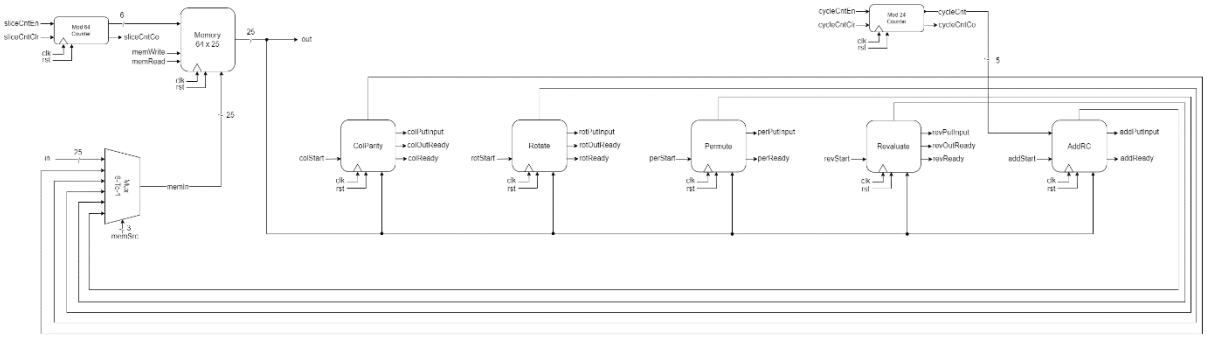
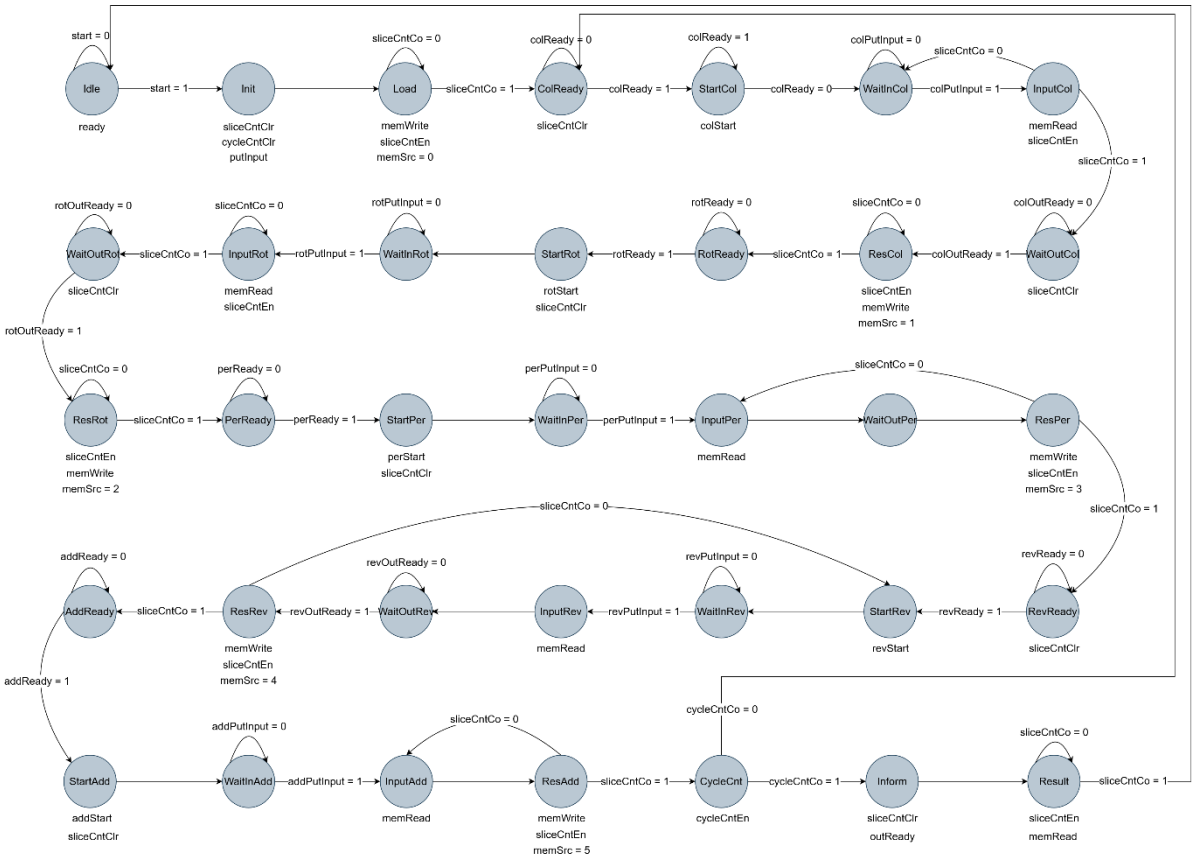
ماژول Encoder

این ماژول، ابتدا ماتریس ورودی را طی 64 کلاک دریافت کرده و در مموری خود ذخیره می‌کند. سپس 5 تابع را روی مموری به ترتیب اعمال کرده و این عمل را 24 بار تکرار می‌کند. در نهایت طی 64 کلاک مقادیر مموری خود را خروجی می‌دهد.

