// i need 19 registers in total and the sizes of each including the stack is all defined below with explanation

reg [3:0] k;

reg [3:0] nextk; //max number they can hold is 8 which is the max depth

reg [2:0] STACK_IX[1:8];//max number of children is 4 so we need to represent 4 in a stack ix entry

reg [3:0] ix;

reg [2:0] STACK_NUMEL[1:8];//max number of children is 4 so we need to represent 4 in a stack ix entry

reg [17:0] STACK_ADRS [1:C][1:D] //2D array for the stack which has C max children in x dim and D depth //in y dim with bit size 18 bits per element

// we have 18 bits since max num of nodes is 21845 and since in the array T[] in memory each of these //wee have a max of 7 entries so biggest pointer is

//21845 *7 which needs 18 bits to represent it.

reg [14:0] wpath[1:8];

//here we need 15 bits since assume a tree where all nodes have max weight

//then we need 4095(max weight per node) *8 which needs 15 bits to represent

reg [17:0] pathmin [1:8];//18 bits since it stores the pointers

reg [17:0] path [1:8];

reg [17:0]c1;//18 bits since it stores the pointers

reg [17:0]c2;

reg [17:0]c3;

reg [17:0]c4;

reg [2:0]nc;// maximum value=4

reg [14:0] wpathmin;//same as wpath logic

reg [11:0] w;// weight needs 12 bits

reg [17:0]adrs;//stores values form STACK_ADRS

reg START,DONE;//one bit signals

// TREE and C and D come as inputs to the module and we dont need to initialize them as registers in //our code we assume TREE is stored in MEMORY

```
// starting from address 0 as mentioned in project description



   //initializations
   if(START==1)
   begin
   STACK_ADRS <- 0; // in verilog this requires a loop since it is an array reg but for simplicity in
//pseudocode this sets whole array to 0
   wpathmin <- 12'b11111111111;//max number possible so as if inifinity
   nextk <- 1;
   STACK_IX <- 1;//sets all array to 1 in each position
   STACK_NUMEL <- 1;//sets all array to 1 in each position
   //from here on start tree traversal
   while nextk != 0
   k <- nextk;
      if STACK_NUMEL[k] <= 0 // less than or equal
   nextk <- k - 1; // move up one level
   else
// read stack at current level
   ix <- STACK_IX[k];
   adrs <- STACK_ADRS[k][ix]; // read top of stack
   STACK_IX[k] <- STACK_IX[k] + 1; // update index
   STACK_NUMEL[k] <- STACK_NUMEL[k] - 1; // update numel left
// read info of node, update path and path-weight
w <- MEMORY[adrs+1];
nc<-MEMORY[adrs+3];
path[k] <- adrs;
   if k==1, wpath[k] <- w;
   else
```

```
    wpath[k] <- wpath[k-1] + w;
  end
  if wpath[k]>wpathmin, nextk <- k-1; //if the weight of the path we are on now is more than the
//minimum no need to check more

  elseif nc==0 // current node is a leaf: update minimum
  if wpath[k] < wpathmin
        wpathmin <- wpath[k]; pathmin <- path(1:k);//for simplicity assume path(1:k) will work as in
matlab inputing to the path register
  end
 else // current node is NOT a leaf
  if nc==1
    c1 <- MEMORY[adrs+4];
    nextk <- k+1;
    STACK_ADRS[k+1][1] <- [c1];
    STACK_IX[k+1] <- 1;
    STACK_NUMEL[k+1] <- 1;
  elseif nc==2
    c1 <- MEMORY[adrs+4];
    c2 <- MEMORY[adrs+5];
    nextk <- k+1;
    STACK_ADRS[k+1][1:2]<- [c1,c2]; // assume the : operator works as it does in matlab for this
//pseudocode so this line is assigining 2 data values to 2 different location in STACK_ADRS
    STACK_IX[k+1]<-1;
    STACK_NUMEL[k+1] <- 2;
  elseif nc==3
    c1 <- MEMORY[adrs+4];
    c2 <- MEMORY[adrs+5];
    c3 <- MEMORY[adrs+6];
    nextk <- k+1;
```

```
      STACK_ADRS[k+1][1:3] <- [c1,c2,c3];
      STACK_IX[k+1] <- 1;
      STACK_NUMEL[k+1] <- 3;
   elseif nc==4
      c1 <- MEMORY[adrs+4];
      c2 <- MEMORY[adrs+5];
      c3 <- MEMORY[adrs+6];
      c4 <- MEMORY[adrs+7];
      nextk <- k+1;
      STACK_ADRS[k+1][1:4] <- [c1,c2,c3,c4];
      STACK_IX[k+1] <- 1;
      STACK_NUMEL[k+1] <- 4;

   end
   end
   end
   end
  DONE <- 1;
     //here we are done with traversing the tree and wpathmin and pathmin hold the weight min and the
//path that gave this weight respectively
     // this needs to be taken care of in verilog code to actually have every node written to pathmin while
//DONE is asserted.
   end
```