

# Lab Assignment – 4

## Mumuksh Jain – 22110160

## Nishant Kumar – 22110170

### Code for Binary Multiplier:

```
`timescale 1ns / 1ps
module binary_multiplier(A,B,out);
parameter n=8;
parameter m=8;
input [n-1:0]A;
input [m-1:0]B;
reg [n+m-1:0]c;
output reg [n+m-1:0]out;
integer i;
integer j;
always @*
begin
    out=0;
    for(i=0;i<m;i=i+1)
    begin
        c=0;
        if(B[i]==1)
        begin
            for(j=0;j<n;j=j+1)
                c[i+j]=A[j];
            out=out+c;
        end
    end
end
endmodule
```

*When using FPGA, change the parameters to  $n=4$ ,  $m=4$*

## ***Code for Array Multiplier:***

```
`timescale 1ns / 1ps
```

```
module fulladd(A,B,O,cout);  
parameter n=8;  
input [n-1:0]A,B;  
output reg [n-1:0]O;  
output reg cout;  
reg [n:0]C;  
integer k;  
  
always @(*)  
begin  
C[0]=0;  
for(k=0;k<n;k=k+1)  
begin  
O[k]=A[k]^B[k]^C[k];  
C[k+1]=(A[k]&B[k])|(A[k]&C[k])|(B[k]&C[k]);  
end  
cout=C[n];  
end  
endmodule
```

```
module shiftright(i,o,c);  
parameter n=8;  
input [n-1:0]i;  
input c;  
output reg [n-1:0]o;  
integer k;  
always@(*)  
begin  
for(k=0;k<n-1;k=k+1)  
begin  
o[k]<=i[k+1];
```

```
end
o[n-1]=c;
end
endmodule
```

```
module sm(a,c,l);
parameter n=8;
input [n-1:0]a;
input c;
output [n-1:0]l;
genvar i;
for(i=0;i<n;i=i+1)
begin:mult1
    and a1([l[i],a[i],c);
end
endmodule
```

```
module arraymulti(a,b,p);
parameter n=8;
input [n-1:0]a;
input [n-1:0]b;
output [2*n-1:0]p;
```

```
wire [n-1:0]m1,m2,m3,m4,m5,m6,m7,m8,m9;
```

```
wire [n-1:0]l1,l2,l3,l4,l5,l6,l7,l8;
```

```
wire [n-1:0]o1,o2,o3,o4,o5,o6,o7,o8;
```

```
wire c1,c2,c3,c4,c5,c6,c7,c8;
```

```
genvar k;
for(k=0;k<n;k=k+1)
begin
```

```
    and a1(m1[k],0,0);  
end
```

```
sm s1(a,b[0],l1);  
fulladd t1(l1,m1,o1,c1);  
assign p[0]=o1[0];  
shiftright j1(o1,m2,c1);
```

```
sm s2(a,b[1],l2);  
fulladd t2(l2,m2,o2,c2);  
assign p[1]=o2[0];  
shiftright j2(o2,m3,c2);
```

```
sm s3(a,b[2],l3);  
fulladd t3(l3,m3,o3,c3);  
assign p[2]=o3[0];  
shiftright j3(o3,m4,c3);
```

```
sm s4(a,b[3],l4);  
fulladd t4(l4,m4,o4,c4);  
assign p[3]=o4[0];  
shiftright j4(o4,m5,c4);
```

```
sm s5(a,b[4],l5);  
fulladd t5(l5,m5,o5,c5);  
assign p[4]=o5[0];  
shiftright j5(o5,m6,c5);
```

```
sm s6(a,b[5],l6);  
fulladd t6(l6,m6,o6,c6);  
assign p[5]=o6[0];  
shiftright j6(o6,m7,c6);
```

```
sm s7(a,b[6],l7);
```

```

fulladd t7(l7,m7,o7,c7);
assign p[6]=o7[0];
shiftright j7(o7,m8,c7);

sm s8(a,b[7],l8);
fulladd t8(l8,m8,o8,c8);
assign p[7]=o8[0];
shiftright j8(o8,m9,c8);

genvar h;
for (h=0;h<n;h=h+1)
begin
assign p[n+h]=m9[h];
end

endmodule

