भारतीय प्रौद्योगिकी संस्थान गुवाहाटी
INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI

# PROJECT REPORT ON IMPLEMENTATION OF MERGE SORT USING VERILOG

26TH April 2022

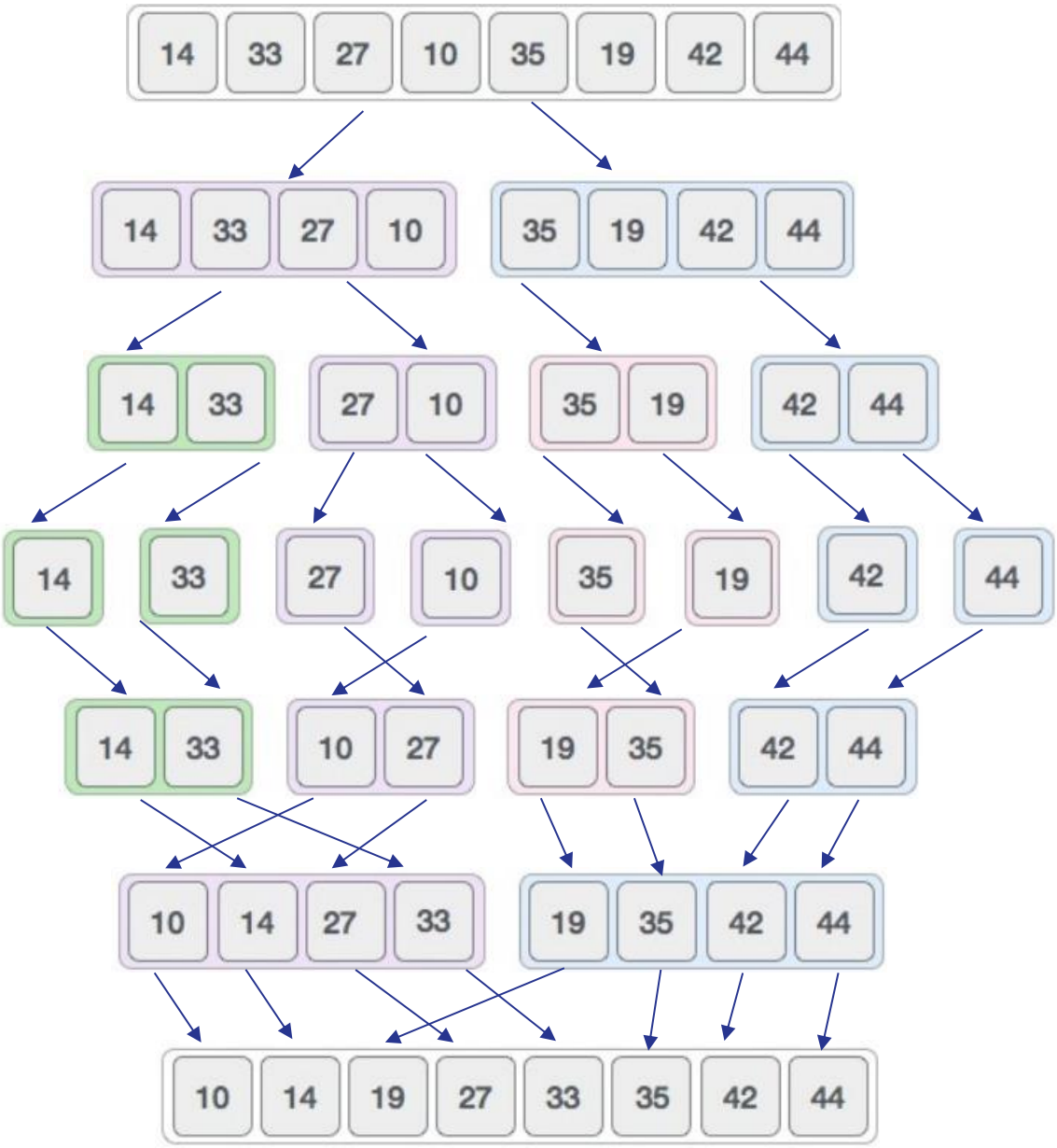
EE518- VLSI LAB-3


SUBMITTED TO:-
DR. GAURAV TRIVEDI
MEENALI JANVEJA
ANKITA TIWARI

SUBMITTED BY:-
ARKAJIT SENGUPTA 214102403
SUYASH MISHRA 214102415
ASHISH KUMAR TIWARI 214102417

## MERGE SORT:-

A *sort* algorithm that splits the items to be sorted into two groups, *recursively* sorts each group, and *merges* them into a final, sorted sequence.