مسیر داده



واحد کنترل



ماژول ColParity

این ماژول در پروژه میانترم پیاده‌سازی شده و از همان در اینجا بدون تغییر استفاده شده است.
این تابع بر روی هر بیت ماتریس، به صورت زیر تعریف می‌شود:

$$matrix[i][j][k] = matrix[i][j][k] \oplus parity(A[i-1][0 \dots 4][k]) \oplus parity(A[i+1][0 \dots 4][k-1])$$

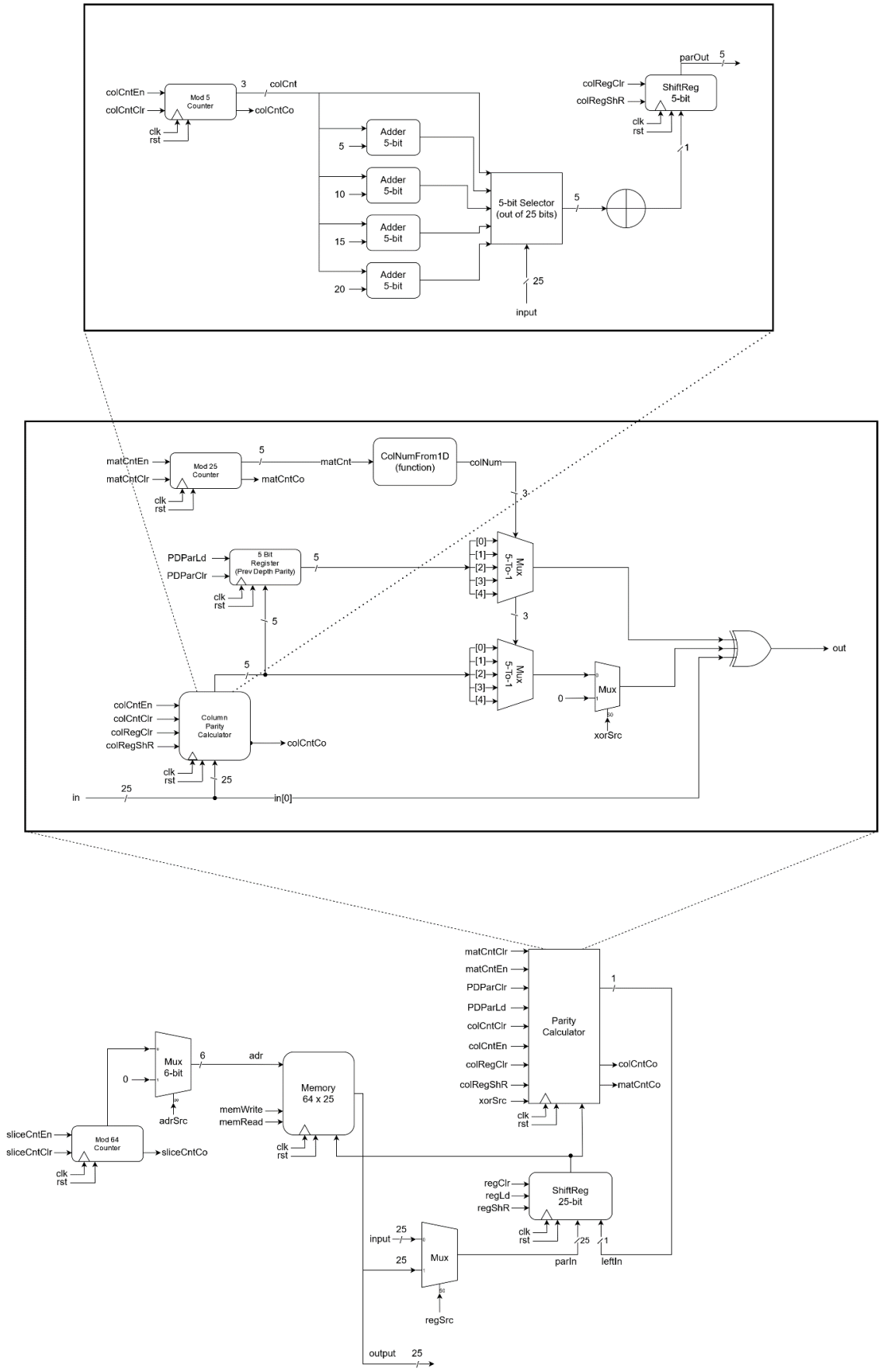
برای پیاده‌سازی این تابع که یعنی هر بیت ماتریس را برابر xor دو ستون سمت چپ، و ستون سمت راست در لایه قبل می‌کند، از دو ماژول داخلی ColumnParityCalculator و ParityCalculator استفاده می‌شود.

ماژول ColumnParityCalculator، یک اسلایس را گرفته و parity پنج ستون آن را محاسبه می‌کند. برای این کار هر بار با استفاده از چهار adder اندیس عناصر ستون به دست آمده، 5 اندیس از اسلایس انتخاب و وارد xor می‌شوند. نتیجه در یک shift register ذخیره شده و پس از محاسبه هر 5 ستون، خروجی داده می‌شود.