```

## Testbench:

### *Binary Multiplier:*

```

`timescale 1ns / 1ps
module bin_mul_tb();
reg [8:0]A;
reg [8:0]B;
wire [15:0]out;
binary_multiplier uut(A,B,out);
initial
begin
A=8'b01010111;
B=8'b10101100;
#5;
A=8'b01110110;
B=8'b10110001;

```

```

#5;
A=8'b00001101;
B=8'b11001101;
#5;
A=8'b11111100;
B=8'b11110101;
#5;
$finish();
end
endmodule

```

### *Array Multiplier:*

```

`timescale 1ns / 1ps
module arraymulti_tb();
reg [8-1:0]a;
reg [8-1:0]b;
wire [2*8-1:0]p;

integer i,j;
arraymulti uut(a,b,p);
initial
begin

for (i=0;i<256;i=i+1)
begin
for(j=0;j<256;j=j+1)
begin
a=i;
b=j;
#10;
end
end

$finish();

```

```
end  
endmodule
```

## Constraint File:

### *Binary Multiplier:*

```
set_property IOSTANDARD LVCMOS33 [get_ports {A[3]}]  
set_property IOSTANDARD LVCMOS33 [get_ports {A[2]}]  
set_property IOSTANDARD LVCMOS33 [get_ports {A[1]}]  
set_property IOSTANDARD LVCMOS33 [get_ports {A[0]}]  
set_property IOSTANDARD LVCMOS33 [get_ports {B[3]}]  
set_property IOSTANDARD LVCMOS33 [get_ports {B[2]}]  
set_property IOSTANDARD LVCMOS33 [get_ports {B[1]}]  
set_property IOSTANDARD LVCMOS33 [get_ports {B[0]}]  
set_property IOSTANDARD LVCMOS33 [get_ports {out[7]}]  
set_property IOSTANDARD LVCMOS33 [get_ports {out[6]}]  
set_property IOSTANDARD LVCMOS33 [get_ports {out[5]}]  
set_property IOSTANDARD LVCMOS33 [get_ports {out[4]}]  
set_property IOSTANDARD LVCMOS33 [get_ports {out[3]}]  
set_property IOSTANDARD LVCMOS33 [get_ports {out[2]}]  
set_property IOSTANDARD LVCMOS33 [get_ports {out[1]}]  
set_property IOSTANDARD LVCMOS33 [get_ports {out[0]}]  
  
set_property PACKAGE_PIN R2 [get_ports {A[3]}]  
set_property PACKAGE_PIN T1 [get_ports {A[2]}]  
set_property PACKAGE_PIN U1 [get_ports {A[1]}]  
set_property PACKAGE_PIN W2 [get_ports {A[0]}]  
set_property PACKAGE_PIN W17 [get_ports {B[3]}]  
set_property PACKAGE_PIN W16 [get_ports {B[2]}]  
set_property PACKAGE_PIN V16 [get_ports {B[1]}]  
set_property PACKAGE_PIN V17 [get_ports {B[0]}]  
set_property PACKAGE_PIN L1 [get_ports {out[7]}]  
set_property PACKAGE_PIN P1 [get_ports {out[6]}]  
set_property PACKAGE_PIN N3 [get_ports {out[5]}]  
set_property PACKAGE_PIN P3 [get_ports {out[4]}]  
set_property PACKAGE_PIN U3 [get_ports {out[3]}]
```

```
set_property PACKAGE_PIN W3 [get_ports {out[2]]}
set_property PACKAGE_PIN V3 [get_ports {out[1]]}
set_property PACKAGE_PIN V13 [get_ports {out[0]]}
```