## DIVIDE AND CONQUER STRATEGY

Using the Divide and Conquer technique, we divide a problem into subproblems. When the solution to each subproblem is ready, we 'combine' the results from the subproblems to solve the main problem.

For example:-
Suppose we had to sort an array A.
A subproblem would be to sort a sub-section of this array starting at index p and ending at index r, denoted as A[p..r].

- Divide
  If q is the half-way point between p and r, then we can split the subarray A[p..r] into two arrays A[p..q] and A[q+1, r].
- Conquer
  In the conquer step, we try to sort both the subarrays A[p..q] and A[q+1, r]. If we haven't yet reached the base case, we again divide both these subarrays and try to sort them.
- Combine
  When the conquer step reaches the base step and we get two sorted subarrays A[p..q] and A[q+1, r] for array A[p..r], we combine the results by creating a sorted array A[p..r] from two sorted subarrays A[p..q] and A[q+1, r].

## Drawbacks of Merge Sort

- Slower comparative to the other sort algorithms for smaller tasks.
- It goes through the whole process even if the array is sorted

## State diagram flow:

## STATE-a

- Initialize all variables and registers.
- Ex-i=1,l=0,j=0 etc.

## STATE-b

- Initialize the register bank with all the numbers to be sorted.
- At the time of inserting to the bank,after inserting two numbers,the current number entered is compared with the previous one.
- This thing is repeated at an interval of every two numbers inserted.

If(index<N)

If(ind==1)

- Store the no in temp register
- Ind=2

Else if(ind==2)

- Compare the current data with the temp register data and store accordingly
- Ind=1
- index=index+2

else

- goto state-c

## STATE-c

- Set subarray_A and subarray_B start index.

If(startA+startB<N)

- goto state-d

else

- goto state-g

## STATE-d

- Comparison of subarray elements

If(count<2^(i+1))

If(L<2^i && j<2^i)

If(reg_bank[startA+L]<reg_bank[startB+j])

- TempR<-reg_bank[startA+L]
- L<-L+1

Else

- TempR<-reg_bank[startB+j]
- j<j+1

else

if(L==(2^i))

- TempR<-reg_bank[startB+j]
- j<-j+1
- state<-d

else if(j==(2^i))

- TempR<-reg_bank[startA+L]
- L<-L+1
- state<-d

else

- state<-e

## STATE-e

- Increment startA and startB index till all the numbers are compared in pairs

  - startA<=startA+2^(i+1)
  - startB<=startB+2^(i+1)
  - if(startA<N)
    - state<=d

    else
    - state<=f

## STATE-f

- Store the sorted subarrays in the reg_bank and increment the subarray length for next comparison.

If(count<N)

