ΕΡΓΑΣΤΗΡΙΟ 4

20/04/2021

ΙΕΡΑΡΧΙΚΗ ΣΧΕΔΙΑΣΗ – ΔΗΜΙΟΥΡΓΙΑ ΓΡΑΦΟΥ , ΕΥΡΕΣΗ BIPARTITE ΓΡΑΦΟΥ ΚΑΙ ΔΗΜΙΟΥΡΓΙΑ ΔΥΟ DATASET ΑΠΟ ΕΝΑ

Ντουνέτας Δημήτρης
ΑΜ: 2016030141

Εισαγωγή

Σκοπός της δεύτερης άσκησης είναι η δημιουργία ενός προγράμματος για παραγωγή δύο αρχείων από το επίπεδο Dataset που παράγει το lab3. Τα δύο αρχεία αυτά είναι χωρισμένα με τέτοιο τρόπο ώστε η ένωση τους να μας δίνει το αρχείο εισόδου χωρίς να χάνεται πληροφορία. Σκοπός είναι να κοπεί το αρχείο με τέτοιο τρόπο ώστε τα καλώδια που κανονικά θα ήταν εσωτερικά και δε θα μας ενδιέφεραν ως είσοδοι ή έξοδοι, αν χρειάζεται να βγαίνουν ως έξοδοι από το πρώτο κομμάτι και ταυτόχρονα να γίνονται είσοδοι στο δεύτερο. Για την υλοποίηση της άσκησης λήφθηκε υπόψιν η παραδοχή που χρησιμοποιούμε και στα προηγούμενα εργαστήρια σχετικά με CMOS ώστε να πηγαίνει το ρεύμα από τη τάση στη γείωση.

Λειτουργία του Προγράμματος

Ο τρόπος λειτουργίας του προγράμματος είναι αρκετά απλός. Ξεκινάει διαβάζοντας το αρχείο εισόδου γραμμή προς γραμμή κάνοντας χρήση της συνάρτησης getline() και σπάει το αρχείο σε κομμάτια με χρήση της συνάρτησης strtok(). Όλα τα χρήσιμα μέρη αποθηκεύονται σε μεταβλητές για να χρησιμοποιηθούν αργότερα. Δημιουργούνται διάφοροι πίνακες που μας βοηθούν να κόψουμε σωστά το επίπεδο Dataset και να διαλέξουμε τα σήματα που μας ενδιαφέρουν. Στο τέλος μας δίνει τα 2 αρχεία που είναι κομμένα και η ένωση τους μας δίνει το αρχικό.

Ανάλυση του Κώδικα

Αρχικοποιήσεις συναρτήσεων και μεταβλητών:

```
//Lab4 Functions
void findPossibleSplitPoints();
void verifySplitPoints();
void splitFile();
void createInputsOutputs(int splitingPoint);

//Functions used
void checkStim(int argc, char *argv[]);//checks if a stim file is entered
void getCircuitinfo(FILE* fpIn,FILE* fpOut);//gets the info from the stim file

//Memory allocations for Circuit Data.
int VCC,GND,numOfInputs,numOfOutputs,netlistRows,netlistCols,numOfTestIn,numOfTestOut,numOfGraphs,numOfSimulations,correctSims,numOfInputsFirst,numOfInputsSecond,numOfOutputsFirst,
int numOfNodes = 0;
int* inputs;
int* outputs;
int* inputsFirst;
int* inputsSecond;
int* outputsFirst;
int* outputsSecond;
int **netlist;
int count,count1;
```

Η main συνάρτηση:

```
int main(int argc, char *argv[])
{
    //Check if circuit file is present
    checkStim(argc, argv);
    //open circuit file
    FILE* fpIn = fopen(argv[1],"r");
    if (fpIn == NULL)
    {
        printf("Could not find file");
    }
    char file1[64];
    char file2[64];
    char dot = '.';
    // strcat(fileOut,argv[1]);
    strcpy(file1,strtok(argv[1],&dot));
    strcpy(file2,strtok(argv[1],&dot));
    strcat(file1,"_out1.txt");
    strcat(file2,"_out2.txt");

    FILE *fp1 = fopen(file1,"w");
    FILE *fp2 = fopen(file2,"w");
    //get circuit file info
    getCircuitinfo(fpIn);

    crete2Files(fp1,fp2);
    return 0;
}
```

