حمایت مالی مرکز تحقیقات مخابرات ایران انجام گرفته است که از آن مرکز قدردانی می‌نماییم.

مراجع

[۱] م. گودرزی،" ادیسه: یک متدولوژی طراحی و خودکارسازی تولید سیستم‌های نهفته از مدلهای شیءگرا ،" پایان‌نامه دکترا، دانشکده کامپیوتر، دانشگاه صنعتی شریف،۱۳۸۴ .
[2] G.E. Spivey, "LOGIC FOUNDRY: A DESIGN ENVIRONMENT FOR THE RAPID PROTOTYPING OF FPGA-BASED DSP SYSTEMS," Dissertation for the degree of Doctor of Philosophy, University of Maryland,2002.
[3] E. Filippi, L. Lavagno, L. Licciardi, A. Montanaro,"Intellectual Property Re-use in Embedded System Co-design: an Industrial Case Study," ISSS,1998.
[4] M. Haldar, A. Nayak, A. Choudhary, P. Banerjee, "A SYSTEM FOR SYNTHESIZING OPTIMIZED FPGA HARDWARE FROM MATLAB," ICCAD 2001.
[5] S.K. Shukla, F. Doucet,R.K. Gupta, "Structured Component Composition Frameworks for Embedded System Design," 9 th International Conference on High Performance Computing,2002.
[6] N. Mohammadzadeh, S. Hessabi, M. Goudarzi," Software Implementation of MPEG2 Decoder on an ASIP JPEG Processor,"ICM,2005.
[7] CORE Generator Guide, www.xilinx.com
[8] Embedded development kit, www.xilinx.com
[9] ISE Foundation, http://www.xilinx.com/products

[1] - Productivity
[2] - Register Transfer Level
[3] -Partitioning