به نام خدا

گزارش طراحی لایه فیزیکی 802.11a

سینا کریمی 97105509

این سخت افزار شامل 4 قسمت برای گیرنده و فرستنده میباشد. قسمت اول Scrambler، قسمت دوم Convolutional Encoder، قسمت سوم Data Interleaver و قسمت آخر خروجی دادن موج سینوسی برای مدولاسیون های مختلف میباشد. (لیست فایل ها در آخر این گزارش قرار دارد)

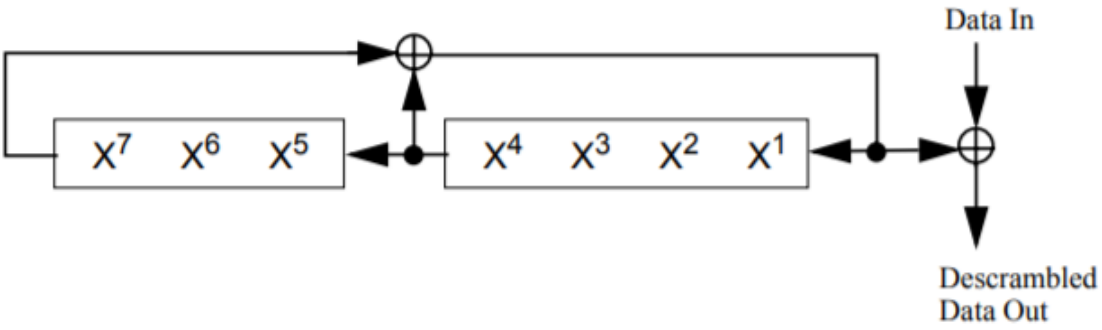
قسمت Scrambler:

- **تئوری و الگوریتم**

الگوریتم مربوط به Scrambling در صفحه 16 در فایل استانداردی که برای پروژه قرار داده شده میباشد. این اسکرمبلر از نوع Additive synchronous scrambler میباشد و رابطه آن به شکل:

$$S(x) = x^7 + x^4 + 1$$

میباشد. شکل آن به صورت زیر میباشد:



حال برای ماژول Descrambler میتوانیم یک ماژول scrambler را در قسمت گیرنده داشته باشیم و آن را با کمک بیت های سرویس با ماژول داخل فرستنده سنکرون کنیم و بعد آن خروجی XOR سمت راست مربوط به دیتا و یکی از ورودی های آن را داریم، پس کافی است که این دو را در یکدیگر دوباره XOR کنیم که ورودی دوم به دست بیاید.

برای سنکرون سازی این دو ماژول از 7 بیت 0 در فیلد service استفاده میکنیم. هر بار که یک 0 وارد ماژول scrambler میشود، این 0 با بیت اول بیت های داخلی ماژول XOR میشود و به دلیل اینکه 0 است خروجی آن برابر همان بیت اول درون Scrambler میشود. در شکل و رابطه نیز میتوانیم ببینیم که بیت ها در 7 کلاک تغییر میکنند، پس اگر در یک تناوب 7 کلاکی ما 7 بیت 0 را وارد این ماژول بکنیم خروجی ما بیت های داخلی Scrambler بعد از خروج آخرین 0 خواهد بود بنابراین این ماژول در گیرنده دقیقا میداند که به چه صورت باید تنظیم بشود که بتواند دیتا را دست دیکود بکند. بعد آن ماژول داخل گیرنده باید یک کلاک جلوتر برود زیرا حالتی که 0 ها گزارش میکنند حالتی است که آخرین 0 دیده است و یک کلاک بعد که دیتا وارد میشود ماژول داخل فرستنده یک بار جلوتر رفته است بنابراین ماژول گیرنده نیز باید 1 کلاک جلو برود و بر اساس آن دیتا را دیکود بکند.

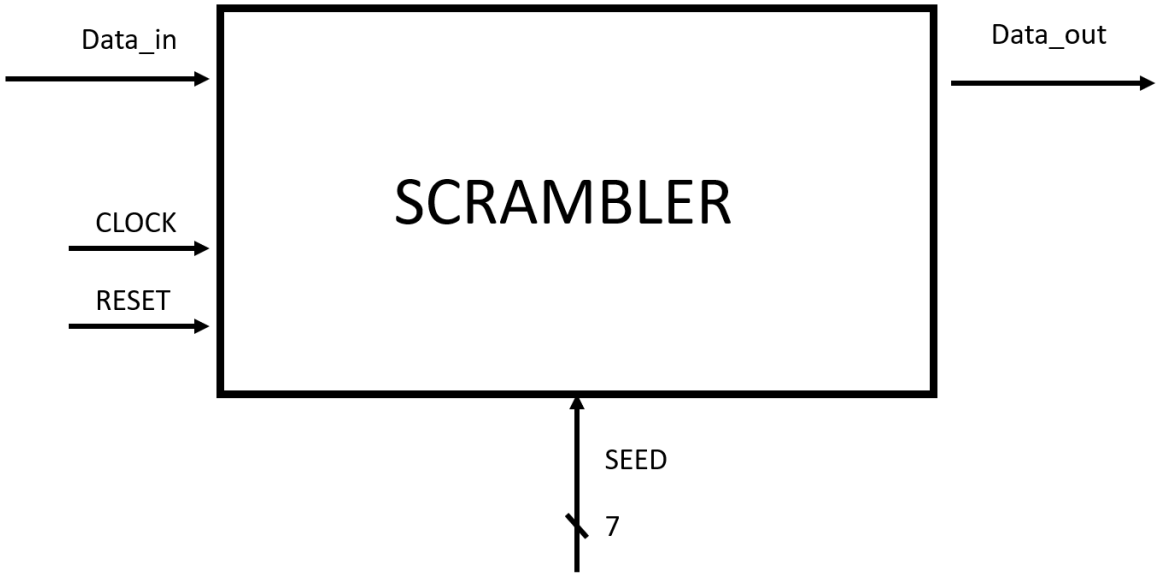
- **ساختار فریم**

در کل فریم تنها قسمت هایی که نیاز است آنها را اسکرمبل کنیم قسمت مربوط به دیتا میشود و در این قسمت به دلیل این که فقط 7 بیت اول سرویس مهم هستند ما فقط 7 بیت اول سرویس و بقیه دیتا را اسکرمبل میکنیم زیرا قسمت های بعد سرویس رزرو هستند و مقادیر آنها برای ما اهمیت ندارد و در اینجا فقط میخواهیم نشان دهیم که این ماژول درست کار میکند.

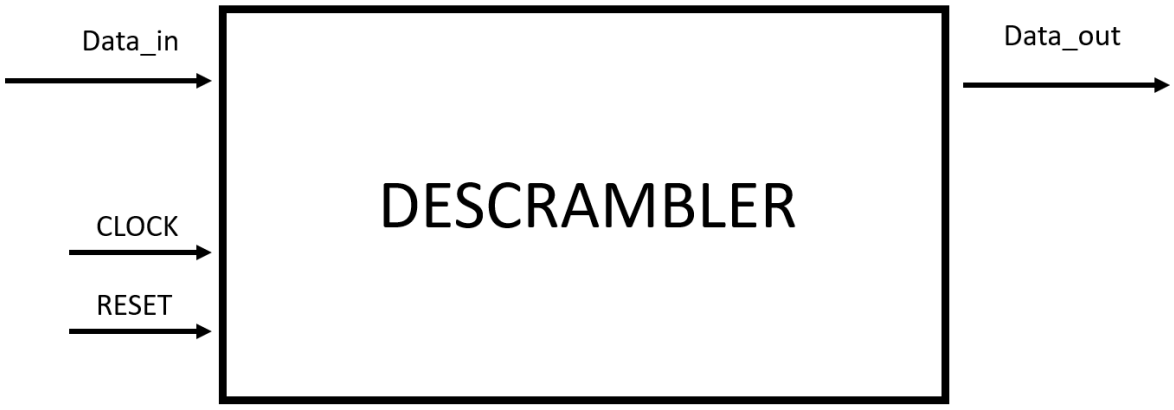
- **ساختار ماژول ها**

همانند شکل بعد ماژول داخل فرستنده دارای پایه های reset, clk, data_in, data_out, seed میباشد.

پایه ریست به این شکل کار میکند که وقتی 1 بشود، بیت های داخل Scrambler به بیت هایی که از طریق seed میدهیم تغییر میکند. Seed یک باس 7 بیتی میباشد. ورودی های دیتا نیز به صورت سریال میباشند.



ماژول Descrambler نیز مانند ماژول Scrambler میباشد با این تفاوت ها که ورودی seed ندارد به دلیل اینکه دیتا به صورت سریال وارد میشود و زمانی که reset میشود، 7 بیت بعد را به عنوان بیت های سنکرون سازی در نظر میگیرد و بعد آن ها دیتا را برای ما descramble میکند.



- **کد متلب**

کد متلب برای این قسمت بر اساس الگوریتمی که بالاتر ارائه شد کار میکند. ورودی این کد یک seed مشخص 7'b0101001 میباشد به دلیل اینکه برای تست ماژول بتوانیم از یک سید مشخص استفاده کنیم و مشکلات دیباگ و

وریفیکیشن کمتر شود. برای ماژول اسکرمبلر بیت های ماژول آرایه x میباشد. این آرایه ابتدا 7 بیت را اسکرمبل میکند تا Descrambler بتواند از آنها استفاده کند سپس نتیجه آن ها را در آرایه service میریزد که مستقیما در descrambler از آن استفاده خواهیم کرد. سپس یک عدد رندوم از 0 تا 127 انتخاب میشود و آن را به صورت یک عدد 7 بیتی در می آوریم و از همان مقادیر x که بعد از اسکرمبل کردن بیت های 0 به دست آمده اند استفاده میکنیم و دیتای آن را اسکرمبل میکنیم. در این حین دیتای باینری قبل اسکرمبل شدن را در یک خط در یک فایل تکست و دیتای اسکرمبل شده را در خط بعدی مینویسیم. به دلیل اینکه سید را یکی در نظر گرفتیم نیازی به دیتای دیگری برای تست نمیباشد (در گروه نیز من این سوال را پرسیدم و پاسخی که گرفتم این بوده که نیازی نیست که سید تصادفی باشد ولی حتی اگر نیازی باشد که تصادفی باشد فقط کافی است یک متغیر در کد وریلاگ تغییر کند که در قسمت های بعد توضیح داده خواهد شد.).بعد این کار همین دیتای اسکرمبل شده را دوباره به ماژول Descrambler میدهیم و میتوانیم در کنسول مشاهده کنیم که دیتای ورودی و خروجی برابر هستند. این عمل را 20 بار تکرار میکنیم تا 40 زوج دیتای وروی و Scrambled داشته باشیم. دوباره نیازی برای تولید دیتای جداگانه برای تست Descrambler نمیباشد زیرا میتوان از همان فایل درست شده به صورتی که ورودی دیتای اسکرمبل شده باشد و با دیتای اصلی مقایسه شود استفاده کرد. خروجی این فایل به نام test.txt میباشد.