- Reg_bank[count]<=TempR
- count<=count+1
- state<=f

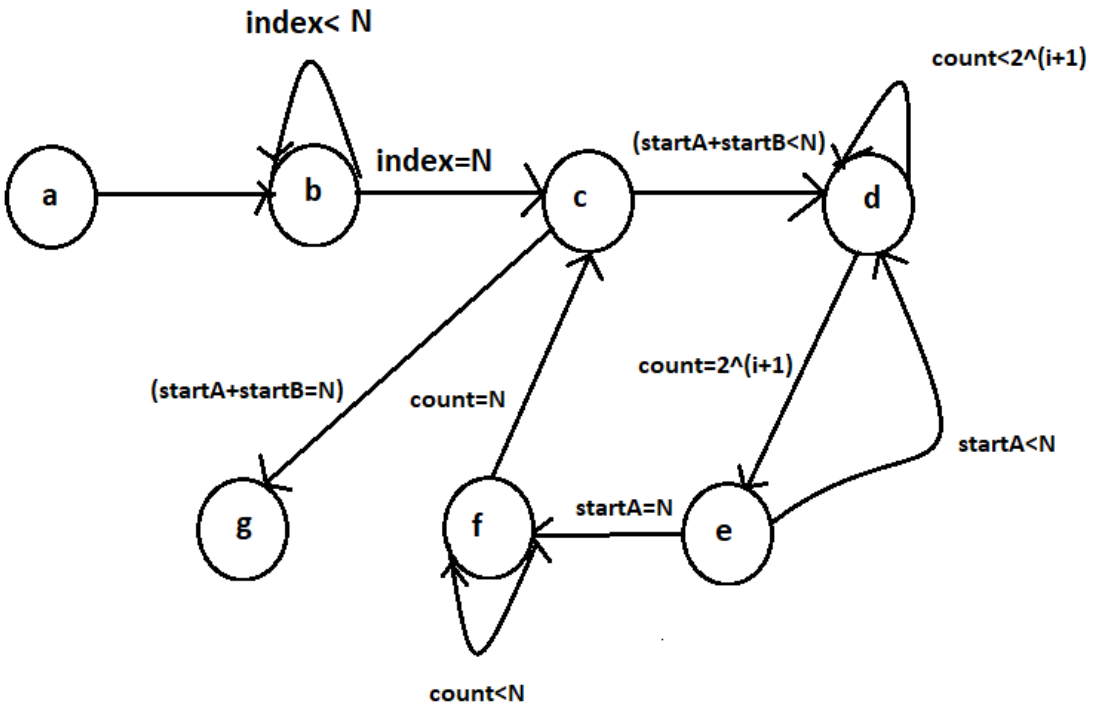
else

- i<=i+1
- state<=c

## STATE-g

- Store the final result in the register bank and display the result.

## STATE DIAGRAM:




## EXPLANATION WITH AN EXAMPLE

Let us consider the numbers below to be sorted

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| -9 | 6 | 0 | 2 | -3 | -1 | 8 | 0 |

## STATE-a

- i=0
- L=0
- J=0
- Index=0

## STATE-b

| CLOCK | DATABUS | temp | ind | index | Reg_bank |
|---|---|---|---|---|---|
| 1 | -9 | -9 | 1 | 0 | |
| 2 | 6 | | 2 | | reg_bank[1]=-6<br>reg_bank[0]=-9 |
| 3 | 0 | 0 | 1 | 2 | |
| 4 | 2 | | 2 | | reg_bank[3]=-2<br>reg_bank[2]=-0 |
| 5 | -3 | -3 | 1 | 4 | |
| 6 | -1 | | 2 | | reg_bank[5]=--1<br>reg_bank[4]=-3 |
| 7 | 8 | 8 | 1 | 6 | |
| 8 | 0 | | 2 | | reg_bank[7]=-8<br>reg_bank[6]=-0 |

- After that as index=8,it moves to state-c

## STATE-c

- startA=0
- startB=2 (as i=1)
- Now,as (startA+startB<8),It will move to state-d

## STATE-d

| | |
|---|---|
| -9 | 6 |

| | |
|---|---|
| 0 | 2 |

- L=0,j=0
- L=1,j=0,count=1,tempR=-9xxxxxxx
- L=1,j=1,count=2,tempR=0 -9xxxxxx
- L=1,j=2,count=3,tempR=2 0 -9xxxxx
- L=2,j=2,count=4,tempR=6 2 0 -9xxxx

- Now as count==4,it moves to state-e

## STATE-e

- startA<=startA+2^(2)=4
- startB<=startB+2^(2)=6
- As startA<N,it goes to state-d

