دانشگاه صنعتی شریف

Skein Hashing

پروژه مستندسازی پیاده‌سازی سخت افزاری الگوریتم به زبان *Verilog*

گروه ۴

سید پارسا اسکندر

کیمیا یزدانی

الهه خدایی

وحید زهتاب

آروین آذرمینا

استاد: فرشاد بهاروند

بهار ۹۸

چکیده‌ی مقاله

در این مقاله که در مورد یکی از روش‌های hash به نام Skein توضیح داده خواهد شد. و هم‌چنین توضیح و مستند برنامه‌های داده شده به دو زبان C به عنوان مدل طلایی و verilog به عنوان طرح سخت‌افزاری ، آورده خواهد شد . در انتها نیز تفاوت‌های این دو مدل و دلایل اختلاف آن‌ها (نقاط اشتباه آن‌ها در پیاده‌سازی) تحلیل خواهد شد.

فهرست مطالب

فصل ۱

مقدمه

توضیییییییح اسکیییییین

فصل ۲

پیاده‌سازی سخت‌افزاری Verilog

۱.۲ مقدمه

دنیای نرم‌افزار و سخت‌افزار رایانه در نگاه کلی می‌توانند بسیار شبیه به هم باشند، برنامه های نرم‌افزاری، مقادیری را به عنوان ورودی دریافت کرده، سپس طی روند مشخصی محاسباتی روی آنها انجام داده و در نهایت مقادیری را به عنوان خروجی به کاربر خود تحویل می دهند، قطعات سخت‌افزاری نیز دارای port هایی برای ارتباط با دنیای خارجی و دریافت ورودی و تحویل خروجی‌های خود می‌باشند و از واحدهای مختلف پردازشی و عملیاتی مختلفی برای محاسبه ی خروجی‌های مناسب تشکیل شده اند. درعمل می‌توان سخت‌افزاری خاص برای اجرای بسیاری از روند های نرم‌افزاری طراحی و پیاده سازی کرد، ساخت سخت‌افزارِ خاصِ مربوط به یک الگوریتم می تواند کاربردهای بسیاری داشته باشد، برای مثال قطعه‌ای که بتواند داده های ورودی را رمزنگاری کند می‌تواند به صورت گسترده برای ذخیره ی اطلاعات به صورت سریع استفاده شود، سخت‌افزار های اختصاصیِ الگوریتم ها سریع و بهینه اند و میتوانند به اجرای هرچه سریع تر روندهای پیچیده‌ای که به الگوریتم مورد نظر وابستگی فراوان دارند کمک کلانی کنند.