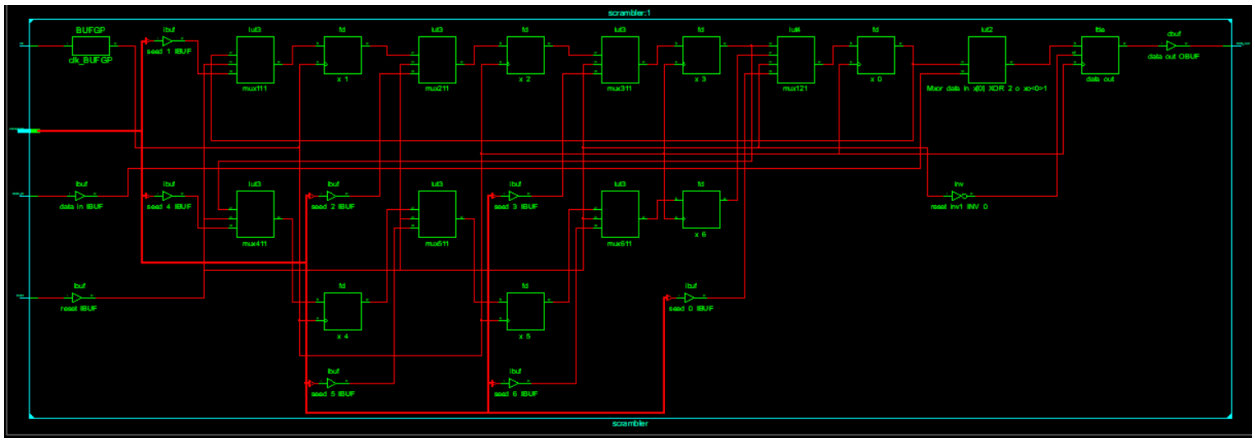
- **کد وریلاگ**

در کد وریلاگ این 2 ماژول برای اینکه چک کردن جواب ماژول ها به صورت چشمی راحت تر باشد خروجی به صورت blocking میباشد اما اگر به non blocking تغییر کند هیچ مشکلی در کد به وجود نمی آید.

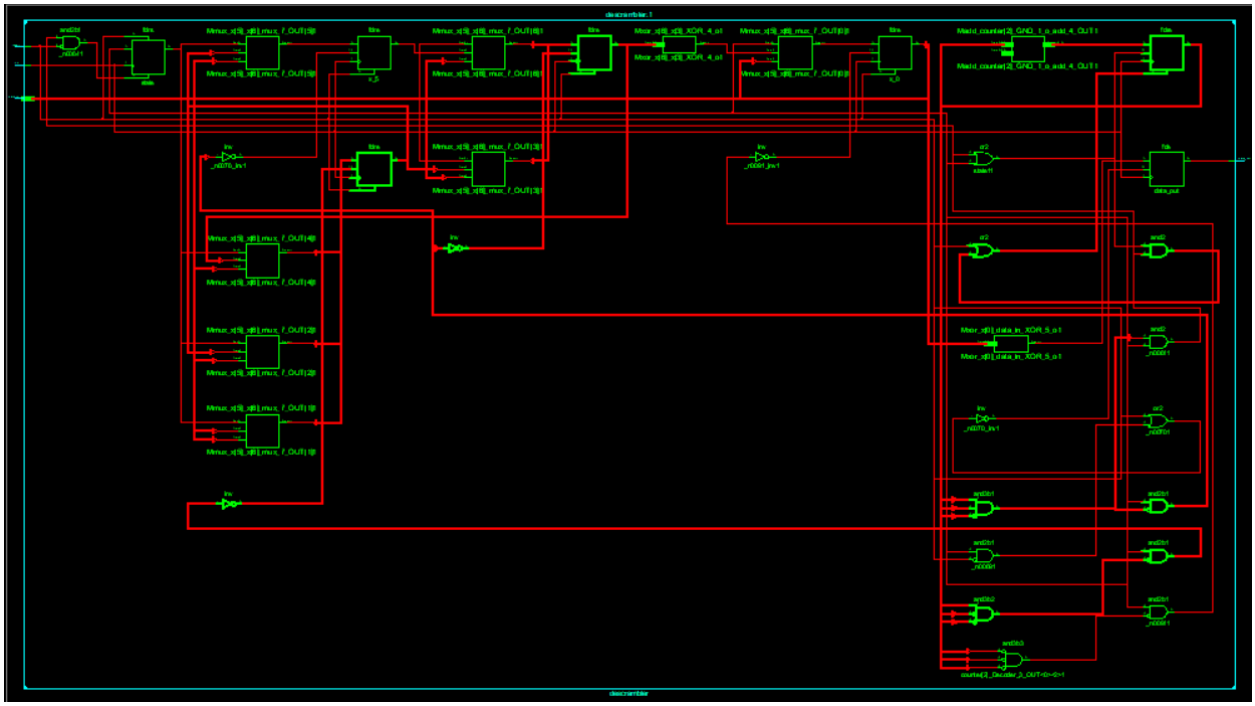
ماژول Scrambler در فایل Scrambler.v میباشد که در این ماژول دقیقا همان الگوریتمی که بالا گفته شده پیاده شده است و هیچ نکته خاصی برای آن وجود ندارد. در هر کلاک یک بیت را اسکرمبل میکند.

ماژول Descrambler در فایل Descrambler.v میباشد. این ماژول ابتدا زمانی که ریست میشود به state 1 که حالت گرفتن 7 صفر اول برای synchronization میباشد میرود و شمارنده داخلی 0 میشود. ابتدا یک پالس ریست دریافت میکند سپس بعد از آن در هر کلاک یک دیتا میگیرد و یک شمارنده داخلی دارد که میشمارد چند بیت وارد آن شده است. وقتی بیت آخر را میخواهد بگیرد متغیر state به 0 و شمارنده نیز 0 میشود و از این به بعد هر دیتایی را که بگیرد descramble میکند. در حالت اول خروجی آن Valid نمیباشد.

نتیجه سنتز برای Scrambler:



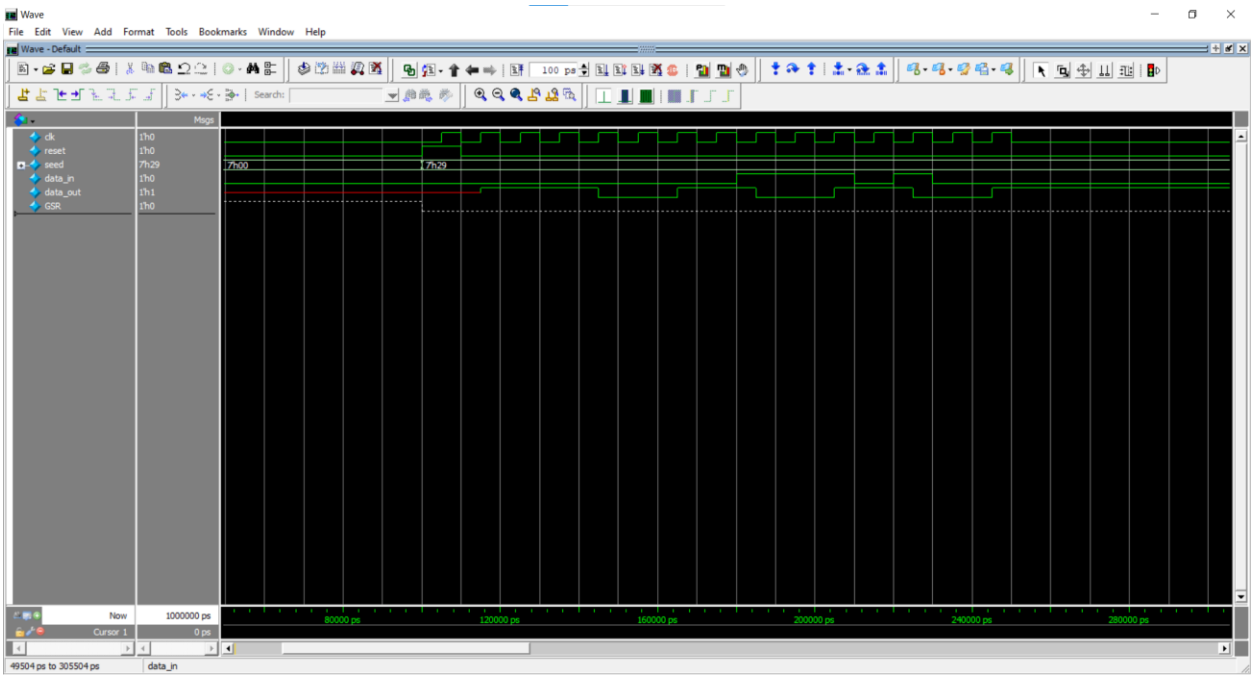
نتیجه سنتر برای Descrambler:



مشخصا مدار مربوط به Descrambler بزرگتر است زیرا علاوه بر قابلیت های scrambler قابلیت های دیگری نیز دارد.

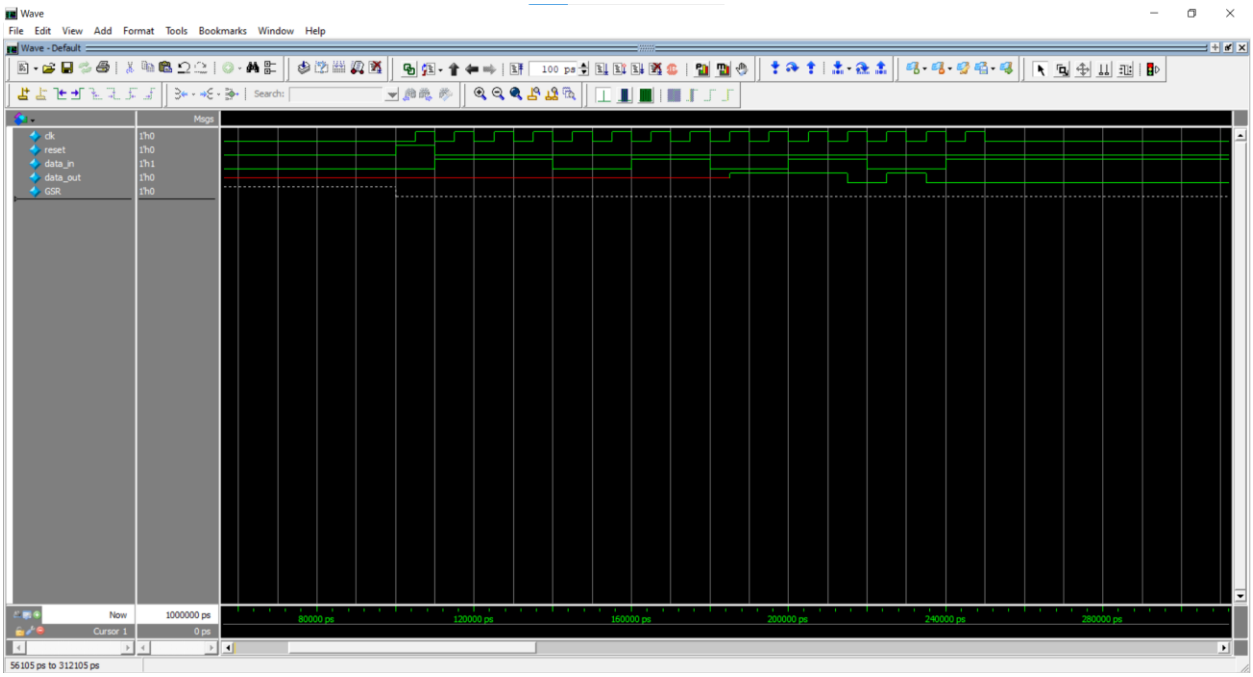
- **نتایج و تست ماژول ها**

این ماژول ها شامل 2 تست بنچ میباشند که برای هر کدام یکی برای تست تک به تک بیت ها میباشد و دیگری فایل test.txt را بر روی آنها اجرا میکند و نتیجه را اعلام میکند. برای Scrambler نام تست بنچ تکی scrambler_tb.v میباشد و نتیجه اجرا این ماژول به صورت زیر میباشد:

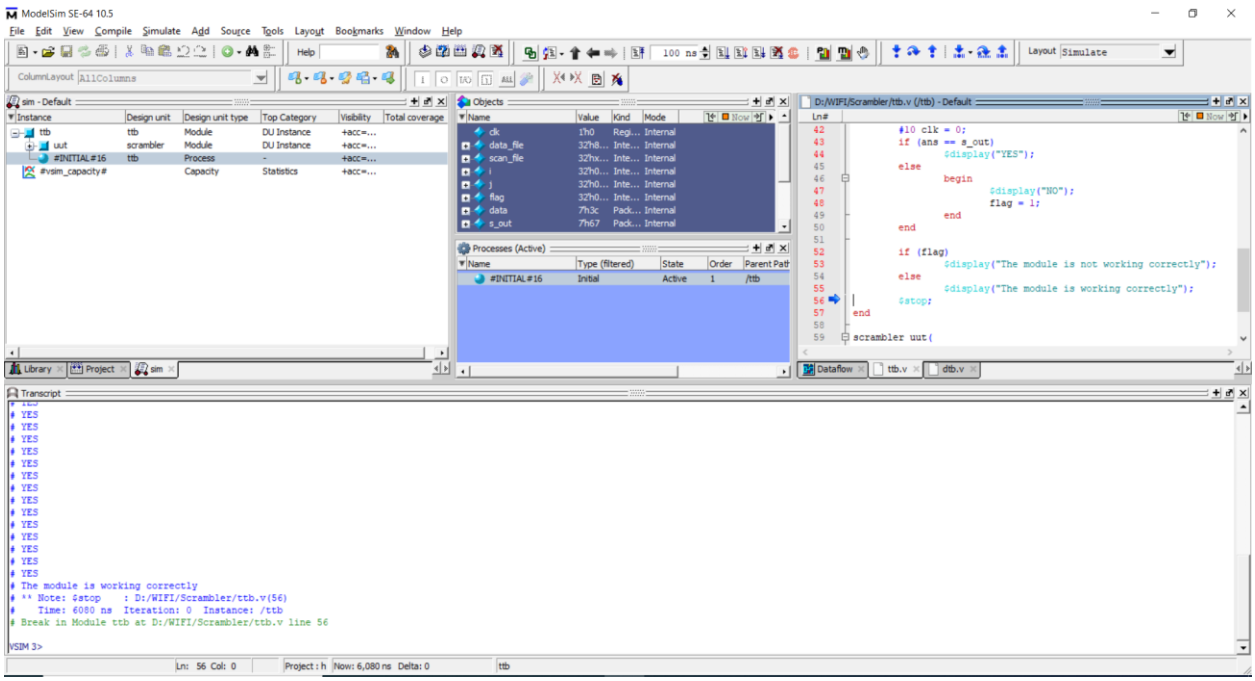


در این ماژول سید ما همان سیدی است که در فایل متلب به کار رفته است و ورودی آن عدد 23 به صورت یک عدد باینتری 7 بیتی میباشد.

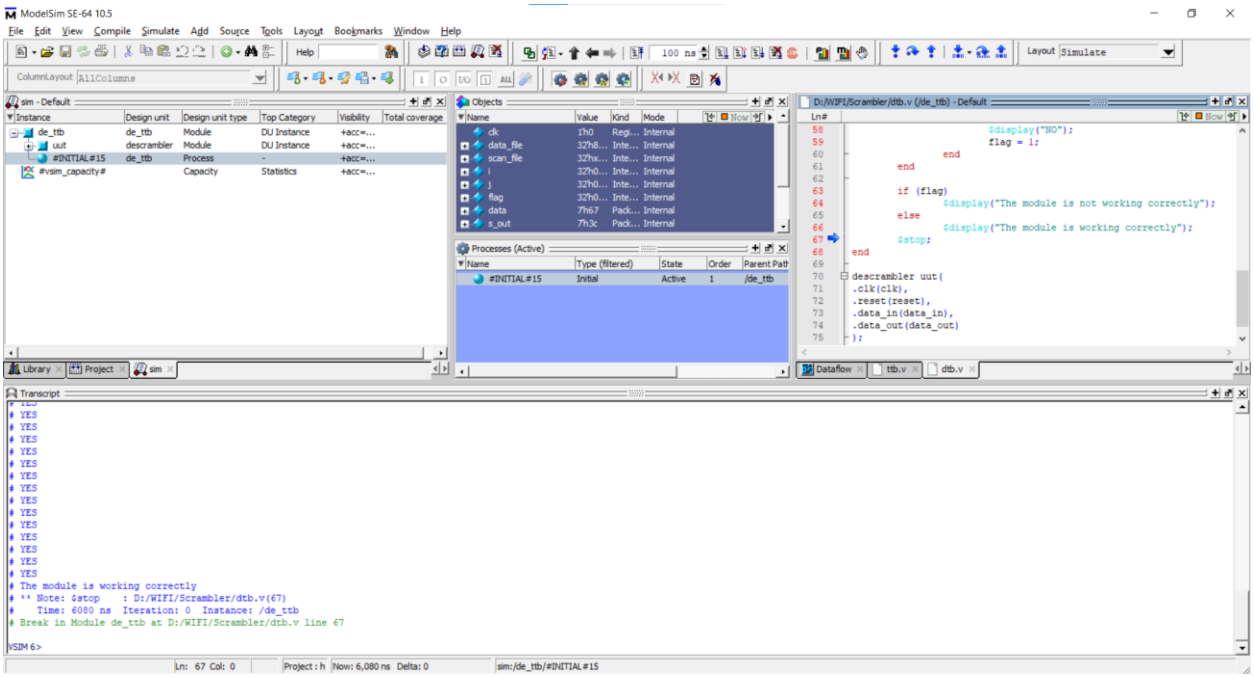
برای descrambler فایل مربوط به تست تکی به نام descrambler_tb.v میباشد و برای این ماژول نیز سنکرون سازی مربوط به سید بالا را داده ایم و ورودی آن نیز اسکرمبل شده عدد 23 میباشد:



برای تست با دیتای test.txt مربوط به scrambler نام فایل sc_tb.v میباشد. در این فایل سید همان سیدی میباشد که در قبل گفته ایم و دیتای خروجی مربوط به متلب را به آن میدهیم. به ازای هر بار که دیتای خروجی با درست برابر باشد عبارت YES چاپ میشود و اگر همه آنها درست باشند عبارت The module is working correctly چاپ میشود و اگر دیتایی درست خروجی ندهد عبارت NO چاپ میشود و در آخر عبارت The module is not working correctlyچاپ میشود. شیوه کار این تست بنچ به این صورت میباشد که دیتای درست در رجیستر ans ذخیره میشود و دیتایی که میخواهیم به ماژول ورودی بدهیم در رجیستر data ذخیره میشود. ابتدا 7 صفر به ماژول داده میشود و سپس مقادیر دیتا به آن به ترتیب داده میشود و در رجیستر s_out ذخیره میشود و اگر s_out با ans برابر باشد جواب درست در نظر گرفته میشود.



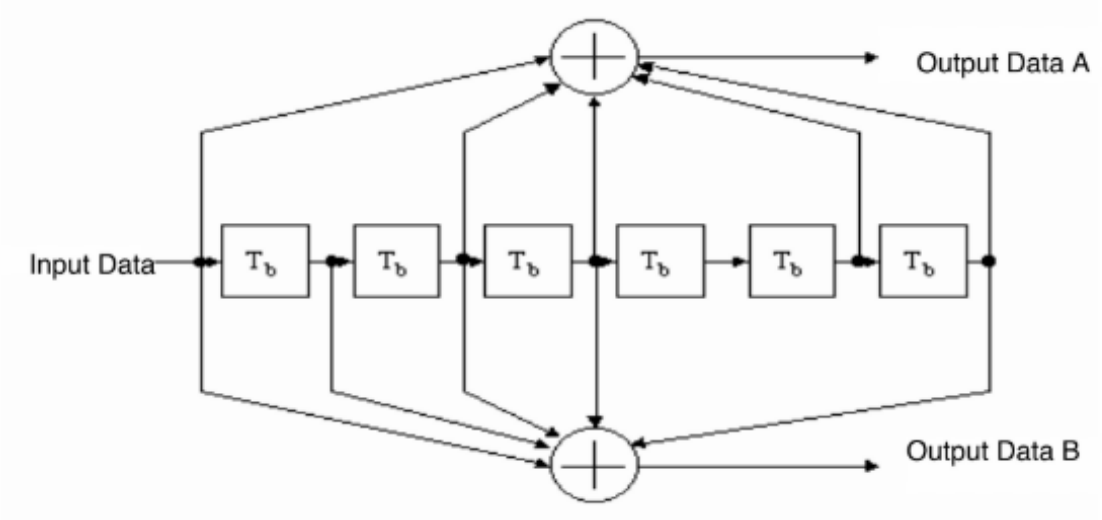
برای descrambler نیز فایل مربوط به تست دیتای test.txt به نام dsc_tb.v میباشد و تفاوت هایی که با ماژول بالا دارد این است که دیتاهای رجیستر های ans و data برعکس قرار میگیرند چون میخواهیم دیتای اسکرمبل شده را برگردانیم و هر بار نتیجه اسکرمبل شدن 7 صفر را به آن میدهیم که مقدار 7'b1100111 میباشد را به آن میدهیم و رجیستر seed نیز در این قسمت وجود ندارد. باقی قسمت ها مانند تست بنچ scrambler میباشد. نتیجه آن به صورت زیر میباشد:


ModelSim SE-64 10.5
File Edit View Compile Simulate Add Source Tools Layout Bookmarks Window Help
sim - Default
Instance Design unit Design unit type Top Category Visibility Total coverage
de_ttb de_ttb Module DU Instance +acc=...
uut descrambler Module DU Instance +acc=...
#INITIAL#15 de_ttb Process - +acc=...
#vsim_capacity# Capacity Statistics +acc=...
Objects
Name Value Kind Mode
clk 1'h0 Regi... Internal
data_file 32'h8... Inte... Internal
scan_file 32'hx... Inte... Internal
i 32'h0... Inte... Internal
j 32'h0... Inte... Internal
flag 32'h0... Inte... Internal
data 7'h67 Pack... Internal
s_out 7'h3c Pack... Internal
Processes (Active)
Name Type (filtered) State Order Parent Path
#INITIAL#15 Initial Active 1 /de_ttb
D:/WIFI/Scrambler/dtb.v (/de_ttb) - Default
$display("NO");
flag = 1;
end
end
if (flag)
$display("The module is not working correctly");
else
$display("The module is working correctly");
$stop;
end
descrambler uut(
.clk(clk),
.reset(reset),
.data_in(data_in),
.data_out(data_out)
);
Transcript
YES
YES
YES
YES
YES
YES
YES
YES
YES
YES
YES
YES
YES
The module is working correctly
** Note: $stop : D:/WIFI/Scrambler/dtb.v(67)
Time: 6080 ns Iteration: 0 Instance: /de_ttb
Break in Module de_ttb at D:/WIFI/Scrambler/dtb.v line 67
VSIM 6>