## STATE-d

| | |
|---|---|
| -3 | -1 |

| | |
|---|---|
| 0 | 8 |

- L=0,j=0
- L=1,j=0,count=1,tempR=--3 6 2 0 -9xxx
- L=2,j=0,count=2,tempR=-1 -3 6 2 0 -9xx
- L=2,j=1,count=3,tempR=0 1 -3 6 2 0 -9x
- L=2,j=2,count=4,tempR=8 0 1 -3 6 2 0 -9
- Now as count==4,it moves to state-e

## STATE-e

- startA<=startA+2^(2)=8
- startB<=startB+2^(2)=10
- as startA>=8,so it goes to state-f

## STATE-f

- Till count is not N(=8),update the register bank with sorted pairs.
- reg_bank[0]=-9, reg_bank[1]=0, reg_bank[2]=2, reg_bank[3]=3, reg_bank[4]=-3, reg_bank[5]=-1, reg_bank[6]=0, reg_bank[7]=8
- When count=8, update i=2 and goto state-c

## STATE-c

- Now as i=2,
- update startA=0 and startB=4
- goto state-d

## STATE-d

| | | | |
|---|---|---|---|
| -9 | 0 | 2 | 3 |

| | | | |
|---|---|---|---|
| -3 | -1 | 0 | 8 |

- L=0,j=0
- L=1,j=0,count=1,tempR=-9xxxxxxx
- L=1,j=1,count=2,tempR=-3 -9xxxxxx
- L=1,j=2,count=3,tempR=-1 -3 -9xxxxx
- L=2,j=2,count=4,tempR=0 -1 -3 -9xxxx
- L=3,j=2,count=5,tempR=0 0 -1 -3 -9xxx
- L=3,j=3,count=6,tempR=2 0 0 -1 -3 -9xx
- L=4,j=3,count=7,tempR=3 2 0 0 -1 -3 -9x
- L=4,j=4,count=8,tempR=8 3 2 0 0 -1 -3 -9

Now as count=8,it goes to state-e

## STATE-e

- startA<=startA+2^(3)=8
- startB<=startB+2^(3)=12
- As startA>=8,it now goes to state-f

## STATE-f

- Till count=8,update the register bank with sorted pairs.
- reg_bank[0]=-9, reg_bank[1]=-3, reg_bank[2]=-1, reg_bank[3]=0, reg_bank[4]=0, reg_bank[5]=2, reg_bank[6]=3, reg_bank[7]=8
- when count=8,update i=3 and goto state-c

## STATE-c

- Update startA and startB
- startA=0
- startB=8
- Now as (startA+startB)>=8,so now it goes to state-g

## STATE-g

- As the whole array of numbers is sorted,we now display the final result in the register bank.