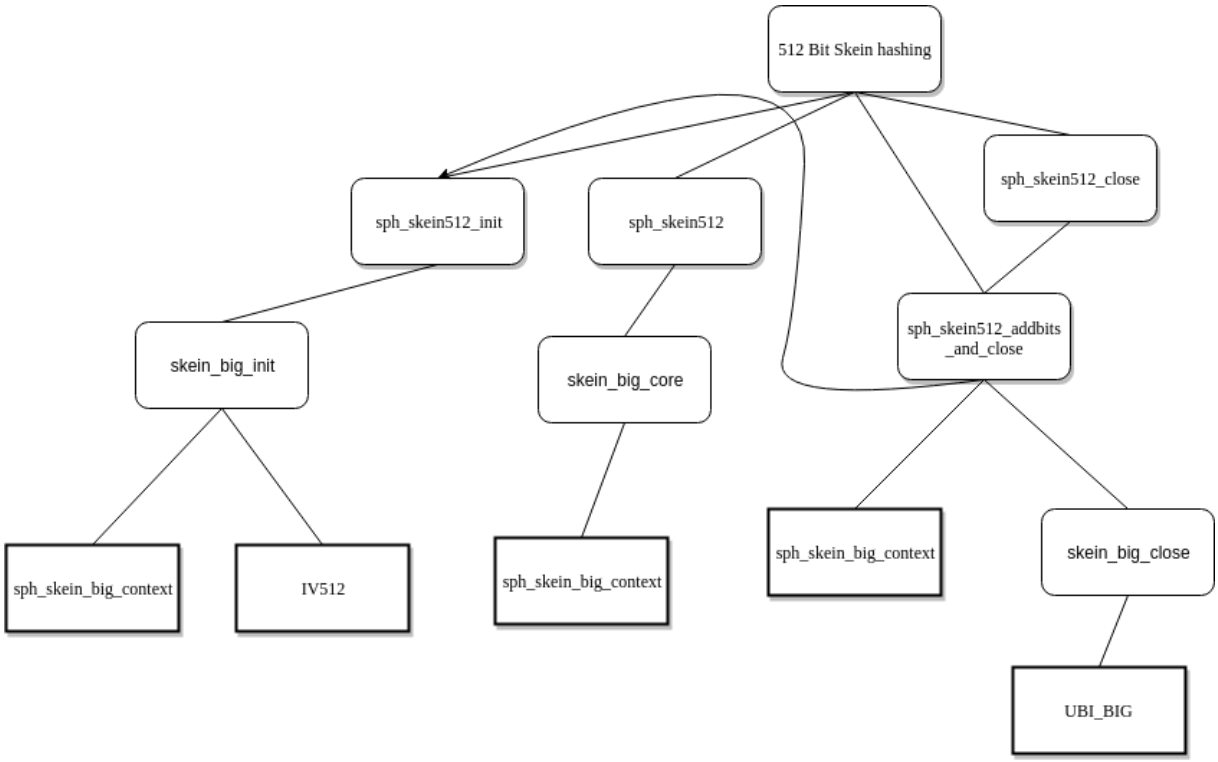
همانند بسیاری از الگوریتم‌های رایانه‌ای، الگوریتم تابعِ Skein Hashing که در بخش قبل کلیتی از آنرا معرفی کردیم را می توان به صورت سخت افزاری پیاده‌سازی کرد، بدین صورت که قطعه‌ای طراحی و پیاده‌سازی کنیم که ورودی ای به اندازه ی دلخواه مارا دریافت و حاصل درهم‌سازی را به صورت خروجی ای به اندازه ی مورد نظر ما خروجی دهد. بر اساس نیاز و کاربرد ما از این قطعه، اندازه‌ی ورودی و خروجی را می توان ثابت و به مقدار دلخواه درنظر گرفت، سپس قطعه‌ای ثابت با پیاده سازیِ بهینه‌ای برای اندازه های مورد نظر طراحی کرد، یا این که قطعه‌ای برای ورودی و خروجی های با اندازه های متغیر طراحی و پیاده سازی کرد. همان طور که در بخش قبل توضیح داده شد، تابع درهم‌سازی Skein Hashing میتواند ورودی‌ای با اندازه ی دلخواه را دریافت کند و خروجی ای با اندازه‌ی دلخواه تحویل دهد، در این مقاله تمرکز ما روی پیاده سازی سخت‌افزاری حالتی از این تابع می باشد که اندازه ی بلاک های درونی تابع (حالت درونی تابع) **۵۱۲ بیت** و حاصل درهم‌سازی نیز به صورت خروجی ای به اندازه ی **۵۱۲ بیت** می باشد، این تابع در اصطلاح *Skein 512-512* نامیده می‌شود.

فصل ۳

مدل طلایی

۱.۳ مقدمه

در مدل طلایی ۴ نوع متفاوت از Skein hash آورده شده‌است (۲۲۴ و ۲۵۶ و ۳۸۴ و ۵۱۲ بیت) که همانطور که در مدل طراحی شده با verilog نیز تنها نوع استاندارد (۵۱۲ بیت)آن پیاده‌سازی شده است ، در مدل طلایی نیز تنها توضیحات و مستندات این نوع ارائه خواهد شد.



۲.۳ پیاده‌سازی الگوریتم

در شکل بالا تمامی توابع و ساختارهای مورد نیاز و سلسله مراتب آن‌ها برای نوع ۵۱۲ بیتی الگوریتم آورده شده است ، برای توضیح نحوه‌ی اجرای الگوریتم با شروع از sph-skein-big-context سلسله اجرای برنامه توضیح داده خواهد شد.