قسمت Convolutional Encoder:

انکودر

برای این قسمت پیچیدگی خاصی وجود ندارد این انکودر مانند ماژول داخل استاندارد به شکل زیر میباشد:



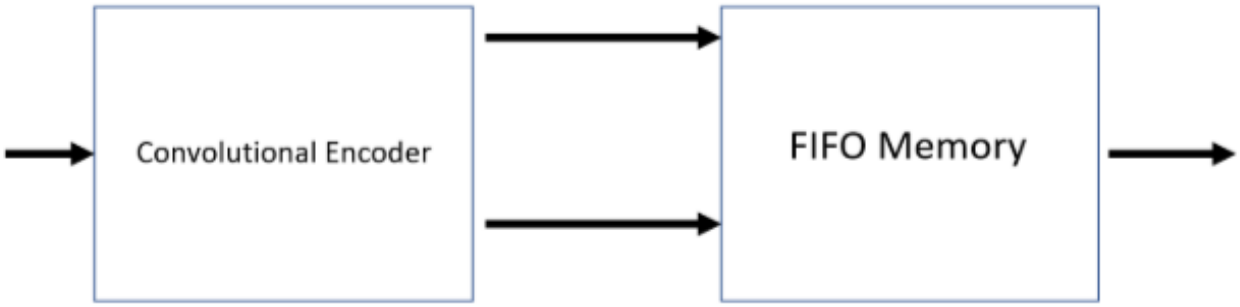
و برای نرخ کد 2/3 نیز کافی است به ازای هر 2 ورودی کد چهارم تولید شده را برداریم و در قسمت دیکودر آن را با 0 جایگزین کنیم.

دیکودر

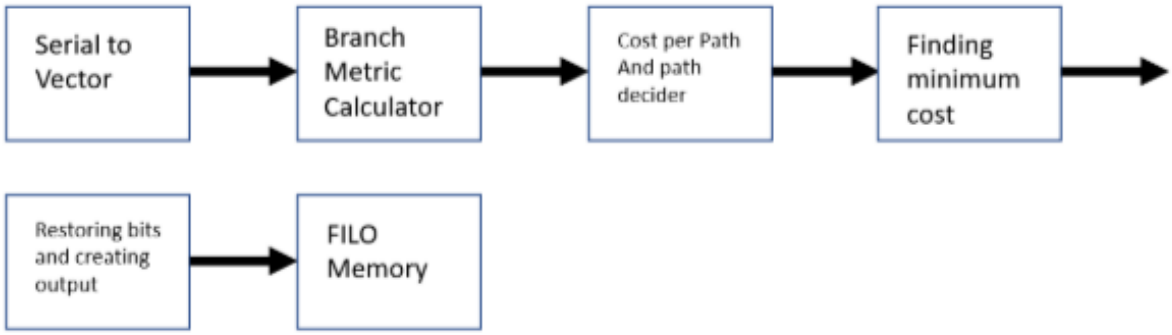
برای این قسمت از الگوریتم Viterbi استفاده میشود به این صورت که در این قسمت ما احتمال حالت داخل انکودر برای زمانی که 2 بیت وارد شده را داریم حساب میکنیم و در آخر مسیر با بیشترین احتمال را خروجی میدهیم. برای این کار در هر مرحله میدانیم زمانی که در یک حالت قرار داریم تنها ممکن است از دو حالت از کل 64 حالت موجود در رجیستر های انکودر به این حالت رسیده باشیم که یکی از آنها حالتی است که مقدار رجیستر آخر 0 باشد و دیگری این است که مقدار آن 1 باشد. هر کدام از این حالت ها کد های مختلفی تولید میکنند. ما فاصله همینگ این کدها با کد ورودی و فاصله کلی که تا آن حالت های موردنظر داشته ایم را حساب میکنیم و با توجه به آنها حالت با هزینه کمتر را نگه میداریم. در آخر با توجه به هزینه کلی همه مسیر ها بهترین مسیر را انتخاب میکنیم و خروجی میدهیم.

ساختار ها

ساختار انکودر تقریبا به شکل بالا میباشد با این تفاوت که حافظه ای در خروجی انکودر وجود دارد که ورودی ها را ذخیره میکند و به صورت FIFO خروجی میدهد. دلیل وجود این حافظه این است که به ازای هر ورودی ما حداقل دو خروجی داریم و برای این که بتوانیم به صورت سریال خروجی داشته باشیم باید این مقادیر در جایی ذخیره شوند.

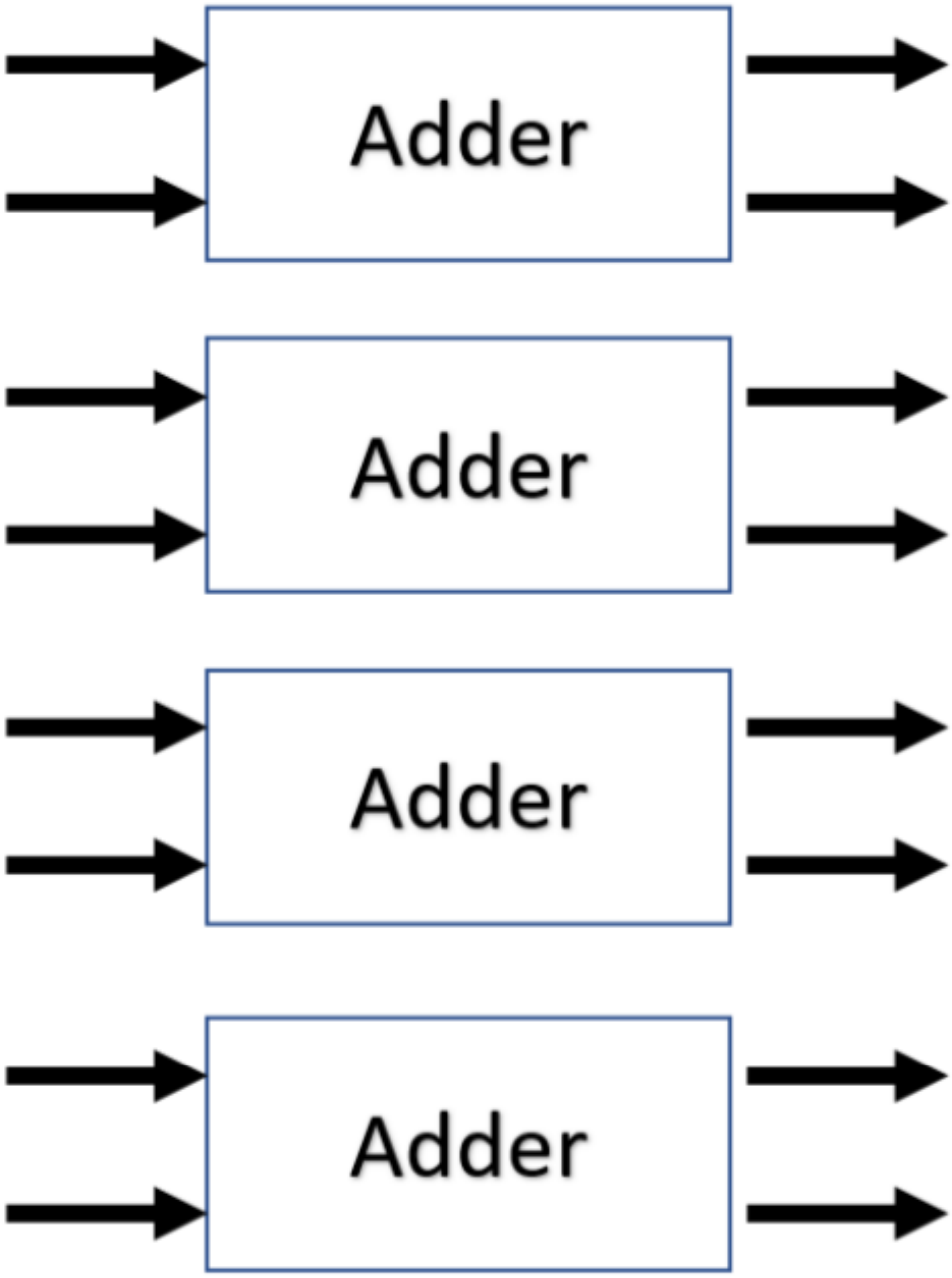


برای دیکودر به طور کلی ساختار زیر وجود دارد:

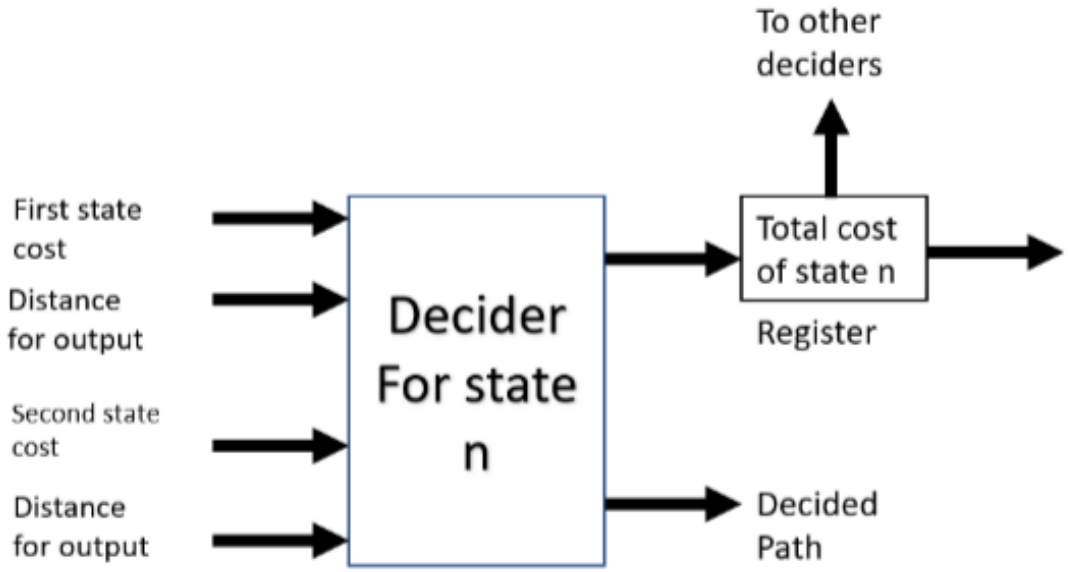


در بلاک اول ورودی های ما 2 تا 2 تا جدا میشوند که حالت های موردنظر ساخته شوند. این کار با توجه به نرخ کد انجام میشود.