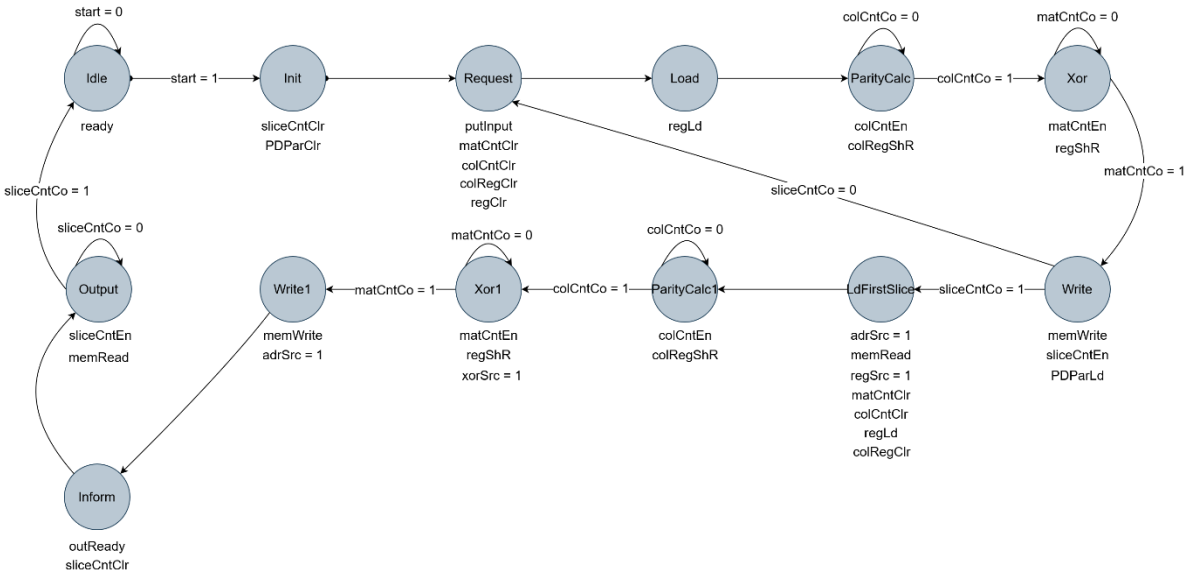
ماژول ParityCalculator ابتدا با استفاده از ماژول قبلی، parity ستون‌های اسلایس کنونی را محاسبه کرده و parity ستون‌های اسلایس قبلی را در یک رجیستر ذخیره نگه می‌دارد. سپس مقدار جدید بیت اول ورودی را طبق رابطه (که xor بیت اول، CurrColParity[i-1] و PrevColParity[i+1] است) خروجی می‌دهد.

ماژول اصلی، شامل یک مموری است که ابتدا طی 64 کلاک ورودی را دریافت کرده، و سپس هر اسلایس را وارد یک shift register می‌کند. این shift register هر بار وارد ParityCalculator شده و پس از محاسبه شدن مقدار جدید بیت اول آن، یک بار شیفت می‌خورد و مقدار جدید را وارد خود می‌کند.
این کار 25 بار انجام شده و هر بار مقدار جدید یک بیت وارد shift register می‌شود و از آنجا که شیفت می‌خورد، ParityCalculator که فقط مقدار جدید بیت اول را می‌دهد، حاصل تمام بیت‌ها را تولید می‌کند.
در نهایت مقدار نهایی shift register وارد مموری شده و پس از پایان محاسبات، مموری خروجی داده می‌شود.

مسیر داده


colCntEn
colCntClr
clk
rst
Mod 5 Counter
3
colCnt
colCntCo
Adder 5-bit
5
10
15
20
5-bit Selector (out of 25 bits)
25
input
5
colRegClr
colRegShR
ShiftReg 5-bit
parOut
1
matCntEn
matCntClr
Mod 25 Counter
matCntCo
matCnt
ColNumFrom1D (function)
colNum
PDParLd
PDParClr
5 Bit Register (Prev Depth Parity)
Mux 5-to-1
[0]
[1]
[2]
[3]
[4]
Mux
xorSrc
out
Column Parity Calculator
in
in[0]
Parity Calculator
matCntClr
matCntEn
PDParClr
PDParLd
colCntClr
colCntEn
colRegClr
colRegShR
xorSrc
colCntCo
matCntCo
Mux 6-bit
adr
adrSrc
Memory 64 x 25
memWrite
memRead
Mod 64 Counter
sliceCntEn
sliceCntClr
sliceCntCo
ShiftReg 25-bit
regClr
regLd
regShR
parIn
leftIn
regSrc
output