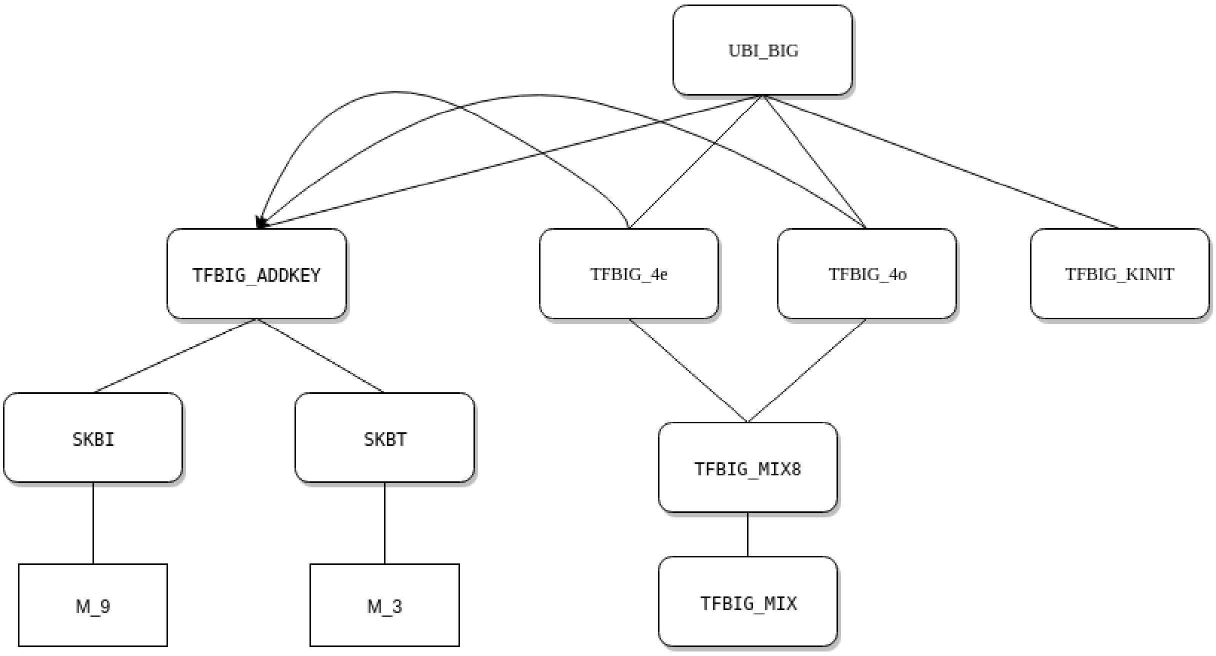
در این برنامه برای ذخیره و استفاده از هش ، از ساختاری استفاده شده است به نام sph-skein-big-context استفاده شده است و هدف برنامه اجرای الگوریتم هش و ذخیره‌ی خروجی در این ساختار است.

برای اجرای الگوریتم هش ۵۱۲ بیتی ، در سلسله‌ی اجرا از توابع زیر استفاده شده است :

در ابتدا برنامه با ذخیره‌ی مقادیر از پیش تعیین شده IV512 در ساختار معرفی شده شروع به کار می‌کند ، و این کار توسط تابع sph-skein512-init انجام میگردد.

سپس با در نظر گرفتن ورودی و سایز این ورودی، اجرای الگوریتم هش توسط تابع sph-skein512 شروع می‌شود و ورودی داده شده تبدیل به هش میشود و با ذخیره شدن در ساختار هش، این تابع پایان می‌پذیرد.

حال برای فهم درست از توابع مورد استفاده لازم است نحوه‌ی پیاده‌سازی UBI-BIG توضیح داده‌شود. تمامی سلسله مراتب طراحی آن در شکل زیر آورده شده‌است.



یه توضیح عه کوچیک براااا یو بی آی بیگ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

فصل ۴

توابع و ساختارها

۱.۴ ساختارها

۱.۱.۴ sph-skein-big-context

این ساختار مورد نظر برای ذخیره و استفاده از هش است (شامل مقادیری از هش قبلی و مقادیر جدید محاسبه شده).
این ساختار شامل یک آرایه‌ی ۶۴ بیتی از کاراکترهاست که به منظور تراز کردن انواع هش استفاده می‌گردد و هشت عدد ۶۴ بیتی که برای ذخیره‌ی ۵۱۲ بیت هش استفاده می‌شوند و هم‌چنین شامل دو عدد با نام‌های ptr, bcount است که این دو عدد به طور معمول برابر ۰ هستند که همانند nonce در پیاده‌سازی وریلاگ آن است.

۲.۱.۴ IV512

این ساختار شامل مقادیر اولیه‌ی هش است. یک عدد ۵۱۲ بیتی را برای خوانا بودن در مبنای ۱۶ و در ۸ بلاک ۱۶ بیتی نگاه می‌دارد. این مقدار در برنامه به زبان وریلاگ همان midstate است.