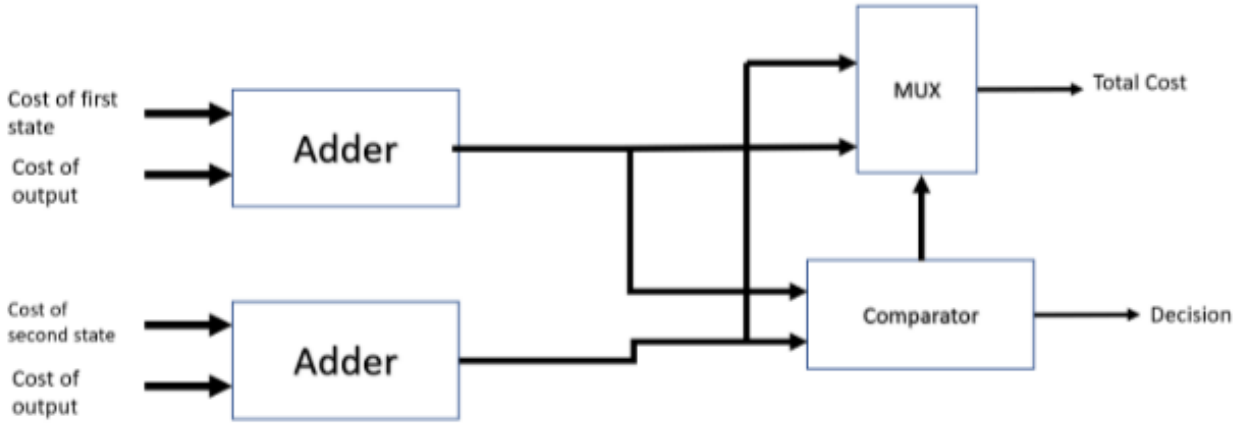
در بلاک دوم فاصله همه خروجی های ممکن انکودر با ورودی مقایسه میشوند. ساختار آن به صورت زیر میباشد:



برای محاسبه و تصمیم گیری برای مسیر های مختلف بلاک سوم به صورت زیر میباشد:



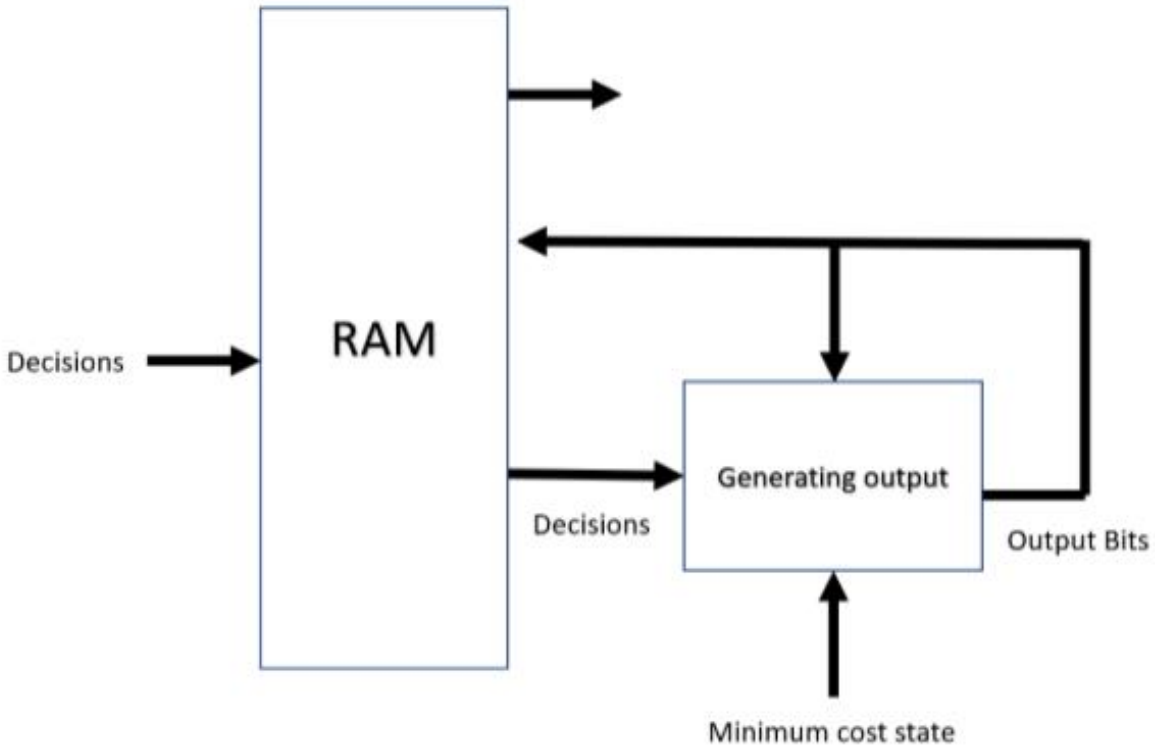
در اینجا حالت های اول و دوم همان حالت های ممکن هستند که در اول توضیح داده شد. در اینجا با توجه به فاصله خروجی آنها بین یکی از آن ها که هزینه کمتری خواهد داشت انتخاب خواهد شد. ساختار بلوک decider به شکل زیر میباشد:



توجه داشته باشید که برای هر حالت به یک عدد از این بلاک ها نیاز داریم تا بتوانیم برای هر کدام هزینه را حساب کنیم.

برای تشخیص کمترین هزینه بعد این که ورودی ها کامل شدند هر کدام را 2 به 2 مقایسه میکنیم و در آخر کمترین را به دست می آوریم که کدام بلاک میباشد. این کار در 6 کلاک انجام میشود و از روش های دیگر سریع تر میباشد.

برای قسمت آخر تشخیص مسیر نیز به شکل زیر ساختار را طراحی میکنیم:



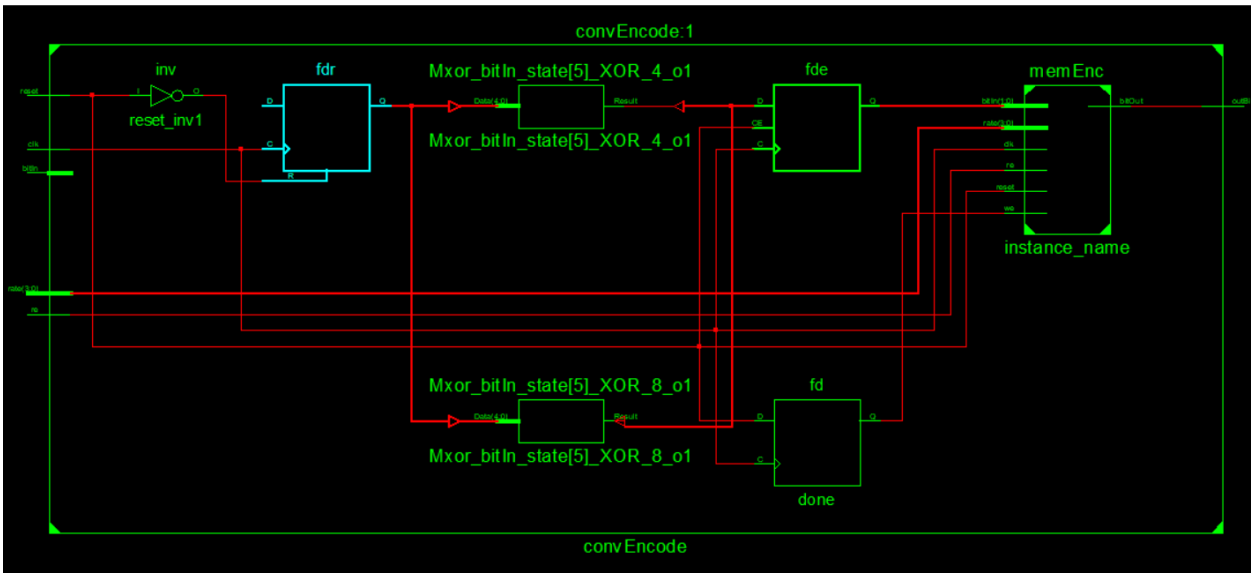
که بیت خروجی این بلاک از آخر به اول میباشد و برای همین نمیتوان مستقیم خروجی داد.

برای این که سرعت کار بالا برود میتوانیم هر بار بعد از تعداد محدودی بیت ورودی خروجی متناظر با آنها را محاسبه کنیم. قانون کلی برای این قسمت حداقل 5 برابر تعداد رجیستر ها در انکودر میباشد بنابراین در این قسمت نیز هر 32 بیت ورودی شروع به تولید خروجی متناظر با آنها میکند و در آخر همه خروجی را با هم به خروجی میدهد.

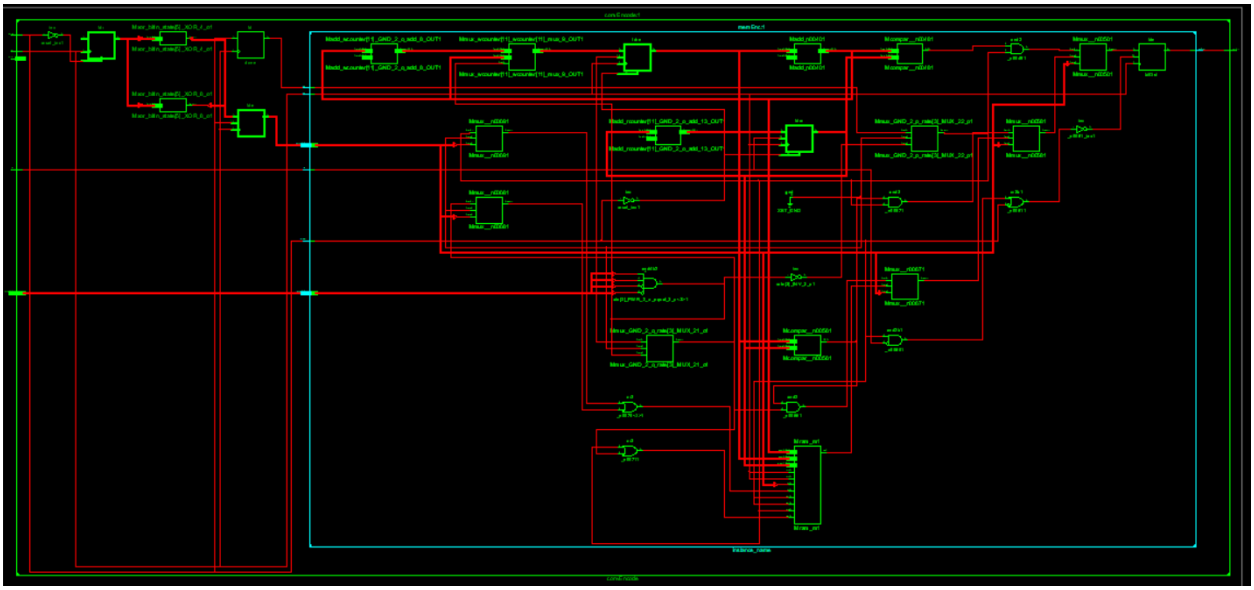
کد وریلاگ:

برای قسمت انکودر شامل 2 ماژول هست که یکی انکودر میباشد و دیگری حافظه ای میباشد که اطلاعات را داخل ان میریزیم.