واحد کنترل

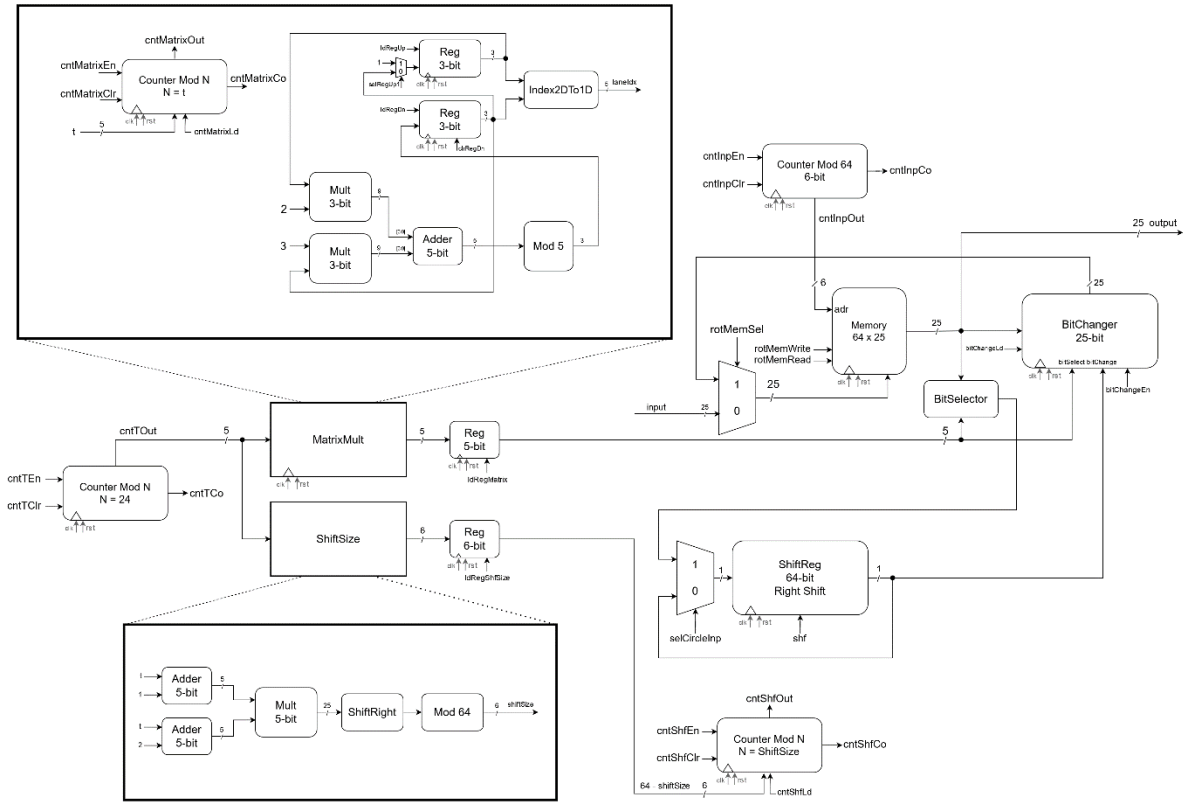

start = 0
Idle
ready
start = 1
Init
sliceCntClr
PDParClr
Request
putInput
matCntClr
colCntClr
colRegClr
regClr
Load
regLd
colCntCo = 0
ParityCalc
colCntEn
colRegShR
colCntCo = 1
matCntCo = 0
Xor
matCntEn
regShR
matCntCo = 1
sliceCntCo = 0
sliceCntCo = 1
Write
memWrite
sliceCntEn
PDParLd
sliceCntCo = 1
LdFirstSlice
adrSrc = 1
memRead
regSrc = 1
matCntClr
colCntClr
regLd
colRegClr
colCntCo = 0
ParityCalc1
colCntEn
colRegShR
colCntCo = 1
matCntCo = 0
Xor1
matCntEn
regShR
xorSrc = 1
matCntCo = 1
Write1
memWrite
adrSrc = 1
sliceCntCo = 0
Output
sliceCntEn
memRead
sliceCntCo = 1
Inform
outReady
sliceCntClr

ماژول Rotate

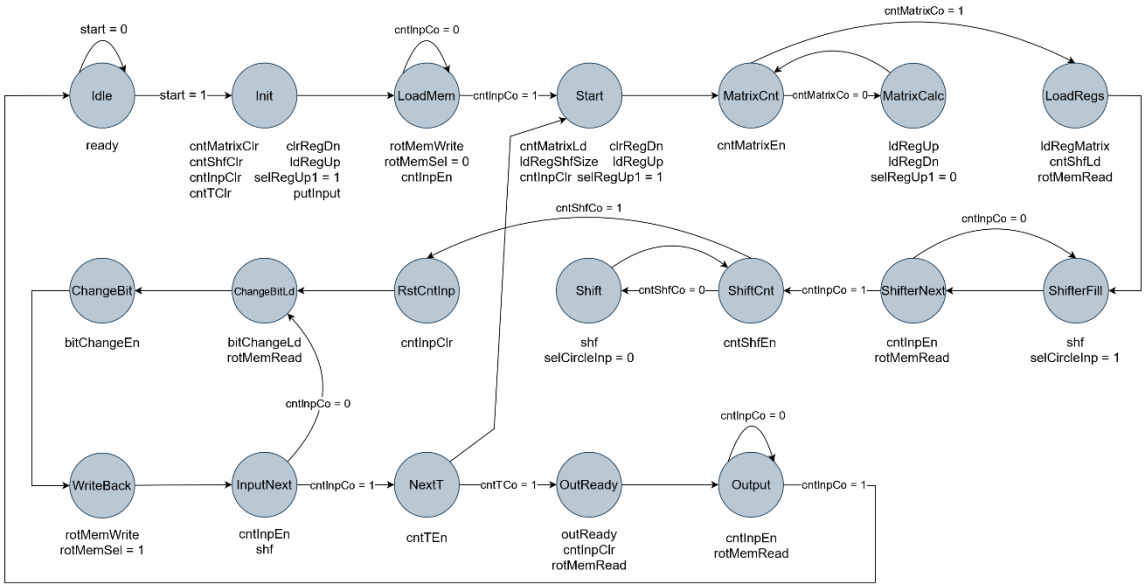
این ماژول هر lane ماتریس را مقدار خاصی شیفت می‌دهد. مقادیر hard code نشده اند و محاسبه می‌شوند. برای این کار دو ماژول MartixMult و ShiftSize داخل Rotate قرار دارند.
MatrixMult، عدد t را گرفته و ضرب ماتریسی ذکر شده در صورت سوال را برای به دست آوردن x و y انجام می‌دهد. سپس با استفاده از تابع Index2DTo1D، اندیس lane ای که باید شیفت بخورد را می‌گیرد.
ShiftSize، مداری combinational بوده که با گرفتن t، حاصل $\frac{(t+1)(t+2)}{2} \ mod \ 64$ را حساب می‌کند که مقدار شیفت کردن است.