Ψήγμα κώδικα από της συνάρτηση getCircuitInfo():

```
//Find Testbench values save them and do test
if(strcmp(line, "## TESTBENCH")==0)
{
    numOfSimulations=0;
    correctSims=0;
    // doTestBench(fpIn,fpOut);
    findPossibleSplitPoints();
    verifySplitPoints();
    splitFile();
    createInputsOutputs();
}
```

Η συνάρτηση getCircuitInfo() δουλεύει με τον ίδιο τρόπο όπως στα προηγούμενα εργαστήρια , δηλαδή αποθηκεύει δεδομένα από το αρχείο εισόδου, εκτός από την περίπτωση του TESTBENCH όπου καλεί τις καινούργιες συναρτήσεις του εργαστηρίου 4.

Ψήγμα κώδικα από τη συνάρτηση findPossibleSplitPoints():

```
void findPossibleSplitPoints()
{
    // Find nodes that their gates are not source or drain above
    for (int i = 1; i < netlistRows; i++)
    {
        for (int j = 0; j < i; j++)
        {
            if ((netlist[i][1]==netlist[j][2])||(netlist[i][1]==netlist[j][3]))
            {
                netlist[i][4]=1;
            }
        }
    }
}
```

Η συγκεκριμένη συνάρτηση διαβάζει όλους τους κόμβους και τους μαρκάρει αν βρει έναν κόμβο που η πύλη του ανήκει σε Source ή Drain παραπάνω. Αυτή η συνθήκη είναι ικανή αλλά όχι αναγκαία και ουσιαστικά χρησιμοποιεί το γεγονός ότι ένα σημείο που πιθανώς να κόψουμε θα πηγαίνει από πολλά source ή drain σε ένα gate.

Από αυτή τη συνάρτηση βγαίνει ένας πίνακας με τη παρακάτω μορφή:

PMOS/ NMOS	GATE	SOURCE/DRAIN	DRAIN/SOURCE	Possible SplitPoint
0/1	Gate node	Source node/Drain Node	Drain node/Source node	0/1

Αν είναι πιθανό splitPoint τότε βάζουμε 1 αλλιώς 0.

Ψήγμα κώδικα από τη συνάρτηση verifySplitPoints():

```
void verifySplitPoints()
{
    for (int i = 1; i < netlistRows-1; i++)
    {
        bool foundBefore = false;
        bool foundAfter = false;
        if (netlist[i][4]==1)
        {
            //Check if gate node is present in a Source or Drain Before current node row
            for (int j = 0; j < i; j++)
            {
                if (((netlist[j][2]==netlist[i][2])||(netlist[j][3]==netlist[i][2]))&&((netlist[i][2]!=VCC)&&(netlist[i][2]!=GND)))
                {
                    foundBefore = true;
                    break;
                }
                if (((netlist[j][2]==netlist[i][3])||(netlist[j][3]==netlist[i][3]))&&((netlist[i][3]!=VCC)&&(netlist[i][3]!=GND)))
                {
                    foundBefore = true;
                    break;
                }
            }
//Check if gate node is present in a Source or Drain After current node row
for (int j = i+1; j < netlistRows; j++)
{
    if (((netlist[j][2]==netlist[i][2])||(netlist[j][3]==netlist[i][2]))&&((netlist[i][2]!=VCC)&&(netlist[i][2]!=GND)))
    {
        foundAfter = true;
        break;
    }
    if (((netlist[j][2]==netlist[i][3])||(netlist[j][3]==netlist[i][3]))&&((netlist[i][3]!=VCC)&&(netlist[i][3]!=GND)))
    {
        foundAfter = true;
        break;
    }
if (!foundBefore && foundAfter)
{
    netlist[i][5] = 1;
}
else if (!foundAfter && foundBefore)
{
    netlist[i][5] = -1;
}
else
{
    netlist[i][4] = 0;
}
```

Η συγκεκριμένη συνάρτηση διαβάζει όλους τους κόμβους που έχουμε μαρκάρει ως πιθανά σημεία κοψίματος και επαληθεύει αν είναι όντως σωστά καθώς και αν πρέπει να κόψουμε από πάνω ή από κάτω από αυτό τον κόμβο.