۳.۱.۴ UBI-BIG

این تابع (در مدل‌طلایی به صورت define تعریف شده) طبق الگوریتم skein در ابتدا وظیفه‌ی ۵۱۲ بیتی کردن ورودی در بافر را بر عهده دارد، در ادامه تمامی ۷۲ مرحله‌ی هش الگوریتم skein را که در مقدمه شرح داده شده است را اجرا می‌کند.
روند اجرای این تابع بدین صورت است که در ابتدا مقادیر ۸ بیتی در بافر را با استفاده از Encoder به مقادیر ۶۴ بیتی تبدیل می‌کند، سپس دو مقدار $t_{۱}, t_{.}$ را که همان $tweak$ ها هستند را با استفاده از ورودی‌ها به‌دست می‌آورد و سپس با استفاده از تابع TFBIG-INIT مقادیر جدیدی از داده‌های قبلی به دست می‌آورد، سپس ۱۸ بار توابع TFBIG-4e و TFBIG-4o صدا می‌شوند که در هر کدام از این تابع‌ها ۴ بار تابع درهم‌سازی توضیح داده شده در مقدمه صدا می‌شوند.

۴.۱.۴ TFBIG-4e, TFBIG-4o

این تابع برای کدگذاری $P_{.}$ تا $P_{۷}$ طراحی شده‌است. همان‌طور که پیش‌تر توضیح داده شده است، ۷۲ بار تابع درهم‌سازی صدا می‌شود، و هر ۸ سلسله از این ۷۲ مرحله یکسان است، هم‌چنین در هر ۸ سری ۴ بار با یک کلید و ۴ بار دیگر با یک کلید دیگر اجرا می‌شود، که به همین دلیل این توابع هر کدام برای آن ۴ باری استفاده می‌شود که در مرحله‌ای زوج یا فرد قرار داریم.
این تابع یک ورودی s دارد. تابع TFBIG-ADDKEY با $p_{.}$ تا $p_{۷}$ و h و t و s به ترتیب به عنوان $w_{.}$ تا $w_{۷}$ و k و t و s صدا شده‌است. h و t برای concat و ساختن کلید در تابع TFBIG-ADDKEY استفاده شده‌اند. سپس TFBIG-MIX8 چهار بار برای ترتیب‌های متفاوتی از $p_{.}$ تا $p_{۷}$ اعداد متفاوت به عنوان rc صدا شده‌است. ترتیب صدا شدن $p_{.}$ تا $p_{۷}$ برای تعداد بلاک ۸ به صورت جدول‌های زیر است، که برای هر راند از ۰ تا ۳، بر حسب راند قبل ترتیب‌ها چهار عدد جابه‌جا شده‌اند. و تفاوت حالت‌های زوج و فرد در اعداد استفاده شده است.

N_w		4		8				16							
j		0	1	0	1	2	3	0	1	2	3	4	5	6	7
	0	14	16	46	36	19	37	24	13	8	47	8	17	22	37
	1	52	57	33	27	14	42	38	19	10	55	49	18	23	52
	2	23	40	17	49	36	39	33	4	51	13	34	41	59	17
$d =$	3	5	37	44	9	54	56	5	20	48	41	47	28	16	25
	4	25	33	39	30	34	24	41	9	37	31	12	47	44	30
	5	46	12	13	50	10	17	16	34	56	51	4	53	42	41
	6	58	22	25	29	39	43	31	44	47	46	19	42	44	25
	7	32	32	8	35	56	22	9	48	35	52	23	31	37	20

		$i =$															
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	4	0	3	2	1												
$N_w =$	8	2	1	4	7	6	5	0	3								
	16	0	9	2	13	6	11	4	15	10	7	12	3	14	5	8	1

TFBIG-INIT ۵.۱.۴

این تابع با ورودی‌های $t.$ تا $t_۲$ و $h.$ تا $h_۸$ مقادیر زیر را محاسبه می‌کند :

$$k_۸ = C \oplus k. \oplus k_۱ \oplus ... \oplus k_۷$$

$$t_۲ = t_۱ \oplus t.$$

که مقدار ثابت C به آن جهت در فرمول وجود دارد که از ۰ نبودن تمامی بیت‌ها اطمینان حاصل شود.

توابع ۲.۴

sph-skein512-init ۱.۲.۴

این تابع مسئولیت مقداردهی اولیه‌ی ساختار هش را بر عهده دارد، که برای آن تابع skein-big-init را با ورودی اولیه‌ی IV512 اجرا می‌کند.

skein-big-init ۲.۲.۴

این تابع دو ورودی می‌پذیرد که یکی از آن‌ها آدرس یک ساختار هش است و دیگری مقدار اولیه، که مقادیر متناظر ساختار داده شده را برابر مقادیر اولیه قرار می‌دهد. که مقدار اولیه در حالت ۵۱۲ بیتی در ساختار IV512 ذخیره شده است.

sph-skein512 ۳.۲.۴

این تابع، مقدار هش محاسبه شده تا این لحظه را از بین می‌برد و