این ماژول در ابتدا طی 64 کلاک ورودی را در مموری داخل خود ذخیره می‌کند.
حال اندیس lane ای که باید شیفت شود و مقدار شیفت به دست آورده می‌شود. این مقادیر وابسته به t اند که با شمارنده mod 24 به دست می‌آید.
سپس lane مورد نظر را طی 64 کلاک در یک shift register وارد می‌کند و به اندازه حساب شده ShiftSize آن را شیفت می‌دهد. پس از شیفت شدن، این مقادیر به مموری ماژول بر می‌گردند.
پس از تکرار این پروسه به ازای هر t، خروجی طی 64 کلاک از مموری داخلی به خارج فرستاده می‌شود.

مسیر داده



واحد کنترل


start = 0
Idle
ready
start = 1
Init
cntMatrixClr
cntShfClr
cntInpClr
cntTClr
clrRegDn
ldRegUp
selRegUp1 = 1
putInput
cntInpCo = 0
LoadMem
rotMemWrite
rotMemSel = 0
cntInpEn
cntInpCo = 1
Start
cntMatrixLd
ldRegShfSize
cntInpClr
clrRegDn
ldRegUp
selRegUp1 = 1
MatrixCnt
cntMatrixEn
cntMatrixCo = 0
MatrixCalc
ldRegUp
ldRegDn
selRegUp1 = 0
cntMatrixCo = 1
LoadRegs
ldRegMatrix
cntShfLd
rotMemRead
ShifterFill
shf
selCircleInp = 1
cntInpCo = 0
ShifterNext
cntInpEn
rotMemRead
cntInpCo = 1
ShiftCnt
cntShfEn
cntShfCo = 0
Shift
shf
selCircleInp = 0
cntShfCo = 1
RstCntInp
cntInpClr
ChangeBitLd
bitChangeLd
rotMemRead
ChangeBit
bitChangeEn
cntInpCo = 0
WriteBack
rotMemWrite
rotMemSel = 1
InputNext
cntInpEn
shf
cntInpCo = 1
NextT
cntTEn
cntTCo = 1
OutReady
outReady
cntInpClr
rotMemRead
Output
cntInpEn
rotMemRead
cntInpCo = 0
cntInpCo = 1