Παίρνει έναν έναν τους κόμβους και ελέγχει αν ένας μαρκαρισμένος κόμβος ανήκει σε Source ή Drain πιο πάνω στο αρχείο ή αν αυτός ο κόμβος ανήκει σε Source ή Drain πιο κάτω στο αρχείο.

Αν ένας κόμβος ανήκει σε Source ή Drain και πιο πάνω και πιο κάτω ουσιαστικά δεν είναι σημείο κοψίματος και απορρίπτεται.

Αν ανήκει μόνο πάνω τότε μαρκάρεται με ένα -1 και σημαίνει ότι ο κόμβος μπορεί να κοπεί από εκεί και κάτω. Αν ανήκει μόνο κάτω τότε μαρκάρεται με ένα 1 και σημαίνει ότι ο κόμβος μπορεί να κοπεί από εκεί και πάνω.

Ο Πίνακας μετά την συνάρτηση verifySplitPoints() έχει τη μορφή:

PMOS/ NMOS	GATE	SOURCE/ DRAIN	DRAIN/SOURCE	Possible SplitPoint	UP/DOWN
0/1	Gate node	Source node/Drain Node	Drain node/Source node	0/1	1/-1

Ψήγμα κώδικα από τη συνάρτηση splitFile():

```
void splitFile()
{
    int halfPoint = 0, splitingPointOffset1=0, splitingPointOffset2=0;
    printf("Number of Transistors %d\n",netlistRows);
    // We want to cut the file about in the middle so we start from there.
    halfPoint = netlistRows/2;
    // Check if the halfpoint is a spliting point.
    if (netlist[halfPoint][4]==1&&netlist[halfPoint][5]==-1)
    {
        splitingPoint = halfPoint;
    }
    else if (netlist[halfPoint][4]==1&&netlist[halfPoint][5]==1)
    {
        splitingPoint = halfPoint-1;
    }
    // Move the spliting point up and down and find the poitn of the minimum distance
    else
    {
        for (int i = halfPoint; i >= 0 ; i--)
        {
            if (netlist[i][4]==1&&netlist[i][5]==-1)
            {
                break;
            }
            if (netlist[i][4]==1&&netlist[i][5]==1)
            {
                splitingPointOffset1--;
                break;
            }

            splitingPointOffset1--;
        }

        for (int j = halfPoint; j < netlistRows; j++)
        {
            if (netlist[j][4]==1&&netlist[j][5]==1)
            {
                splitingPointOffset2--;
                break;
            }
            if (netlist[j][4]==1&&netlist[j][5]==-1)
            {
                break;
            }
            splitingPointOffset2++;
        }
        // Find the nearest spliting point
        if (abs(splitingPointOffset1)<=abs(splitingPointOffset2))
        {
            splitingPoint = halfPoint + splitingPointOffset1;
        }
        else
        {
            splitingPoint = halfPoint + splitingPointOffset2;
        }

        printf("\nSplitting point = %d",splitingPoint);
    }
    // Print Splited netlist
    printf("\nNETLIST IS : (where 0 in first column = PMOS and 1 = NMOS)\n");
    for (int i = 0; i <  netlistRows; i++)
    {

        for (int j = 0; j < netlistCols+2; j++)
        {
            printf("%d ",netlist[i][j]);
        }
        if (i==splitingPoint)
        {
            printf("\n");
        }
        printf("\n");
    }
```

Η συγκεκριμένη συνάρτηση εκτελεί το κόψιμο του αρχείου σε 2 νέα. Επειδή θέλουμε τα δύο αρχεία που θα παράξουμε να είναι όσο το δυνατόν πιο συμμετρικά και ισοδύναμα ξεκινάμε την αναζήτηση του σημείου που θα κόψουμε από τη μέση. Έτσι ελέγχουμε αν ο μεσαίο κόμβος είναι σημείο κοψίματος και αν είναι κόβουμε κατάλληλα πάνω ή κάτω. Αλλιώς ελέγχουμε μια-μια θέσεις πάνω και κάτω και διαλέγουμε το σημείο που είναι πιο κοντά στο μεσαίο σημείο.