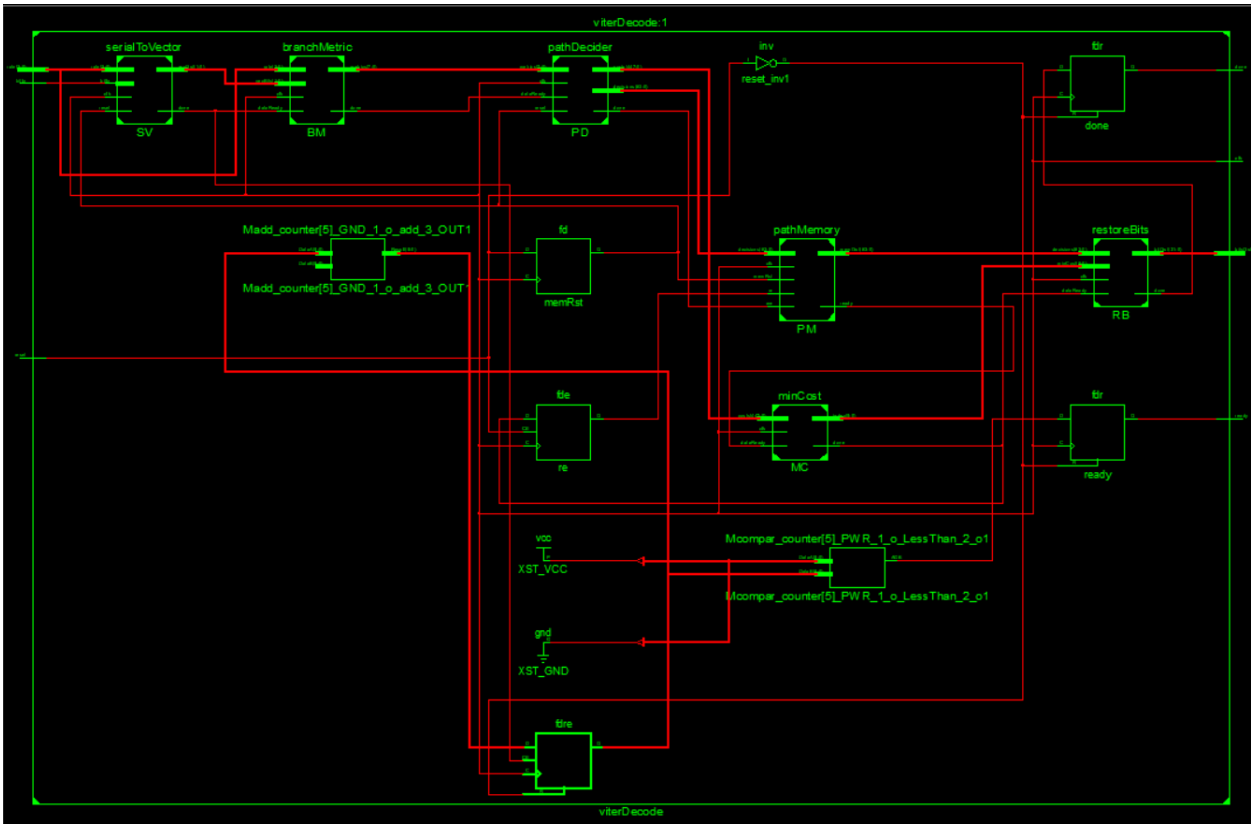
انکودر:



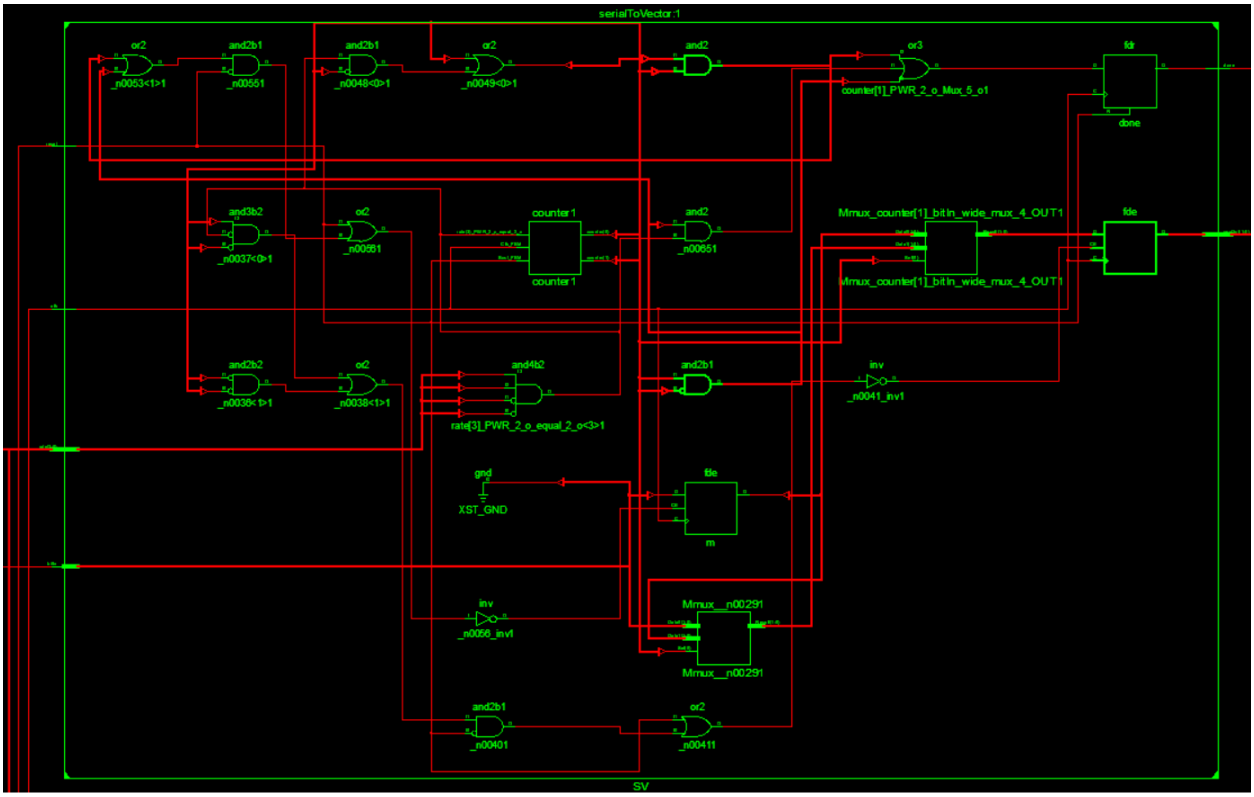
انکودر به همراه ممووری مربوط به انکودر:



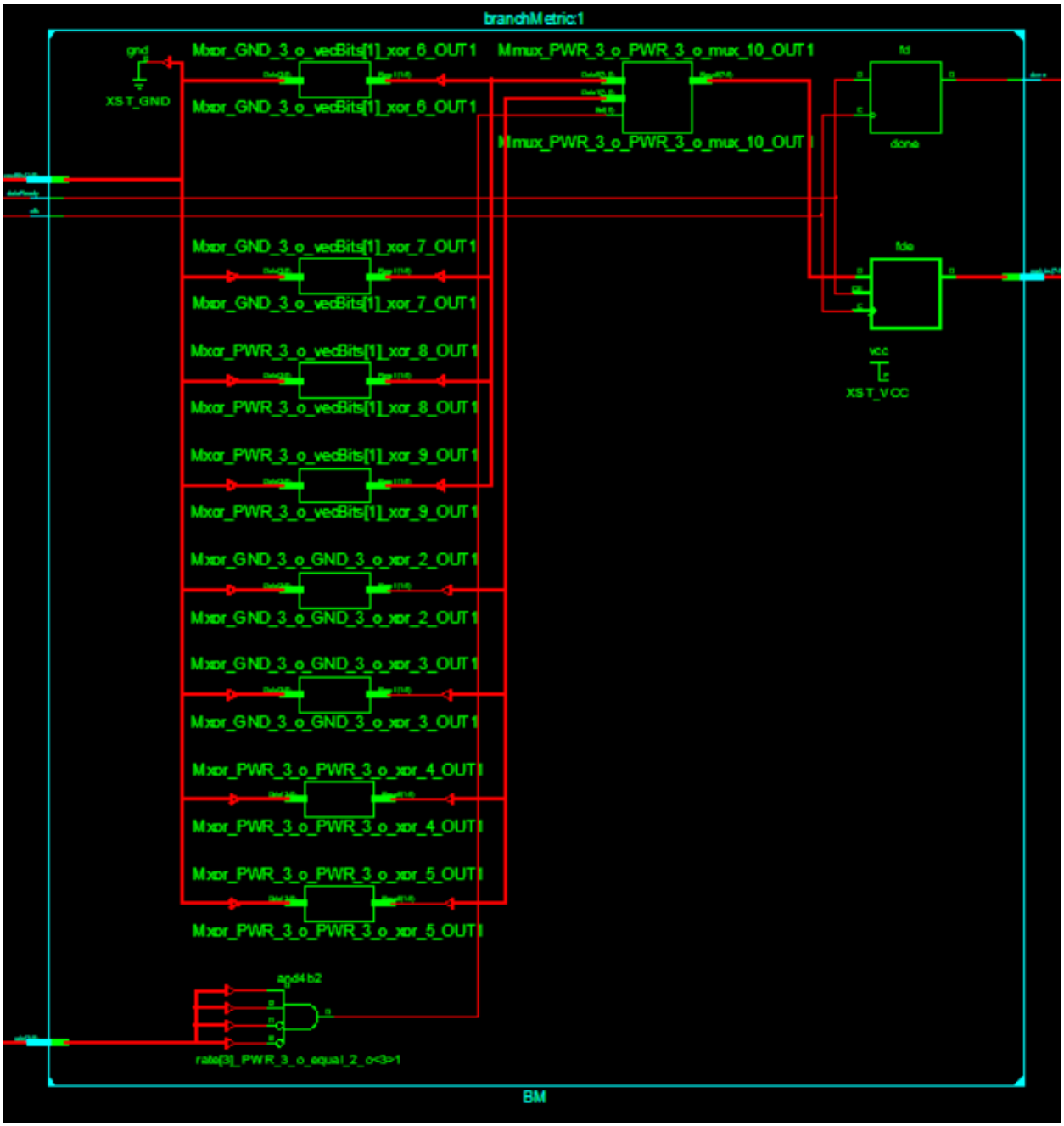
قسمت دیکودر دقیقا به همان شکلی میباشد که در قسمت تئوری گفته شده است:



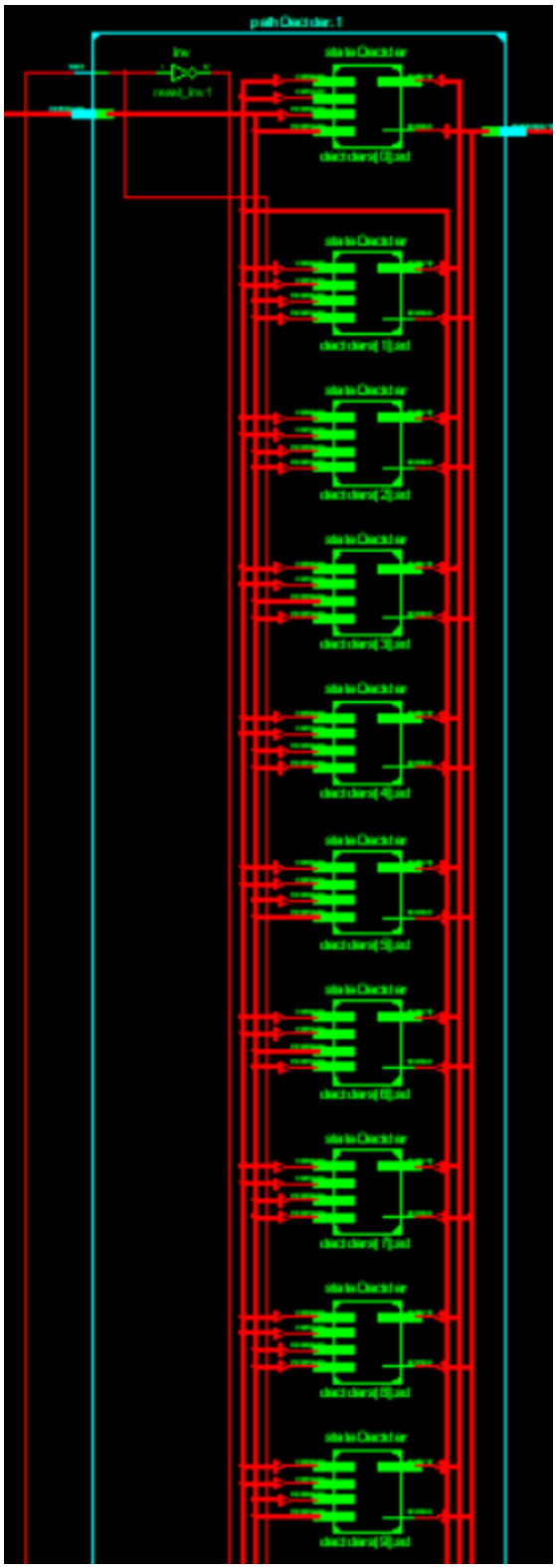
قسمت Serial to Vector :