### *Array Multiplier:*

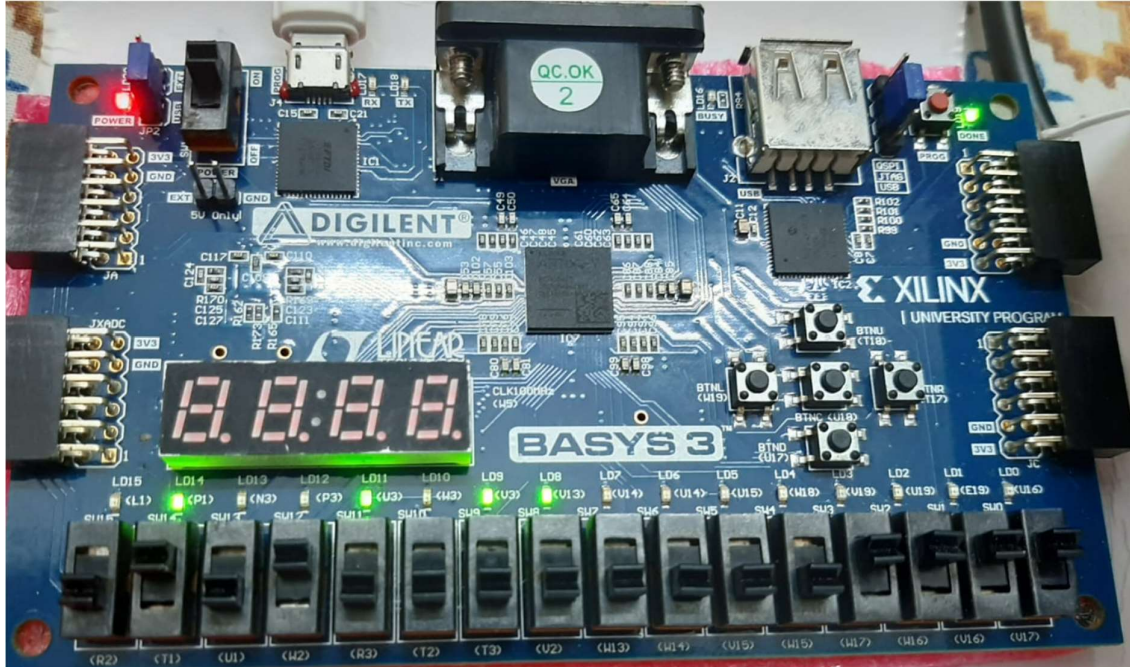
```
set_property IOSTANDARD LVCMOS33 [get_ports {a[3]]}
set_property IOSTANDARD LVCMOS33 [get_ports {a[2]]}
set_property IOSTANDARD LVCMOS33 [get_ports {a[1]]}
set_property IOSTANDARD LVCMOS33 [get_ports {a[0]]}
set_property IOSTANDARD LVCMOS33 [get_ports {b[3]]}
set_property IOSTANDARD LVCMOS33 [get_ports {b[2]]}
set_property IOSTANDARD LVCMOS33 [get_ports {b[1]]}
set_property IOSTANDARD LVCMOS33 [get_ports {b[0]]}
set_property IOSTANDARD LVCMOS33 [get_ports {p[2]]}
set_property IOSTANDARD LVCMOS33 [get_ports {p[3]]}
set_property IOSTANDARD LVCMOS33 [get_ports {p[4]]}
set_property IOSTANDARD LVCMOS33 [get_ports {p[5]]}
set_property IOSTANDARD LVCMOS33 [get_ports {p[7]]}
set_property IOSTANDARD LVCMOS33 [get_ports {p[6]]}
set_property IOSTANDARD LVCMOS33 [get_ports {p[1]]}
set_property IOSTANDARD LVCMOS33 [get_ports {p[0]]}
set_property PACKAGE_PIN R2 [get_ports {a[3]]}
set_property PACKAGE_PIN T1 [get_ports {a[2]]}
set_property PACKAGE_PIN U1 [get_ports {a[1]]}
set_property PACKAGE_PIN W2 [get_ports {a[0]]}
set_property PACKAGE_PIN W17 [get_ports {b[3]]}
set_property PACKAGE_PIN W16 [get_ports {b[2]]}
set_property PACKAGE_PIN V16 [get_ports {b[1]]}
set_property PACKAGE_PIN V17 [get_ports {b[0]]}
set_property PACKAGE_PIN L1 [get_ports {p[7]]}
set_property PACKAGE_PIN P1 [get_ports {p[6]]}
set_property PACKAGE_PIN N3 [get_ports {p[5]]}
set_property PACKAGE_PIN P3 [get_ports {p[4]]}
set_property PACKAGE_PIN U3 [get_ports {p[3]]}
```