Ψήγμα κώδικα από τη συνάρτηση createInputsOutputs):

```
void createInputsOutputs()
{
    // Find first's files inputs
    // Find for every gate that its node doesnt't exist as a source or drain in the first part
    for (int i = 0; i < splitingPoint; i++)
    {
        bool nodeExists = false;
        for (int j = 0; j < splitingPoint; j++)
        {
            if ((netlist[i][1]==netlist[j][2])||(netlist[i][1]==netlist[j][3]))
            {
                nodeExists = true;
                break;
            }
        }
        // Dont add duplicates
        if (!nodeExists)
        {
            //check for duplicates
            bool duplicate = false;
            for (int k = 0; k < numOfInputsFirst; k++)
            {
                duplicate = false;
                if (netlist[i][1]==inputsFirst[k])
                {
                    duplicate = true;
                    break;
                }
            }
            // If not a duplicate then add it
            if (!duplicate)
            {
                numOfInputsFirst++;
                if (numOfInputsFirst==1)
                {
                    inputsFirst = (int*)malloc((sizeof(int)*numOfInputsFirst)); //Allocate memory
                    inputsFirst[numOfInputsFirst-1]=netlist[i][1];
                }
                else
                {
                    inputsFirst = (int*)realloc(inputsFirst,(sizeof(int)*numOfInputsFirst)); //Reallocate memory
                    inputsFirst[numOfInputsFirst-1]=netlist[i][1];
                }
            }
        }
    }
```

```
    // Find seconds's files inputs
    // Find for every gate that node exists as a source or drain in the first part
    for (int i = splitingPoint+1; i < netlistRows; i++)
    {
        bool nodeExists = false;
        for (int j = splitingPoint+1; j < netlistRows; j++)
        {
            if ((netlist[i][1]==netlist[j][2])||(netlist[i][1]==netlist[j][3]))
            {
                nodeExists = true;
                break;
            }
        }
        // Dont add duplicates
        if (!nodeExists)
        {
            //check for duplicates
            bool duplicate = false;
            for (int k = 0; k < numOfInputsSecond; k++)
            {
                duplicate = false;
                if (netlist[i][1]==inputsSecond[k])
                {
                    duplicate = true;
                    break;
                }
            }
            // If not a duplicate then add it
            if (!duplicate)
            {
                numOfInputsSecond++;
                if (numOfInputsSecond==1)
                {
                    inputsSecond = (int*)malloc((sizeof(int)*numOfInputsSecond)); //Allocate memory
                    inputsSecond[numOfInputsSecond-1]=netlist[i][1];
                }
                else
                {
                    inputsSecond = (int*)realloc(inputsSecond,(sizeof(int)*numOfInputsSecond)); //Reallocate memory
                    inputsSecond[numOfInputsSecond-1]=netlist[i][1];
                }
            }
        }
    }
}
```

```
//Find First's Outputs
for (int i = 0; i < splitingPoint; i++)
{
    for (int j = 0; j < numOfOutputs; j++)
    {
        if ((netlist[i][2]==outputs[j]) || (netlist[i][3]==outputs[j]))
        {
            //check for duplicates
            bool duplicate = false;
            for (int k = 0; k < numOfOutputsFirst; k++)
            {
                duplicate = false;
                if (outputs[j]==outputsFirst[k])
                {
                    duplicate = true;
                break;
                }
            }
            // If not a duplicate then add it
            if (!duplicate)
            {
                numOfOutputsFirst++;
                if (numOfOutputsFirst==1)
                {
                    outputsFirst = (int*)malloc((sizeof(int)*numOfOutputsFirst)); //Allocate memory
                    outputsFirst[numOfOutputsFirst-1]=outputs[j];
                }
                else
                {
                    outputsFirst = (int*)realloc(outputsFirst,(sizeof(int)*numOfOutputsFirst)); //Reallocate memory
                    outputsFirst[numOfOutputsFirst-1]=outputs[j];
                }
            }
        }
    }
}
```

```
// Add all the inputs of the second part if they are not original inputs, without duplicates
for (int i = 0; i < numOfInputsSecond; i++)
{
    //check ig original input
    bool originalInput = false;
    for (int m = 0; m < numOfInputs; m++)
    {
        if (inputsSecond[i]==inputs[m])
        {
            originalInput = true;
            break;
        }
    }
    // if original input move to the next
    if (originalInput)
    {
        continue;
    }
    //check for duplicates
    bool duplicate = false;
            for (int k = 0; k < numOfOutputsFirst; k++)
            {
                duplicate = false;
                if (inputsSecond[i]==outputsFirst[k])
                {
                    duplicate = true;
                break;
                }
            }
            // If not a duplicate then add it
            if (!duplicate)
            {
                numOfOutputsFirst++;
                if (numOfOutputsFirst==1)
                {
                    outputsFirst = (int*)malloc((sizeof(int)*numOfOutputsFirst)); //Allocate memory
                    outputsFirst[numOfOutputsFirst-1]=inputsSecond[i];
                }
                else
                {
                    outputsFirst = (int*)realloc(outputsFirst,(sizeof(int)*numOfOutputsFirst)); //Reallocate memory
                    outputsFirst[numOfOutputsFirst-1]=inputsSecond[i];
                }
            }
```

```
//Find Second's Outputs
for (int i = splitingPoint+1; i < netlistRows; i++)
{
    for (int j = 0; j < numOfOutputs; j++)
    {
        if ((netlist[i][2]==outputs[j]) || (netlist[i][3]==outputs[j]))
        {
            //check for duplicates
            bool duplicate = false;
            for (int k = 0; k < numOfOutputsSecond; k++)
            {
                duplicate = false;
                if (outputs[j]==outputsSecond[k])
                {
                    duplicate = true;
                break;
                }
            }
            // If not a duplicate then add it
            if (!duplicate)
            {
                numOfOutputsSecond++;
                if (numOfOutputsSecond==1)
                {
                    outputsSecond = (int*)malloc((sizeof(int)*numOfOutputsSecond)); //Allocate memory
                    outputsSecond[numOfOutputsSecond-1]=outputs[j];
                }
                else
                {
                    outputsSecond = (int*)realloc(outputsSecond,(sizeof(int)*numOfOutputsSecond)); //Reallocate memory
                    outputsSecond[numOfOutputsSecond-1]=outputs[j];
                }
            }
        }
    }
}
```