## VERILOG CODE:

```
module merge_sort_vsd(
output reg [31:0] op,input [31:0] data_bus,input clk,res
    );
    parameter N=8;
    parameter O=7;
    parameter a=3'b000,b=3'b001,c=3'b010,d=3'b011,e=3'b100,f=3'b101,g=3'b110,h=3'b111;
    parameter def_value=32'h7f7f_ffff;
    reg [31:0] rg_bank [0:O-1];
    reg [(32*N)-1:0] tempR;
    reg [2:0] state;
    integer count,ind,startA,startB,i,j,k,l,index,M;
    reg [31:0] temp,tempA;
    reg [7:0] word=8'b0000_0001;
    reg [31:0] reg_bank [0:N-1];
    always @(posedge clk)
        begin
            if(res)
                begin
                    state<=a;
                    M<=(O[0]==0)?O:(O+1);
                end
            else
```

```
begin
    case(state)
        a: begin
            count<=0;
            state<=b;
            ind<=1;
            i<=1;
            k<=0;
            l<=0;
            j<=0;
            index<=N-M;
          end
        b: begin
                if(index<N)
                    begin
                        state<=b;
                        if(ind==1)
                            begin
                                tempA<=data_bus;
                                ind<=2;
                            end
                         else if(ind==2 && ((index!=(N-2))||O[0]==0))
                            begin
                                if(tempA[31]==1'b0 && data_bus[31]==1'b0)
                                    begin
                                    if(tempA>data_bus)
                                        begin

                                            reg_bank[index]<=data_bus;
                                            reg_bank[index+1]<=tempA;
                                            index<=index+2;
                                            ind<=1;
                                        end
                                    else
                                        begin

                                            reg_bank[index]<=tempA;
                                            reg_bank[index+1]<=data_bus;
                                            index<=index+2;
                                            ind<=1;
                                        end
```

```
            end
        else if(tempA[31]==1'b1 && data_bus[31]==1'b1)
            begin
            if(tempA<data_bus)
                begin

                    reg_bank[index]<=data_bus;
                    reg_bank[index+1]<=tempA;
                    index<=index+2;
                    ind<=1;
                end
            else
                begin

                    reg_bank[index]<=tempA;
                    reg_bank[index+1]<=data_bus;
                    index<=index+2;
                    ind<=1;
                end
         end


        else if(tempA[31]==1'b0 && data_bus[31]==1'b1)
            begin
                    reg_bank[index]<=data_bus;
                    reg_bank[index+1]<=tempA;
                    index<=index+2;
                    ind<=1;
            end

       else
            begin
                    reg_bank[index]<=tempA;
                    reg_bank[index+1]<=data_bus;
                    index<=index+2;
                    ind<=1;
            end
    end
```

```
                else if(index==(N-2))
                    begin
                        reg_bank[index]=data_bus;
                        reg_bank[index+1]=def_value;
                        index<=index+2;
                    end


            end
        else
            begin
                count<=0;
                state<=c;
            end
    end
c:  begin
        count<=count+1;
        startA<=0;
        startB<=(word<<i);

        if(count==0)
            begin
                state<=c;
            end
        else
            begin
        if((startA+startB)<N)
            begin
               state<=d;
               count<=0;
            end
        else
            begin
                state<=g;
               count<=0;
            end
           end
    end
```

```
d:  begin

        if(count<(word<<(i+1)))
            begin
                count<=count+1;
                if(l<(word<<i)&&j<(word<<i))
                    begin
                        if(reg_bank[startA+j][31]==1'b0 && reg_bank[startB+l][31]==1'b0)
                           begin
                                if(reg_bank[startA+j]<=reg_bank[startB+l])
                                   begin
                                       tempR<={reg_bank[startA+j],tempR}>>32;
                                       j<=j+1;
                                   end
                                else if(reg_bank[startA+j]>reg_bank[startB+l])
                                   begin
                                       tempR<={reg_bank[startB+l],tempR}>>32;
                                       l<=l+1;
                                   end
                           end
                        else if(reg_bank[startA+j][31]==1'b1 && reg_bank[startB+l][31]==1'b1)
                           begin
                                if(reg_bank[startA+j]>reg_bank[startB+l])
                                   begin
                                       tempR<={reg_bank[startA+j],tempR}>>32;
                                       j<=j+1;
                                   end
                                else if(reg_bank[startA+j]<=reg_bank[startB+l])
                                   begin
                                       tempR<={reg_bank[startB+l],tempR}>>32;
                                       l<=l+1;
                                   end
                           end
                       else if(reg_bank[startA+j][31]==1'b1 && reg_bank[startB+l][31]==1'b0)
                           begin
                                       tempR<={reg_bank[startA+j],tempR}>>32;
                                       j<=j+1;
                           end
```

```
                end
            else
                begin
                    if(l==(word<<i))
                        begin
                            tempR<={reg_bank[startA+j],tempR}>>32;
                            j<=j+1;
                            state<=d;
                        end


                    else
                        begin
                            tempR<={reg_bank[startB+l],tempR}>>32;
                            l<=l+1;
                            state<=d;
                        end
                end
         end
      else
         begin
             state<=e;
         end
end
```