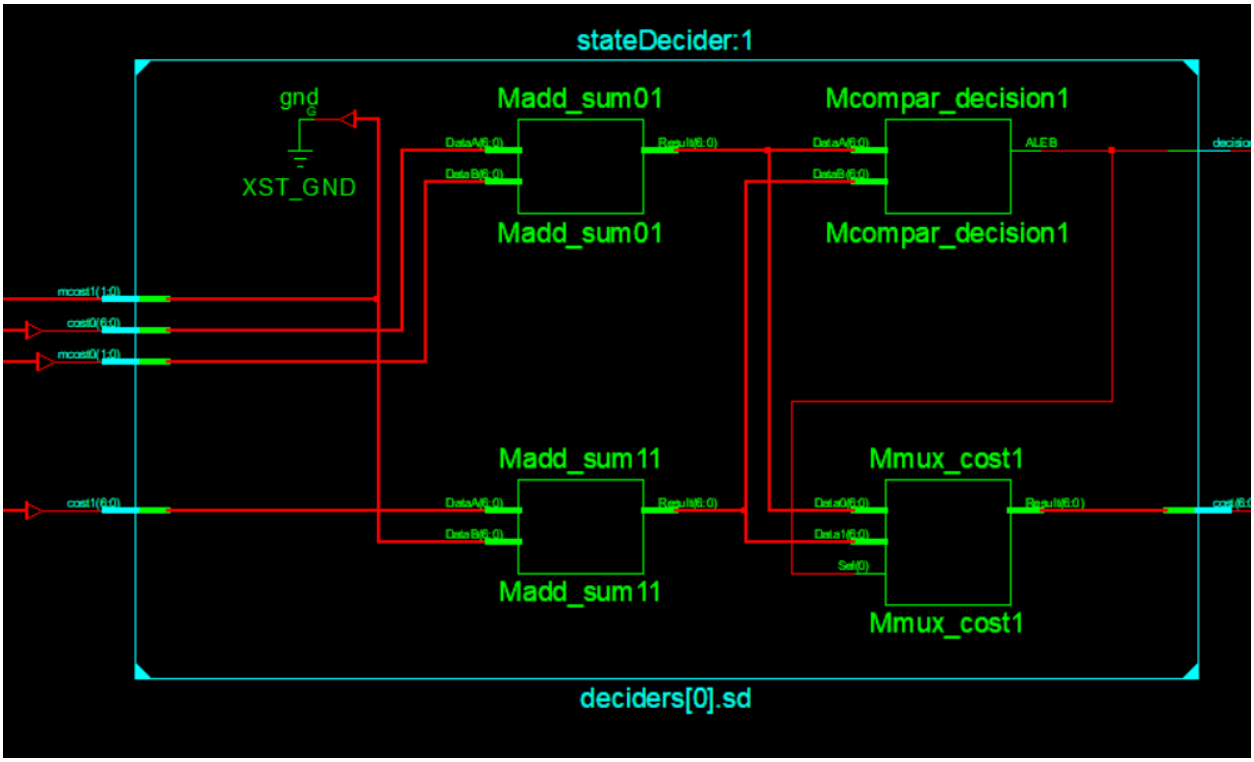
قسمت Branch Metrics:



قسمت Path Decider:

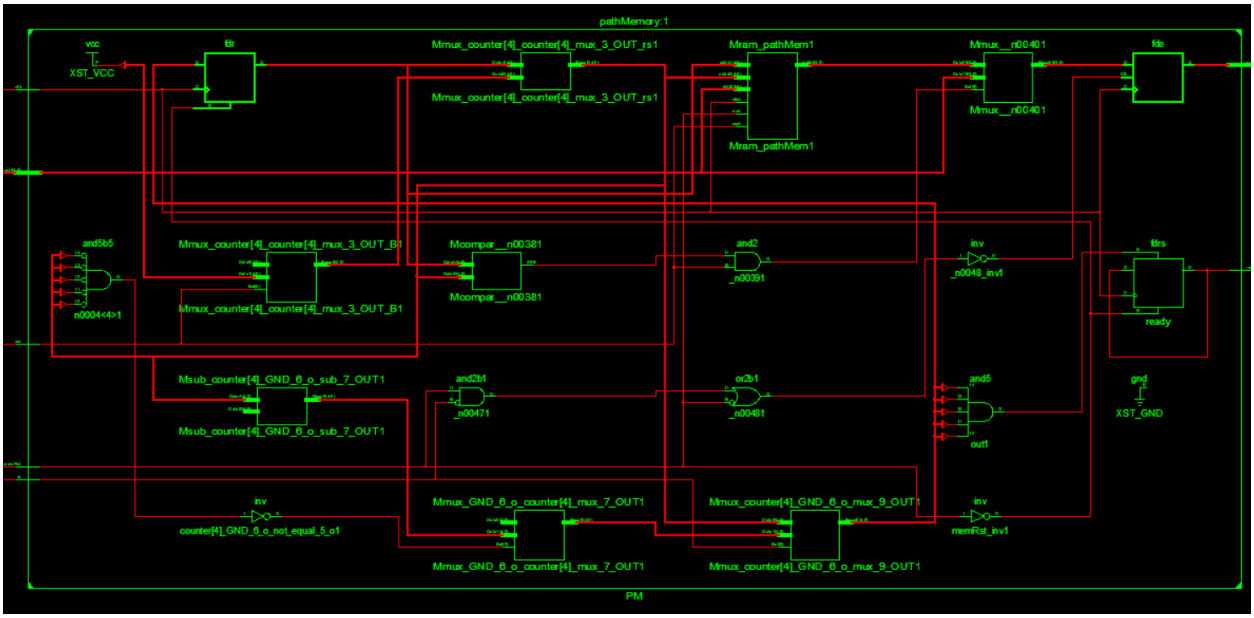


هر کدام از این بلاک ها همان بلاک های decider میباشند و محتویات داخل آنها به شکل زیر میباشد:

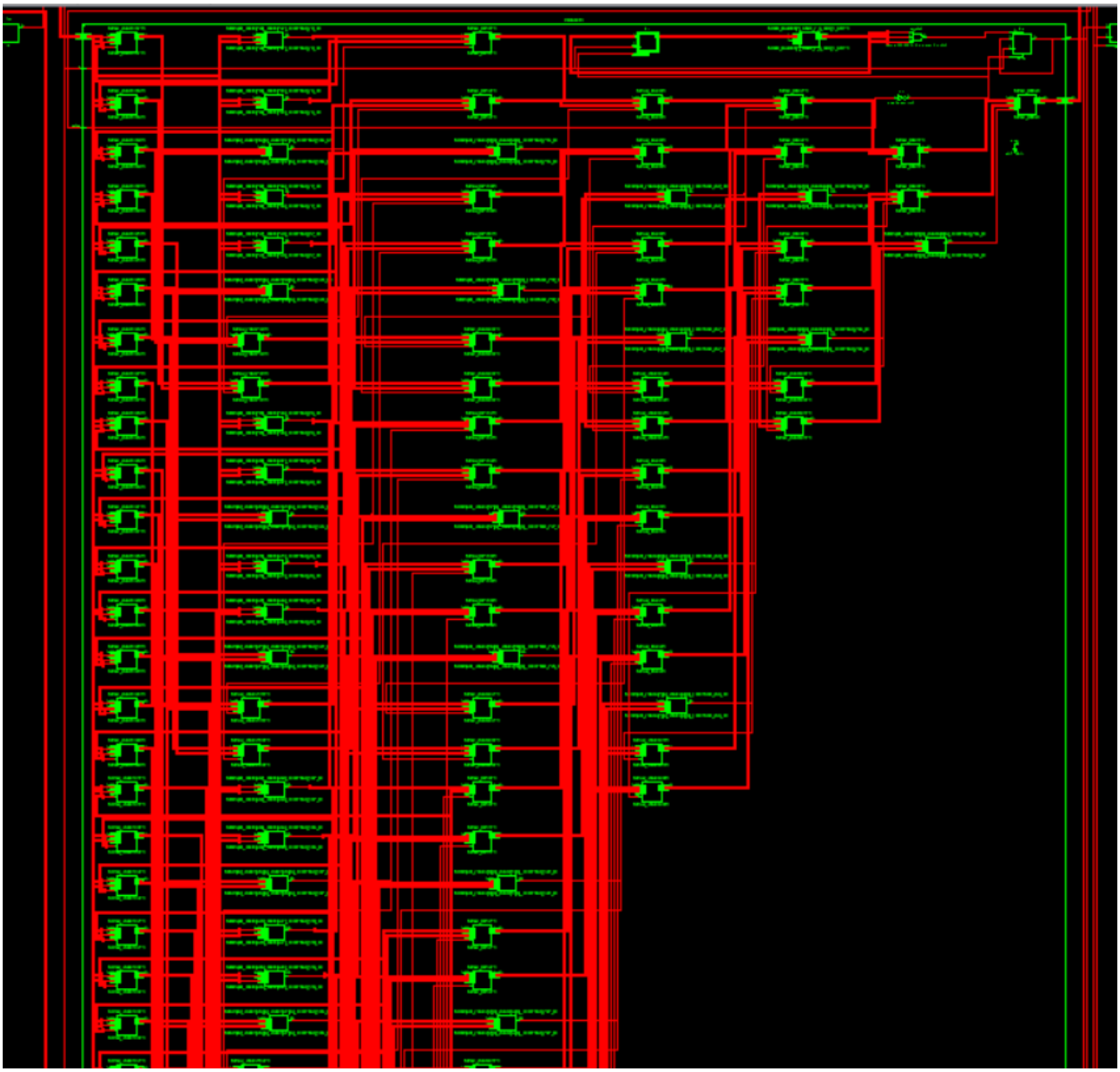


در مجموع 64 تا از این بلاک ها برای هر حالت رجیستر های داخل انکودر میباشد.