Η συνάρτηση αυτή δημιουργεί τους κόμβους εισόδου και εξόδου των 2 αρχείων που θα δώσουμε ως έξοδο.

Ξεκινάει δημιουργώντας την είσοδο του πρώτου αρχείου διαβάζοντας τους κόμβους που έχουν ρόλο πύλης μέχρι το σημείο τομής και δεν εμφανίζονται ως Source ή Drain.

Ύστερα δημιουργεί την είσοδο του δεύτερου αρχείου με τον ίδιο τρόπο.

Στη συνέχεια δημιουργεί την έξοδο του πρώτου αρχείου βρίσκοντας όλα τα Source ή Drain μέχρι το σημείο τομής, που ανήκουν σε έξοδο του αρχικού κυκλώματος. Μετά προσθέτει σε αυτά τις εισόδους του δεύτερου αρχείου.

Τέλος δημιουργεί την έξοδο του δεύτερου αρχείο διαβάζοντας τα Source και Drain από το σημείο τομής μέχρι το τέλος και εισάγει όσα είναι κοινά με την έξοδο του αρχικού κυκλώματος.

Σε όλα τα παραπάνω δε επιτρέπει τη δημιουργία διπλοτύπων.

Η συνάρτηση create2Files():

```
void crete2Files(FILE* fp1,FILE* fp2)
{
    printf("## RAILS\n");
    fprintf(fp1,"## RAILS\n");
    fprintf(fp2,"## RAILS\n");

    printf("VCC %d ; GND %d\n",VCC,GND);
    fprintf(fp1,"VCC %d ; GND %d\n",VCC,GND);
    fprintf(fp2,"VCC %d ; GND %d\n",VCC,GND);
    printf("## INPUTS\n");
    fprintf(fp1,"## INPUTS\n");
    fprintf(fp2,"## INPUTS\n");
    for (int i = 0; i < numOfInputsFirst; i++)
    {
        if (i==(numOfInputsFirst-1))
        {
            printf("%d\n",inputsFirst[i]);
            fprintf(fp1,"%d\n",inputsFirst[i]);
            break;
        }
        printf("%d,",inputsFirst[i]);
        fprintf(fp1,"%d,",inputsFirst[i]);
    }
    for (int i = 0; i < numOfInputsSecond; i++)
    {
        if (i==(numOfInputsSecond-1))
        {
            printf("%d\n",inputsSecond[i]);
            fprintf(fp2,"%d\n",inputsSecond[i]);
            break;
        }
        printf("%d,",inputsSecond[i]);
        fprintf(fp2,"%d,",inputsSecond[i]);
    }
```

```
printf("## OUTPUTS\n");
fprintf(fp1,"## OUTPUTS\n");
fprintf(fp2,"## OUTPUTS\n");
for (int i = 0; i < numOfOutputsFirst; i++)
{
    if (i==(numOfOutputsFirst-1))
    {
        printf("%d\n",outputsFirst[i]);
        fprintf(fp1,"%d\n",outputsFirst[i]);
        break;
    }
    printf("%d,",outputsFirst[i]);