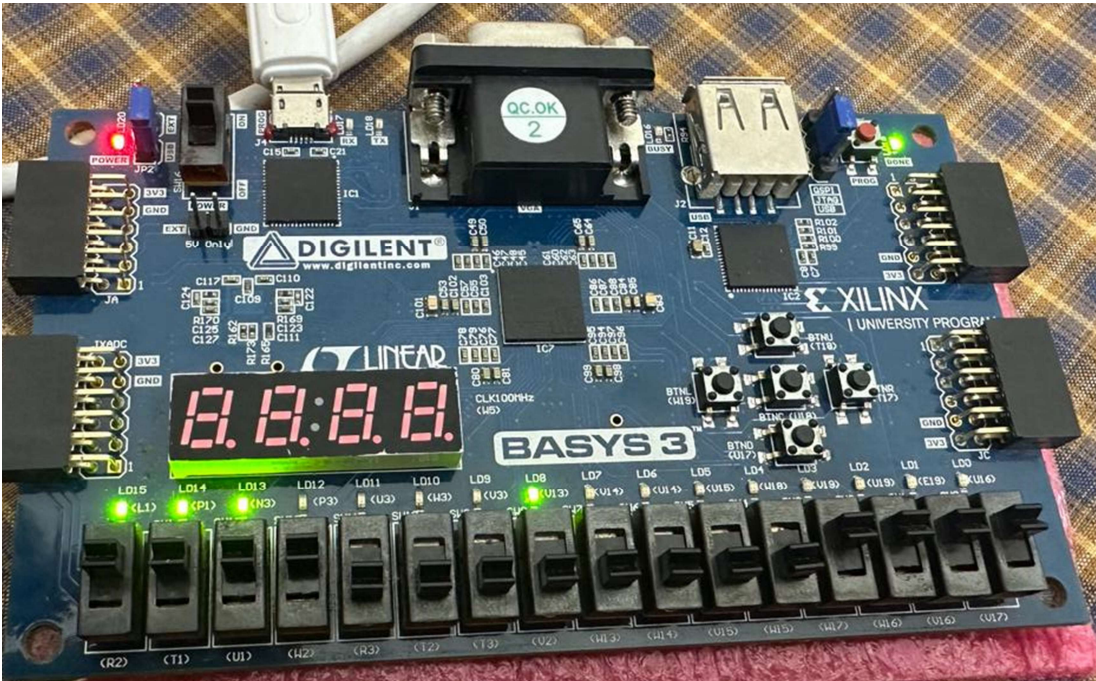
```
set_property PACKAGE_PIN W3 [get_ports {p[2]}]
set_property PACKAGE_PIN V3 [get_ports {p[1]}]
set_property PACKAGE_PIN V13 [get_ports {p[0]}]
```

## Photos:

*Binary Multiplier:*






Array Multiplier:

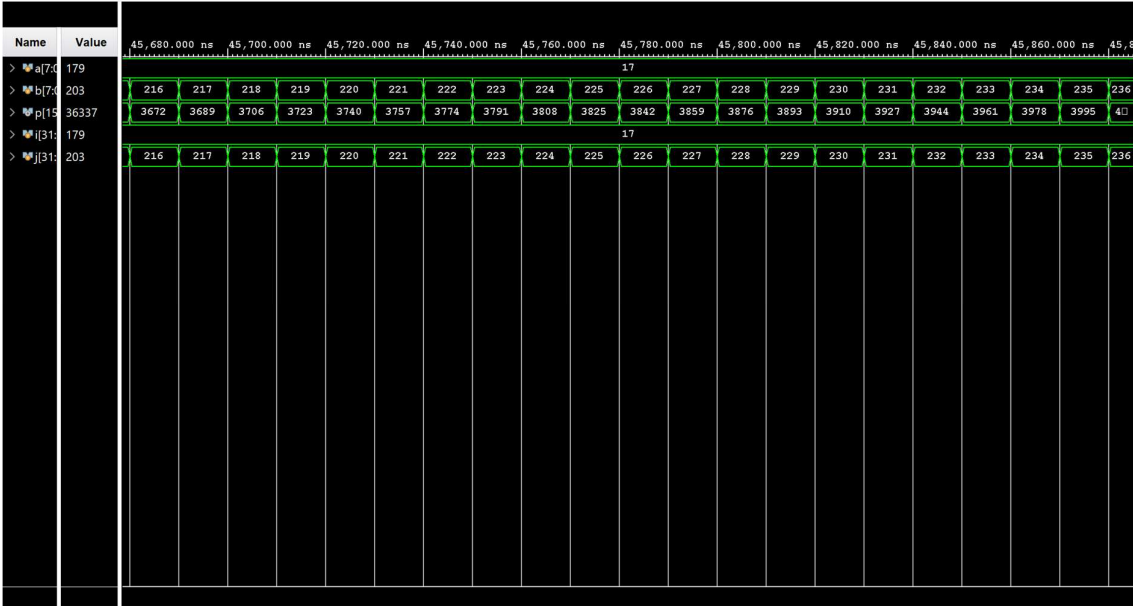
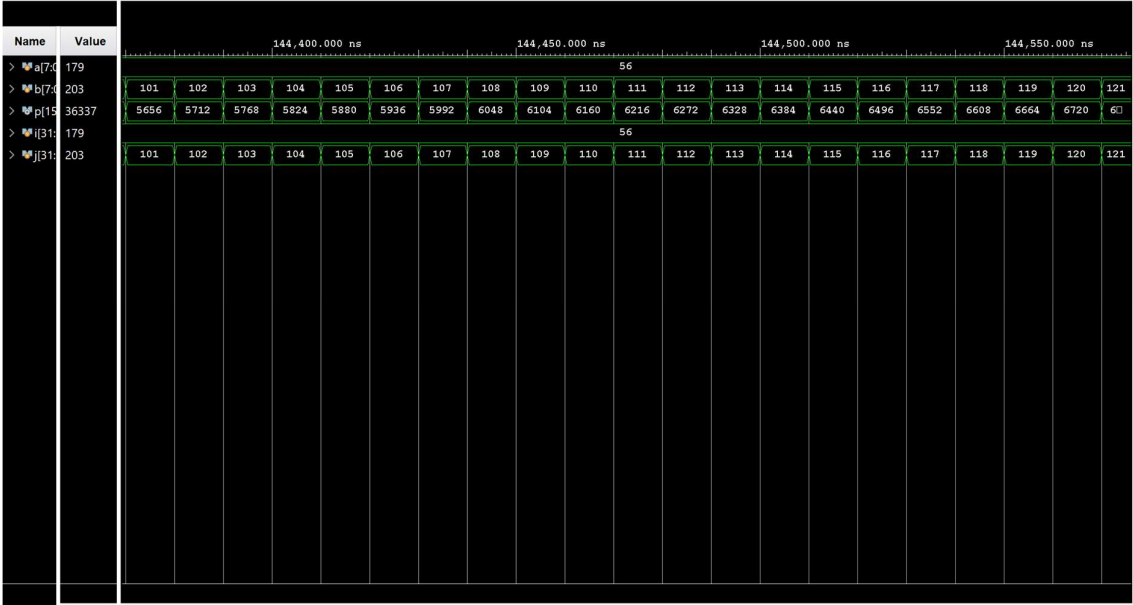


Simulation:

Binary Multiplier:

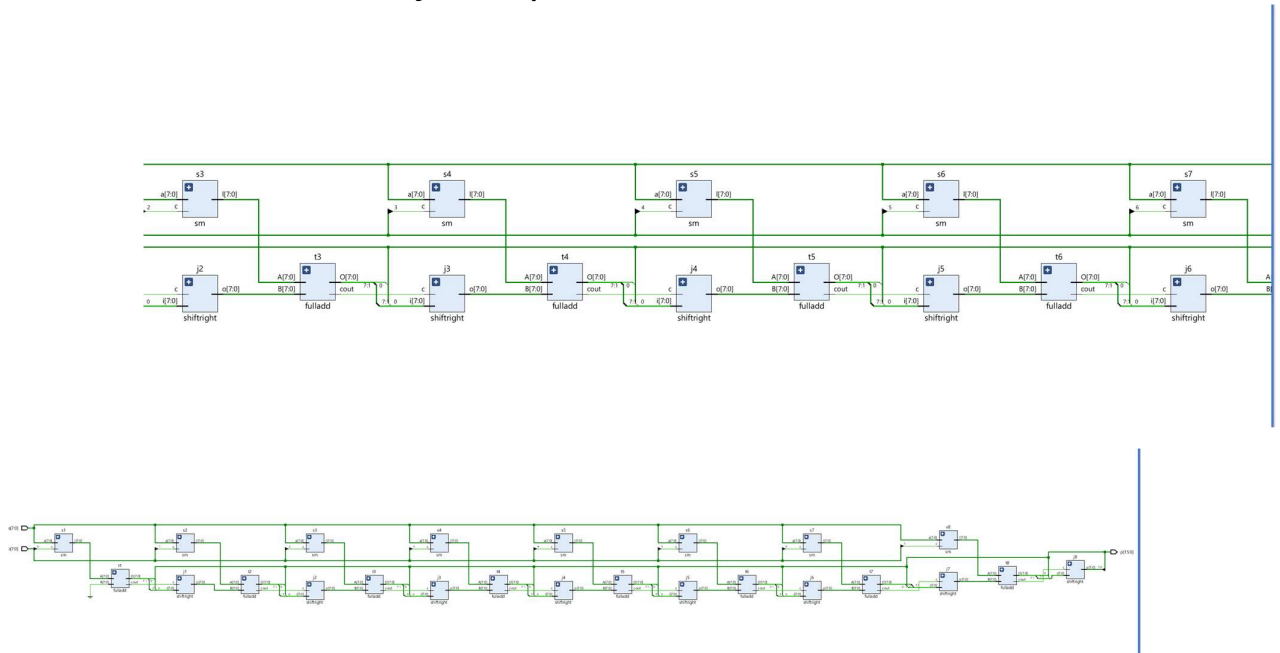
Name		Value				
		0.000 ns	5.000 ns	10.000 ns	15.000 ns	
>	 A[8:0]	252	87	118	13	252
>	 B[8:0]	245	172	177	205	245
>	 out[15:0]	61740	14964	20886	2665	61740