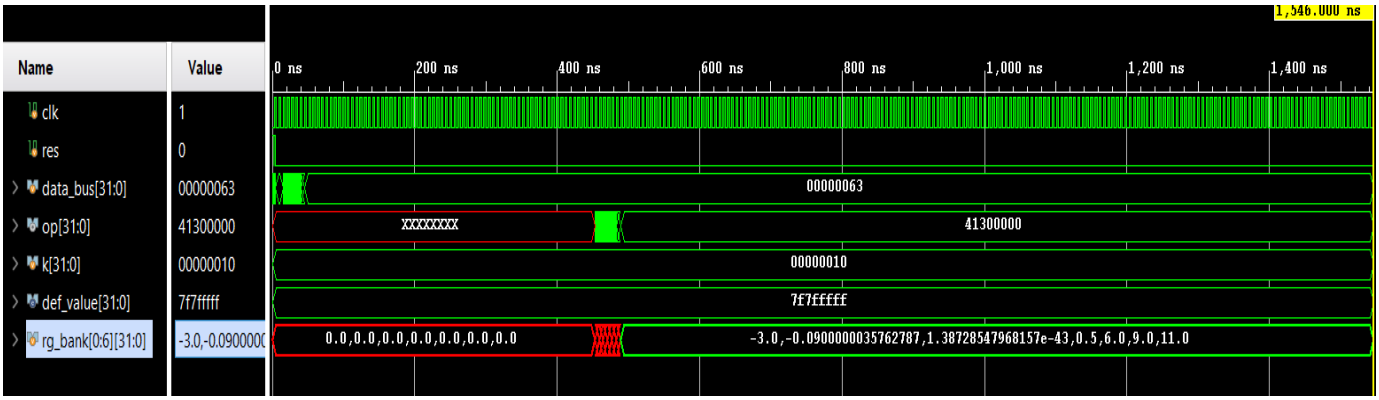
```
e: begin
        count<=0;
        j<=0;
        l<=0;
        startA<=startA+(word<<(i+1));
        startB<=startB+(word<<(i+1));
        if(startA<N)
            begin
                state<=d;
            end
        else
            begin
                state<=f;
                index<=0;
            end
        end
f: begin
        index<=index+1;
        if(index==0)
            begin
                state<=f;
                {tempR,temp}<={tempR,temp}>>32;
            end
        else
            begin
                {tempR,temp}<={tempR,temp}>>32;
                if(count<N)
                    begin
                        reg_bank[count]<=temp;
                        count<=count+1;
                        state<=f;
                    end
                else
                    begin
                        i<=i+1;
                        state<=c;
                        count<=0;
                    end
            end
   end
```

```
            g: begin
                        if(count<0)
                              begin
                                    rg_bank[count]<=reg_bank[count];
                                    count<=count+1;
                                    state<=g;
                                   op<=reg_bank[count];
                              end
                        else
                              begin
                                  state<=g;
                              end
                   end


                default: begin    state<=g; end
              endcase
          end
      end
endmodule
```

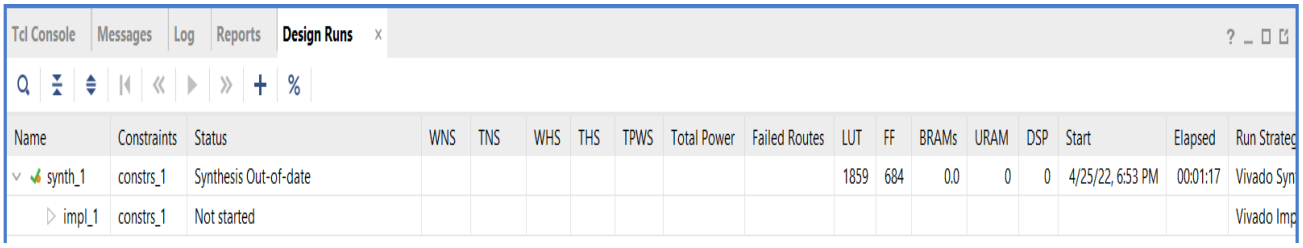
# VERILOG TESTBENCH:

```
module merge_sort_vsd_tb;
    reg clk,res;
    reg [31:0] data_bus;
    wire [31:0] op;
    integer k;
    parameter def_value=32'h7f7f_ffff;
    merge_sort_vsd ms(.op(op),.clk(clk),.res(res),.data_bus(data_bus));
    initial
        begin
            clk=1'b0;
            forever #3 clk=~clk;
        end
    initial
        begin
         res=1;

         for(k=0;k<16;k=k+1)
            begin
                ms.reg_bank[k]=def_value;
            end

         #4 res=0;data_bus=32'b10111101101110000101000111101100;
         #12 data_bus=32'b00111111000000000000000000000000;
         #6 data_bus=32'b01000000110000000000000000000000;
         #6 data_bus=32'b11000000010000000000000000000000;
         #6 data_bus=32'b01000001001100000000000000000000;
         #6 data_bus=32'b01000001000100000000000000000000;
         #6 data_bus=32'd99;
         #1500 $finish;
        end
endmodule
```