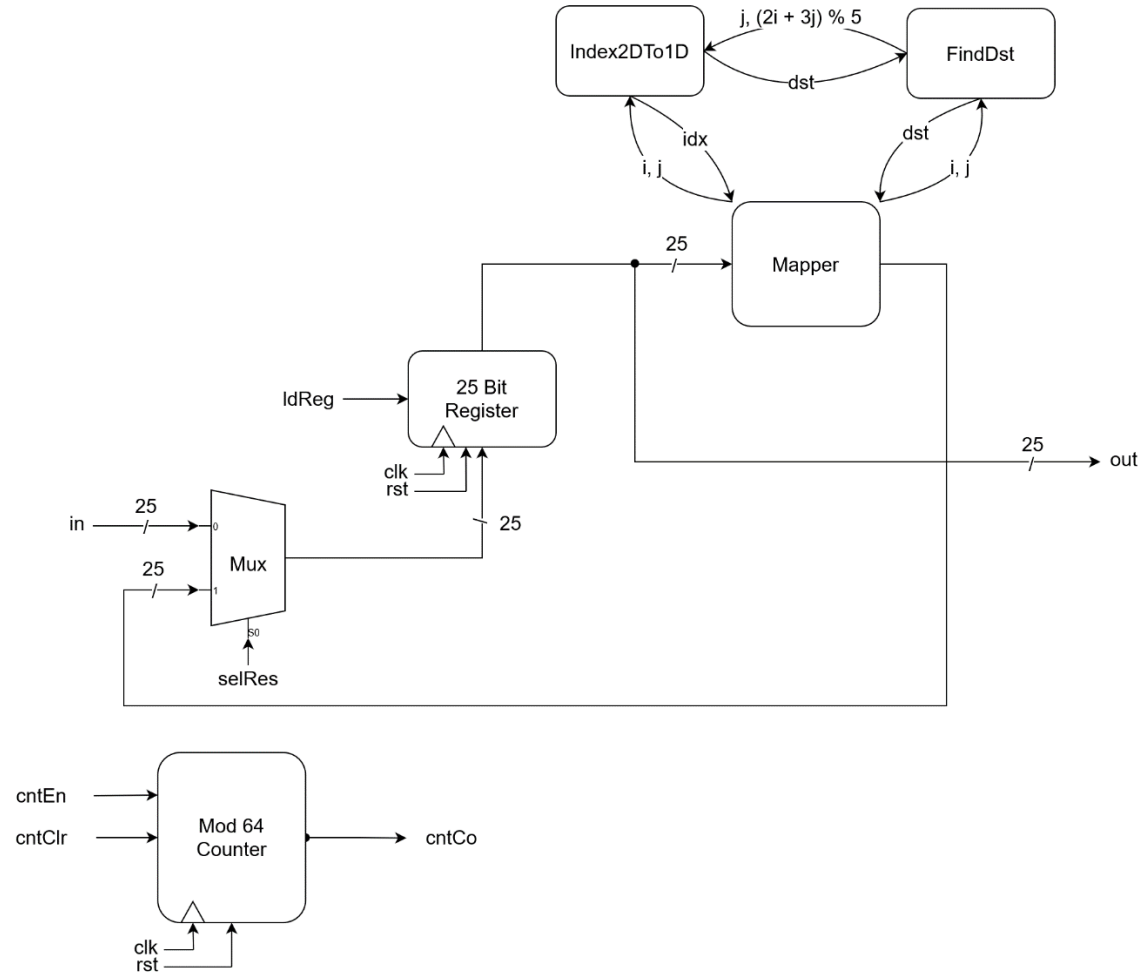
ماژول Permute

این ماژول در پروژه اول پیاده‌سازی شده و از همان در اینجا بدون تغییر استفاده شده است.
این تابع بر روی هر اسلایس ماتریس، نگاشت زیر را انجام می‌دهد:

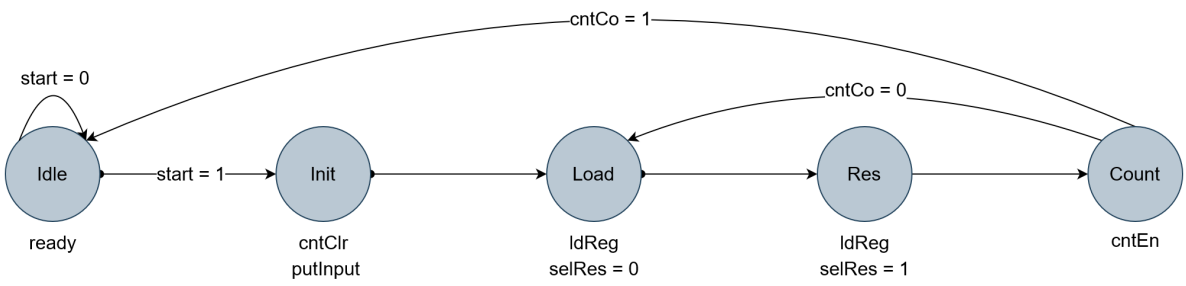
$$slice[j][(2i + 3j)\%5] = slice[i][j]$$

از یک رجیستر برای ورودی و خروجی اسلایس پس از انجام نگاشت استفاده شده است.
ماژول داخلی Mapper، کار نگاشت را انجام داده که از دو تابع Index2DTo1D برای تبدیل خانه ماتریس به اندیس خطی، و FindDst برای یافتن مقصد هر خانه پس از نگاشت استفاده می‌کند.

مسیر داده



واحد کنترل



ماژول Revaluate

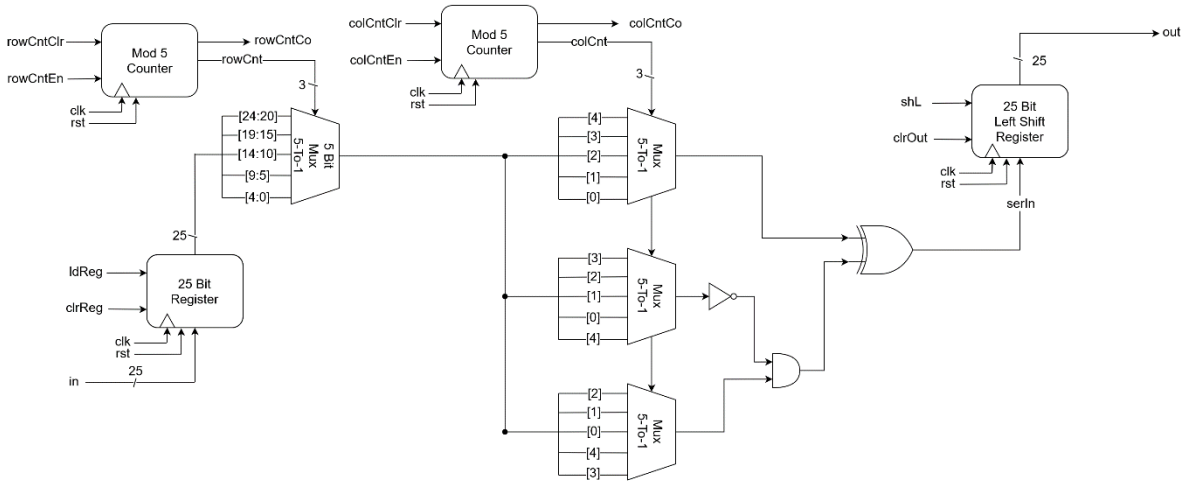
این ماژول هر بیت ماتریس را به صورت زیر تغییر می‌دهد:

$$A[x,y,z] = A[x,y,z] \oplus (\sim A[x+1,y,z] \,\&\, A[x+2,y,z])$$

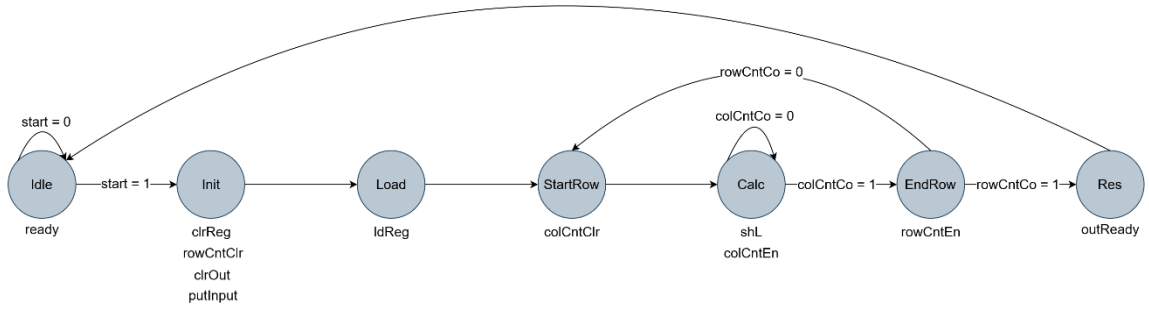
برای پیاده‌سازی این تابع غیر خطی، هر ردیف از اسلایس‌های ماتریس جدا در نظر گرفته شده و با استفاده از دو شمارنده (یکی برای ردیف‌ها و دیگری برای هر خانه از آن ردیف) پیمایش می‌شود.

مقدار محاسبه شده تابع وارد یک shift register شده که در نهایت تمام مقادیر جدید اسلایس ماتریس را خواهد داشت.