Array Multiplier:

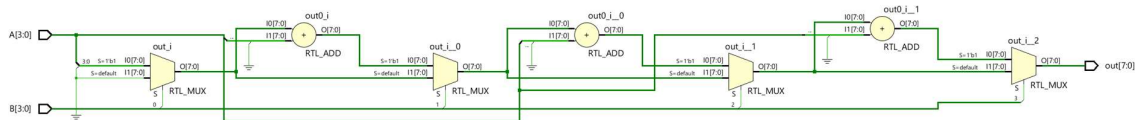


## Comparison between Array Multiplier and Binary Multiplier:

### A. Schematic Array Multiplier



### B. Schematic Binary Multiplier:



### Binary Multiplier:

A binary multiplier is a straightforward and basic circuit that performs binary multiplication using the standard method taught in elementary school.

It involves multiplying each bit of one binary number by each bit of the other binary number and adding up the results.

### Array Multiplier:

An array multiplier is a more sophisticated and efficient multiplication circuit.

*It uses an array of AND gates to generate partial products and then adds these partial products together to get the final result.*

*The array structure allows for parallel processing of the multiplication, making it faster than a simple binary multiplier.*

*The size of the array is determined by the number of bits in the binary numbers being multiplied.*

### Binary Multiplier:

Site Type	Used	Fixed	Available	Util%
Slice LUTs	16	0	20800	0.08
LUT as Logic	16	0	20800	0.08
LUT as Memory	0	0	9600	0.00
Slice Registers	0	0	41600	0.00
Register as Flip Flop	0	0	41600	0.00
Register as Latch	0	0	41600	0.00
F7 Muxes	0	0	16300	0.00
F8 Muxes	0	0	8150	0.00

On-Chip	Power (W)	Used	Available	Utilization (%)
Slice Logic	0.076	22	---	---
LUT as Logic	0.067	16	20800	0.08
CARRY4	0.009	2	8150	0.02
Others	0.000	2	---	---
Signals	0.203	26	---	---
I/O	13.795	16	106	15.09
Static Power	0.234			
Total	14.309			



## Array Multiplier:

Site Type	Used	Fixed	Available	Util%
Slice LUTs	38	0	20800	0.18
LUT as Logic	38	0	20800	0.18
LUT as Memory	0	0	9600	0.00
Slice Registers	0	0	41600	0.00
Register as Flip Flop	0	0	41600	0.00
Register as Latch	0	0	41600	0.00
F7 Muxes	0	0	16300	0.00
F8 Muxes	0	0	8150	0.00

On-Chip	Power (W)	Used	Available	Utilization (%)
Slice Logic	0.297	49	---	---
LUT as Logic	0.297	38	20800	0.18
Others	0.000	1	---	---
Signals	0.893	60	---	---
I/O	17.395	28	106	26.42
Static Power	0.428			
Total	19.013			