# SIMULATION OUTPUT:




# SYNTHESIS REPORT



1. Slice Logic

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Slice LUTs* | 1859 | 0 | 41000 | 4.53 |
| LUT as Logic | 1827 | 0 | 41000 | 4.46 |
| LUT as Memory | 32 | 0 | 13400 | 0.24 |
| LUT as Distributed RAM | 0 | 0 | | |
| LUT as Shift Register | 32 | 0 | | |
| Slice Registers | 684 | 0 | 82000 | 0.83 |
| Register as Flip Flop | 684 | 0 | 82000 | 0.83 |
| Register as Latch | 0 | 0 | 82000 | 0.00 |
| F7 Muxes | 86 | 0 | 20500 | 0.42 |
| F8 Muxes | 0 | 0 | 10250 | 0.00 |

4. IO and GT Specific
---

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Bonded IOB | 66 | 0 | 300 | 22.00 |
| Bonded IPADs | 0 | 0 | 26 | 0.00 |
| Bonded OPADs | 0 | 0 | 16 | 0.00 |
| PHY_CONTROL | 0 | 0 | 6 | 0.00 |
| PHASER_REF | 0 | 0 | 6 | 0.00 |
| OUT_FIFO | 0 | 0 | 24 | 0.00 |
| IN_FIFO | 0 | 0 | 24 | 0.00 |
| IDELAYCTRL | 0 | 0 | 6 | 0.00 |
| IBUFDS | 0 | 0 | 288 | 0.00 |
| GTXE2_COMMON | 0 | 0 | 2 | 0.00 |
| GTXE2_CHANNEL | 0 | 0 | 8 | 0.00 |
| PHASER_OUT/PHASER_OUT_PHY | 0 | 0 | 24 | 0.00 |
| PHASER_IN/PHASER_IN_PHY | 0 | 0 | 24 | 0.00 |
| IDELAYE2/IDELAYE2_FINEDELAY | 0 | 0 | 300 | 0.00 |
| ODELAYE2/ODELAYE2_FINEDELAY | 0 | 0 | 100 | 0.00 |
| IBUFDS_GTE2 | 0 | 0 | 4 | 0.00 |
| ILOGIC | 0 | 0 | 300 | 0.00 |
| OLOGIC | 0 | 0 | 300 | 0.00 |

5. Clocking
---

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| BUFGCTRL | 1 | 0 | 32 | 3.13 |
| BUFIO | 0 | 0 | 24 | 0.00 |
| MMCME2_ADV | 0 | 0 | 6 | 0.00 |
| PLLE2_ADV | 0 | 0 | 6 | 0.00 |
| BUFMRCE | 0 | 0 | 12 | 0.00 |
| BUFHCE | 0 | 0 | 96 | 0.00 |
| BUFR | 0 | 0 | 24 | 0.00 |

## 7. Primitives

| Ref Name | Used | Functional Category |
|---|---|---|
| LUT6 | 962 | LUT |
| FDRE | 684 | Flop & Latch |
| LUT5 | 465 | LUT |
| LUT2 | 331 | LUT |
| LUT4 | 273 | LUT |
| CARRY4 | 119 | CarryLogic |
| MUXF7 | 86 | MuxFx |
| LUT3 | 75 | LUT |
| IBUF | 34 | IO |
| SRL16E | 32 | Distributed Memory |
| OBUF | 32 | IO |
| LUT1 | 10 | LUT |
| BUFG | 1 | Clock |