    fprintf(fp1,"%d,",outputsFirst[i]);
}
for (int i = 0; i < numOfOutputsSecond; i++)
{
    if (i==(numOfOutputsSecond-1))
    {
        printf("%d\n",outputsSecond[i]);
        fprintf(fp2,"%d\n",outputsSecond[i]);
        break;
    }
    printf("%d,",outputsSecond[i]);
    fprintf(fp2,"%d,",outputsSecond[i]);
}
```

```
printf("## NETLIST\n");
fprintf(fp1,"## NETLIST\n");
fprintf(fp2,"## NETLIST\n");
int num=0;
for (int i = 0; i < splitingPoint; i++)
{
    num++;
    if (netlist[i][0]==0)
    {
        printf("U%d PMOS ",num);
        fprintf(fp1,"U%d PMOS ",num);
    }
    else
    {
        printf("U%d NMOS ",num);
        fprintf(fp1,"U%d NMOS ",num);
    }
    for (int j = 1; j < netlistCols; j++)
    {
        printf("%d ",netlist[i][j]);
        fprintf(fp1,"%d ",netlist[i][j]);
    }
    printf("\n");
    fprintf(fp1,"\n");
}
for (int i = splitingPoint+1; i < netlistRows; i++)
{
    num++;
    if (netlist[i][0]==0)
    {
        printf("U%d PMOS ",num);
        fprintf(fp2,"U%d PMOS ",num);
    }
    else
    {
        printf("U%d NMOS ",num);
        fprintf(fp2,"U%d NMOS ",num);
    }
    for (int j = 1; j < netlistCols; j++)
    {
        printf("%d ",netlist[i][j]);
        fprintf(fp2,"%d ",netlist[i][j]);
    }
    printf("\n");
    fprintf(fp2,"\n");
}
```

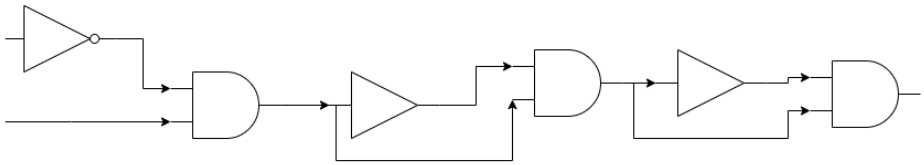
Η συνάρτηση create2Files() είναι υπεύθυνη για τη τελική δημιουργία των 2 αρχείων. Παίρνει όλα τα στοιχεία που έχουν δημιουργηθεί για τα 2 αρχεία και τα εκτυπώνει μέσα σε κάθε αρχείο ξεχωριστά.

Ανάλυση της Διαδικασίας

Παράδειγμα με χρήση του παρακάτω κυκλώματος:

```
## RAILS
VCC 11 ; GND 14
## INPUTS
12 ; 25 ;
## OUTPUTS
62 ;
## NETLIST
U1 PMOS 12 11 13
U2 NMOS 12 13 14
U3 PMOS 25 11 22
U4 PMOS 13 11 22
U5 NMOS 25 22 23
U6 NMOS 13 23 14
U7 PMOS 22 11 33
U8 NMOS 22 33 14
U9 PMOS 33 11 42
U10 PMOS 22 11 42
U11 NMOS 33 42 43
U12 NMOS 22 43 14
U13 PMOS 42 11 53
U14 NMOS 42 53 14
U15 PMOS 42 11 62
U16 PMOS 53 11 62
U17 NMOS 42 62 63
U18 NMOS 53 63 14
```

Το κύκλωμα αποτελείται από 3 NOT και 3 NAND συνδεδεμένες σε σειρά με την εξής μορφή.