مسیر داده



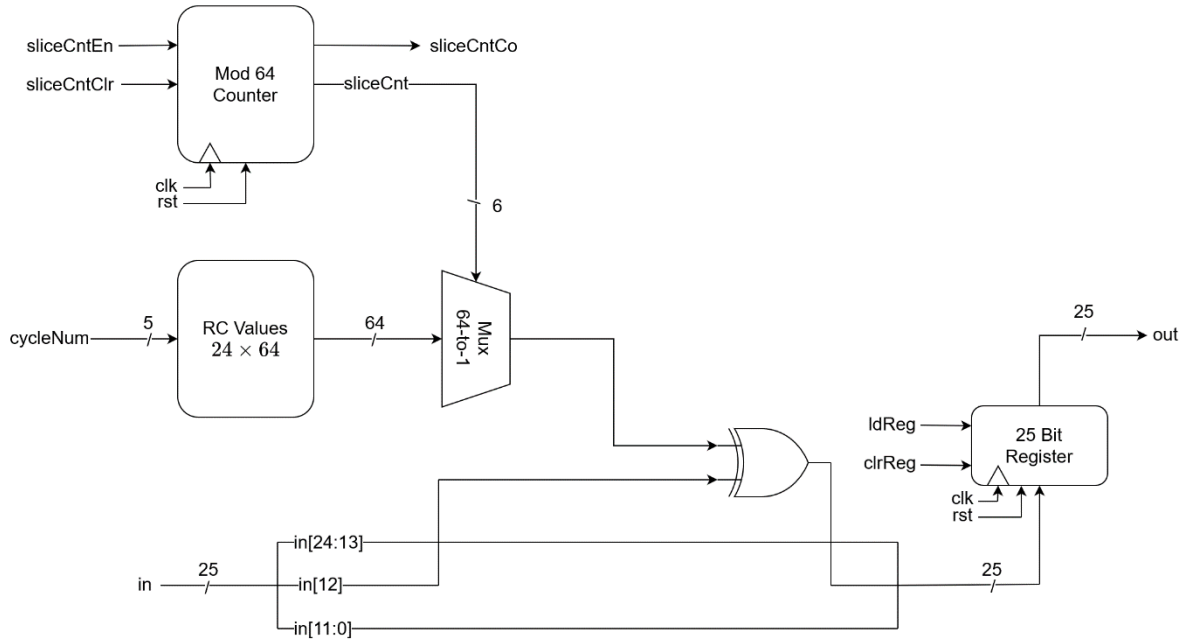
واحد کنترل



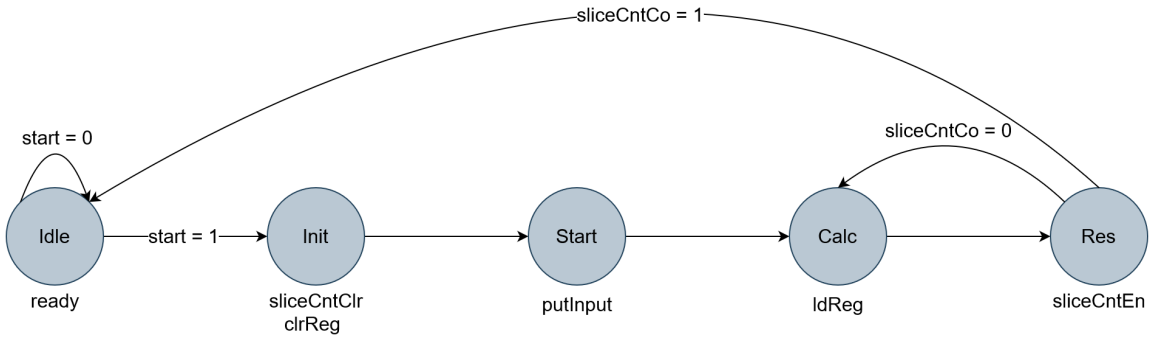
ماژول AddRC

این ماژول lane وسط ماتریس یعنی خانه (0, 0) را با مقدار RC متناظر با دور cycleNum-ام ماژول تاپ-لول xor می‌کند.
مقادیر RC در فایل rc.hex ذخیره شده و توسط مموری این ماژول خوانده می‌شود.
این ماژول 25 بیت یک slice ماتریس را دریافت کرده و خروجی تغییر یافته را در رجیستر خروجی ذخیره می‌کند.
این کار تا گرفتن 64 ورودی انجام می‌شود.

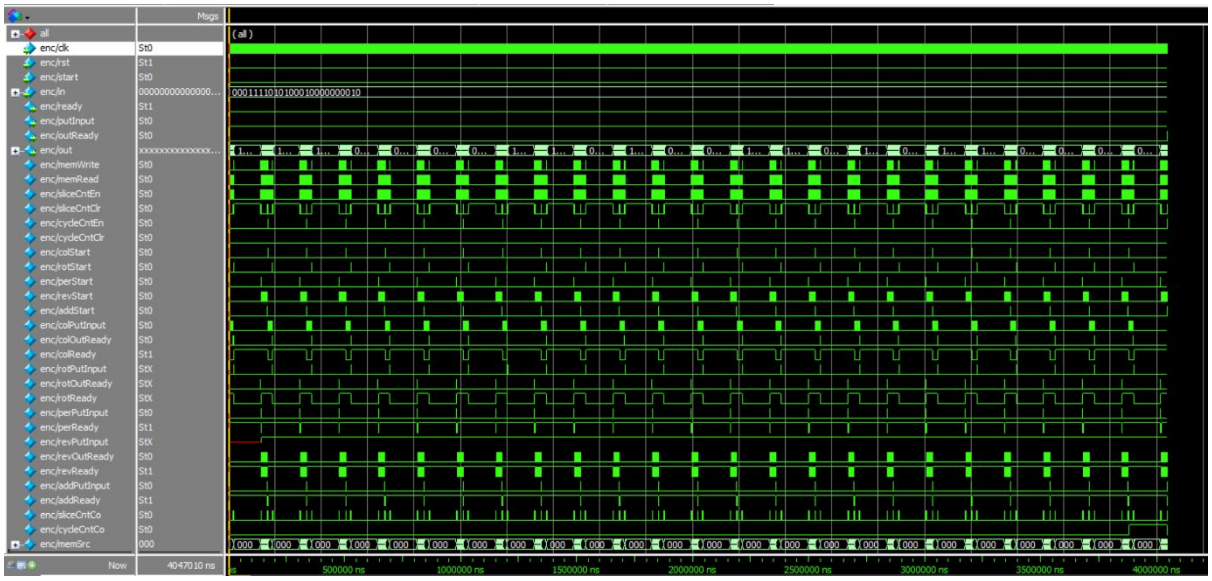
مسیر داده



واحد کنترل



نمونه اجرا


Msgs
all
enc/clk
enc/rst
enc/start
enc/in
enc/ready
enc/putInput
enc/outReady
enc/out
enc/memWrite
enc/memRead
enc/sliceCntEn
enc/sliceCntClr
enc/cycleCntEn
enc/cycleCntClr
enc/colStart
enc/rotStart
enc/perStart
enc/revStart
enc/addStart
enc/colPutInput
enc/colOutReady
enc/colReady
enc/rotPutInput
enc/rotOutReady
enc/rotReady
enc/perPutInput
enc/perReady
enc/revPutInput
enc/revOutReady
enc/revReady
enc/addPutInput
enc/addReady
enc/sliceCntCo
enc/cycleCntCo
enc/memSrc
Now
500000 ns
1000000 ns
1500000 ns
2000000 ns
2500000 ns
3000000 ns
3500000 ns
4000000 ns