|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| This program calculates the final signatures based on the number of branches in the program  The calculated signatures can be grouped into a maximum of eight different ANDOR | | | | | | | | | | | | | | | | |
| new\_gen\_group\_sizes(n, k, m, j, flag, numNeq): | | | | | | | | | | | | | | | | |
|  | n: problem size (init with problem size),   k: array of number of new primes in each grouping (init with []),   m: array of configuration (init with []),   j: inner index of m and k array (init with 0),   flag: the config with redandant primes(init with 0),   numeq: number of unused element in "flag" and the past ones(init with 0). | | | | | | | | | | | | | | | |
| This function is recursive, and you need to only initial it with the number of branches as ‘n’. The rest will be filled by recursive calls | | | | | | | | | | | | | | | |
|  |  | For i any power of two numbers in the range of (20 downto 0) | | | | | | | | | | | | | | |
|  |  |  | If n = 2\*\*i | | | | | | | | | | | | | |
|  |  |  |  | Return k, m, flag, numNeq | | | | | | | | | | | | |
|  |  |  | Else if n > 2\*\*i | | | | | | | | | | | | | |
|  |  |  |  | new\_gen\_group\_sizes (n-(2\*\*i), k.append(2\*\*i), m.append(i), j+1, flag, numNeq) | | | | | | | | | | | | |
|  |  |  | Else if n > 2\*\*(i-1) | | | | | | | | | | | | | |
|  |  |  |  | This condition generates prime implicants with overlap with previous prime implicants  Only one prime implicant with overlap is supported in this scenario  This condition is just to reduce the number of prime implicants | | | | | | | | | | | | |
|  |  |  |  | p = the needed overlap | | | | | | | | | | | | |
|  |  |  |  | new\_gen\_group\_sizes (n-(2\*\*i)+p, k.append(2\*\*i-p), m.append(i), j+1, 20-i, p) | | | | | | | | | | | | |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| mk\_gr (x, flag, numNeq): | | | | | | | | | | | | | | | | |
|  | x: m return from new\_gen\_group\_sizes  flag: flag return from new\_gen\_group\_sizes  numeq: numeq return from new\_gen\_group\_sizes | | | | | | | | | | | | | | | |
|  | This function generates the prime implicants by considering the adjacent minterms. The order of minterms can be indicated in int\_primes.  Any change to int\_primes will generate different distribution to the final signatures in the 2\*\*20 space  To consider there is no overlap in different prime implicants, one of the midterms must be opposite to other midterms in the previous groups  Flag defines the prime implicant with overlap. To consider this, there is a need to only have overlap with a specific prime implicant and avoid any overlap with the others | | | | | | | | | | | | | | | |
|  | int\_primes = [ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19] | | | | | | | | | | | | | | | |
|  | For i in range len(x): | | | | | | | | | | | | | | | |
|  |  | If flag = 0 or flag > i: | | | | | | | | | | | | | | |
|  |  |  | This condition generates prime implicants without overlap by adding NOT in the binary fashion for first minterms: 000, 001, 010, 011, … ones indicate NOTs | | | | | | | | | | | | | |
|  |  | Else if flag = i : | | | | | | | | | | | | | | |
|  |  |  | By considering “numeq” generate the prime implicant with overlap. | | | | | | | | | | | | | |
|  | Return two different arrays; the first one indicates the adjacent minterms that need to be considered as zero are not written here, and the second one indicates the minterms that are variable for each group. | | | | | | | | | | | | | | | |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| bin\_config\_mem(a, nota): | | | | | | | | | | | | | | | | |
|  | a, nota : both are the return objects from mk\_gr | | | | | | | | | | | | | | | |
|  | This function gets the results of mk\_gr and generates the configuration bits for PLA  The generation bit fashion is based on the current reading instruction and filling pattern of PLA; for each line, there is a need to 40’b, and in total, the output will be 320’b  ‘1’ means that the PLA cell is active | | | | | | | | | | | | | | | |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| call\_groups(primes,x,y): | | | | | | | | | | | | | | | | |
|  | primes: usually fill with the null list ([]), but it can also fill with a list of individual signatures that we need  x, y: both are the return objects from mk\_gr | | | | | | | | | | | | | | | |
|  | This function generates all the signatures in integer | | | | | | | | | | | | | | | |
|  | For i in range len(x): | | | | | | | | | | | | | | | |
|  |  | Based on the len(x[i]) different functions will be called in the format of ”group”+ ”len(x[i])” | | | | | | | | | | | | | | |
|  |  | the ”group”+ ”len(x[i])” is responsible for generating different members for each prime implicant groups | | | | | | | | | | | | | | |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| group+num(primes,x,y): | | | | | | | | | | | | | | | | |
|  | primes: usually filled by the program with the list of individual signatures that we already have but also can be filled with the null list ([])  x, y: both are one of the return objects from the list that is generated in mk\_gr  num is equal to the len(y) | | | | | | | | | | | | | | | |
|  |  | Based on different num, there are different numbers of nested for loops to generate all the member groups. | | | | | | | | | | | | | | |