Αφού διαβαστεί το αρχείο και έχουμε αποθηκεύσει όλο το Netlist ξεκινάει η εκτέλεση των συναρτήσεων μας.

Μετά την συνάρτηση findPossibleSplitPoints() ο πίνακας του netlist έχει τη παρακάτω μορφή:

```
NETLIST IS : (where 0 in first column = PMOS and 1 = NMOS)
0 12 11 13 0
1 12 13 14 0
0 25 11 22 0
0 13 11 22 1
1 25 22 23 0
1 13 23 14 1
0 22 11 33 1
1 22 33 14 1
0 33 11 42 1
0 22 11 42 1
1 33 42 43 1
1 22 43 14 1
0 42 11 53 1
1 42 53 14 1
0 42 11 62 1
0 53 11 62 1
1 42 62 63 1
1 53 63 14 1
```

Όπου η τελευταία στήλη με 1 μας δείχνει πιθανά σημεία τομής και με 0 σημεία που σίγουρα δεν είναι σημείο τομής.

Μετά την συνάρτηση verifySplitPoints() ο πίνακας του netlist έχει τη παρακάτω μορφή:

```
NETLIST IS : (where 0 in first column = PMOS and 1 = NMOS)
0 12 11 13 0 0
1 12 13 14 0 0
0 25 11 22 0 0
0 13 11 22 0 0
1 25 22 23 0 0
1 13 23 14 1 -1
0 22 11 33 1 1
1 22 33 14 1 -1
0 33 11 42 1 1
0 22 11 42 0 0
1 33 42 43 0 0
1 22 43 14 1 -1
0 42 11 53 1 1
1 42 53 14 1 -1
0 42 11 62 1 1
0 53 11 62 0 0
1 42 62 63 0 0
1 53 63 14 1 0
Number of Transistors 18
```

Όπου στην τελευταία στήλη βλέπουμε με 1 τους κόμβους που μπορούμε να σπάσουμε πριν από αυτούς και δε τους συμπεριλαμβάνουμε στο δεύτερο κύκλωμα , ενώ βλέπουμε -1 στους κόμβους που μπορούμε να κόψουμε μετά από αυτούς και του συμπεριλαμβάνουμε στο δεύτερο κύκλωμα.

Μετά την συνάρτηση splitFile() ο πίνακας του netlist έχει τη παρακάτω μορφή:

```
Splitting point = 7
NETLIST IS : (where 0 in first column = PMOS and 1 = NMOS)
0 12 11 13 0 0
1 12 13 14 0 0
0 25 11 22 0 0
0 13 11 22 0 0
1 25 22 23 0 0
1 13 23 14 1 -1
0 22 11 33 1 1
1 22 33 14 1 -1

0 33 11 42 1 1
0 22 11 42 0 0
1 33 42 43 0 0
1 22 43 14 1 -1
0 42 11 53 1 1
1 42 53 14 1 -1
0 42 11 62 1 1
0 53 11 62 0 0
1 42 62 63 0 0
1 53 63 14 1 0
```

Ουσιαστικά βρίσκει τον κόμβο που θα γίνει η τομή του αρχείου και προσομοιώνει πως θα γίνει η τομή του αρχείου στα 2 καινούργια αρχεία.

Τελικά η συνάρτηση createInputsOutputs() μας δημιουργεί την παρακάτω πληροφορία:

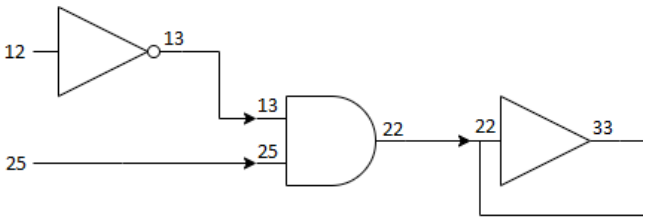
```
First Inputs:
12
25
First's Outputs:
33
22
Second's Inputs:
33
22
Second's Outputs:
62
```