همه اطلاعات مربوط به بیت ها به داخل قسمت Path Memory میرود که به شکل زیر میباشد:

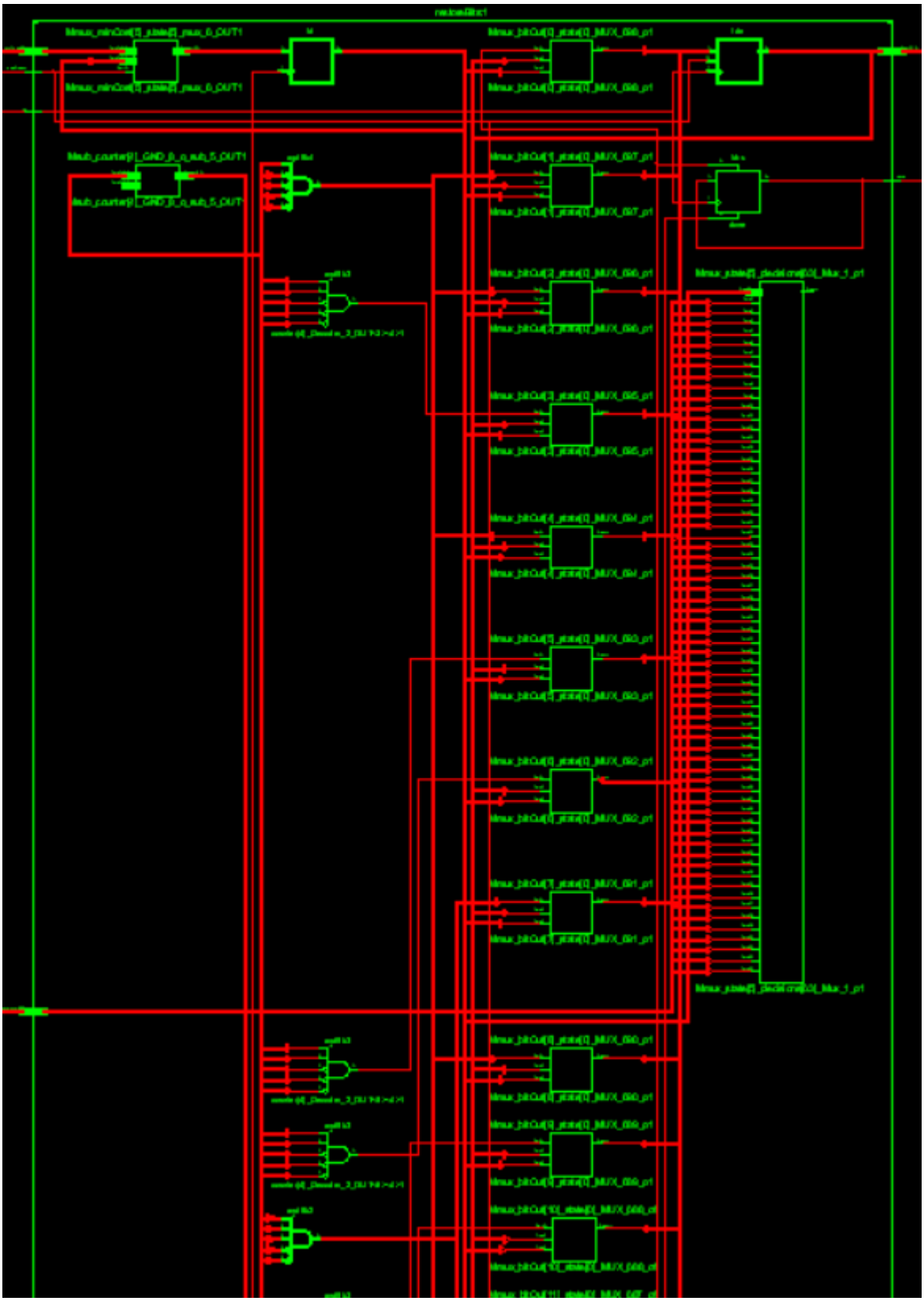


قسمت minCost شماره حالت با کمترین هزینه را در یک کلاک میتواند حساب کند:



هر کدام از این بلاک ها مقایسه کننده میباشند.

قسمت restore bits جایی هست که از اطلاعات minCost و pathMemory استفاده میکنیم و بیت های درست را دوباره میسازیم:



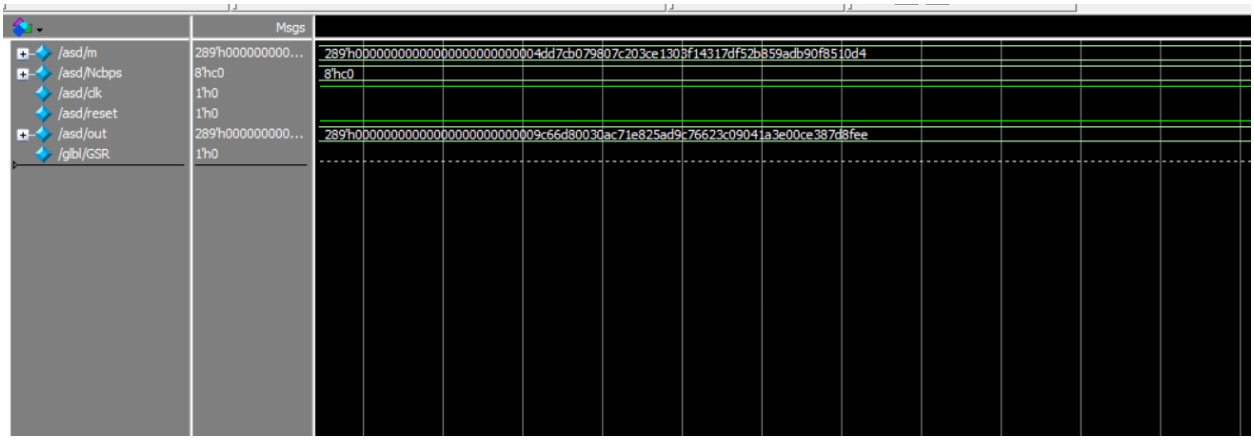
طریقه کلی کار دیکودر به این شکل میباشد که 64 بیت را میخواند و بعد از 36 کلاک بیت های واقعی را میسازد و بعد آن دوباره اجازه میدهد 64 بیت دیگر وارد آن شود.

قسمت interleaver:

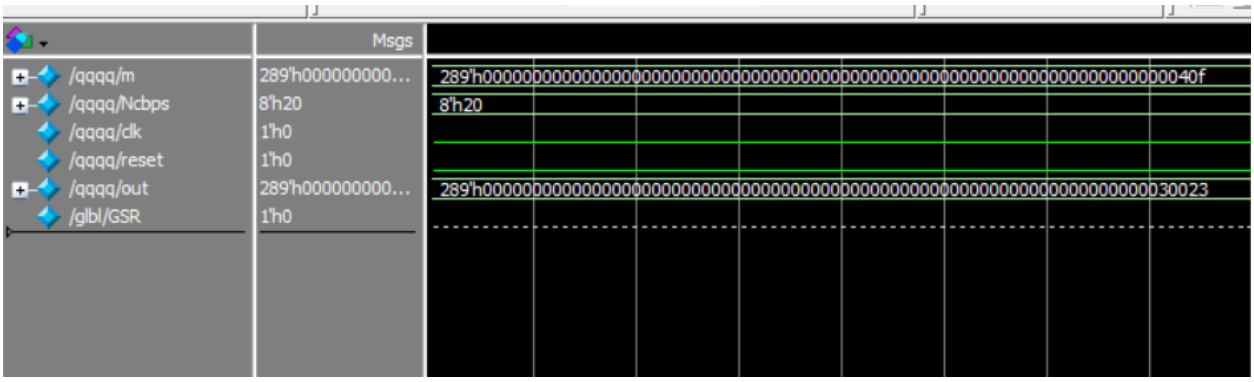
این قسمت دقیقا بر اساس فرمول هایی که در خود استاندارد گفته شده زده شده است و نکته خاصی ندارد.

به دلیل زمان بسیار زیادی که صرف سنتز این مدار میشد نتوانستم عکسی از سنتز آن داشته باشم.

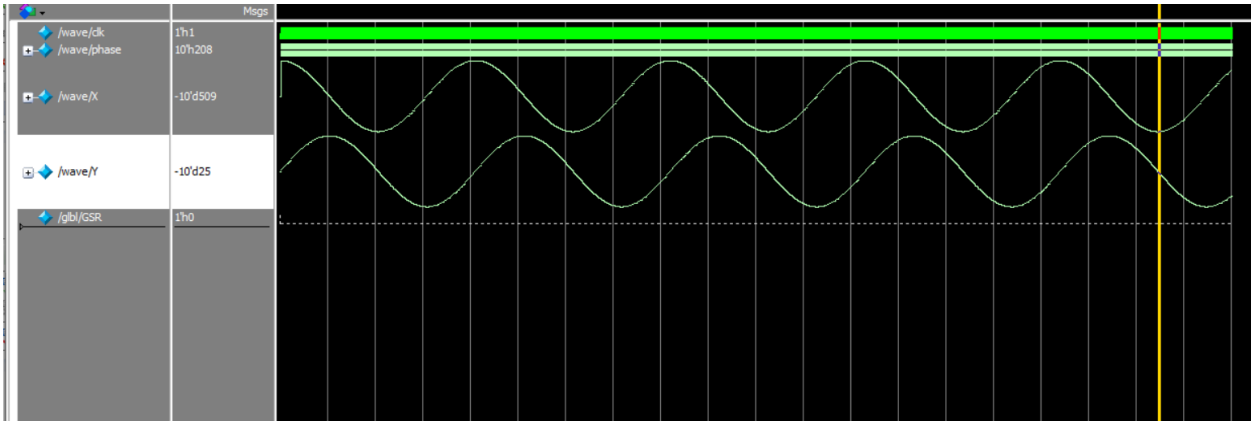
نمونه از شبیه سازی : (این مثال از خود ضمیمه استاندارد آورده شده است.)



یک نمونه کوچکتر از Deinterleaver:



برای قسمت ساختن موج های مثلثی به جای Cordic از DDS Compiler استفاده شده به دلیل اینکه استفاده از آن راحت تر میباشد و لیتنسی آن به جای 2، 1 میباشد. متاسفانه در این قسمت به این مشکل برخورد کردم که نمیدانستم که چه مدت زمان باید به ازای هر بیت خروجی مثلثی بدهم برای همین فقط شکل موج را در اینجا میگذارم:



لیست فایل ها:

- فایل های مربوط به Scrambler در پوشه Scrambler قرار دارد و در گزارش Scrambler اسم آنها مشخص شده است.
- فایل های Convolutional Encoder:

فایل ماژول branch metric: BranchMetric.v

فایل ماژول سریال به وکتور: serialToVector.v

فایل ماژول Path decider: PathDecider.v

فایل ماژول decider داخل Path decider: stateDecider.v

فایل ماژول pathMemory: pathMemory.v

فایل ماژول minCost: minCost.v

فایل ماژول restore bits: restoreBits.v

فایل ماژول Top layer برای دیکودر:viterDecoder.v

فایل ماژول موری انکودر:memEnc.v

فایل ماژول Top layer : convEncoder.v

- فایل های ماژول Interleaver

فایل interleaver: interleaver.v

فایل deinterleaver: deinterleaver.v

فایل تست بنچ interleaver: tb__interleaver.v

فایل تست بنچ deinterleaver: tb__deinterleaver.v

- فایل های wave generator:

فایل wave generator: waveGen.v

فایل تست بنچ wave generator: wave.v