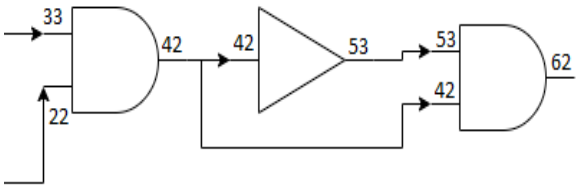
Όπως βλέπουμε εκτυπώνει τις εισόδους των 2 αρχείων και τις εξόδους τους. Ο κώδικας του εργαστηρίου μπορεί να διαχειριστεί και τομές αρχείων σε παραπάνω από ένα καλώδια όπως συμβαίνει σε αυτό το παράδειγμα όπως η τομή συμβαίνει στους κόμβους 22, 33 οι οποίοι βγαίνουν ως έξοδο από το πρώτο κύκλωμα και ταυτόχρονα γίνονται είσοδο στο δεύτερο.

Τέλος η συνάρτηση create2Files() δημιουργεί τα 2 αρχεία τα οποία έχουν τη παρακάτω μορφή:

```
## RAILS
VCC 11 ; GND 14
## INPUTS
12,25
## OUTPUTS
33,22
## NETLIST
U1 PMOS 12 11 13
U2 NMOS 12 13 14
U3 PMOS 25 11 22
U4 PMOS 13 11 22
U5 NMOS 25 22 23
U6 NMOS 13 23 14
U7 PMOS 22 11 33
```

```
## RAILS
VCC 11 ; GND 14
## INPUTS
33,22
## OUTPUTS
62
## NETLIST
U8 PMOS 33 11 42
U9 PMOS 22 11 42
U10 NMOS 33 42 43
U11 NMOS 22 43 14
U12 PMOS 42 11 53
U13 NMOS 42 53 14
U14 PMOS 42 11 62
U15 PMOS 53 11 62
U16 NMOS 42 62 63
U17 NMOS 53 63 14
```





Αρχείο 1

Αρχείο 2

Ripple Carry Adder 8 bits

Έγινε κανονικά η διαδικασία παράγωγή του Ripple Carry Adder δημιουργώντας ένα αρχείο από πύλες και στη συνέχεια με χρήση του εργαστηρίου 3 πήραμε το επίπεδο Dataset για τον RCA_8bit. Στη συνέχεια αυτό το αρχείο το δώσαμε στον κώδικα του εργαστηρίου 4 και λάβαμε τα 2 νέα αρχεία. Προσθέσαμε τα TESTBENCH που θέλαμε με έναν editor και ύστερα δώσαμε τα αρχεία ως είσοδο στον απλό προσομοιωτή του εργαστηρίου 2 και τέλος λάβαμε σωστή προσομοίωση.

Τα αρχεία εισόδου και εξόδου βρίσκονται στον ίδιο φάκελο με αυτό το έγγραφο με ονόματα:

Στον φάκελο **Input Lab3**:

RCA_8bits.txt : Το αρχείο εισόδου της άσκησης 3.

RCA.LIB : Το αρχείο βιβλιοθήκης για της πύλες του αρχείου εισόδου της άσκησης 3.

Στον φάκελο **input Lab4**:

RCA_8bits_out.txt : Το αρχείο εξόδου της άσκησης 3 και εισόδου για την άσκηση 4.

Στον φάκελο **Output Lab4**:

RCA_8bits_out_out1.txt : Το πρώτο αρχείο εξόδου της άσκησης 4 και είσοδος στον απλό προσομοιωτή.

RCA_8bits_out_out2.txt : Το δεύτερο αρχείο εξόδου της άσκησης 4 και είσοδος στον απλό προσομοιωτή.

Στον φάκελο **Output Lab2**:

RCA_8bits_out_out1_out.txt: Το πρώτο αρχείο εξόδου του απλού προσομοιωτή.

RCA_8bits_out_out2_out.txt: Το δεύτερο αρχείο εξόδου του απλού προσομοιωτή.

Συμπέρασμα

Όπως φαίνεται και από όλα τα παραπάνω αρχεία το πρόγραμμα λειτουργεί κανονικά και κόβει το αρχείο σε 2 περίπου ίσα αρχεία χωρίς λάθη τα οποία μπορούν ύστερα να προσομοιωθούν κανονικά από τον απλό προσομοιωτή του εργαστηρίου 2.