



南京大學
NANJING UNIVERSITY

《VLSI-DSPS 设计与实现》课程项目

项目报告

(2024 – 2025 秋季学期)

姓名 : 张乐天

学号 : 221900182

专业 : 集成电路设计与集成系统



1 项目简介

乘法器的设计对于计算机系统结构底层运算架构十分重要。本项目基于 Booth 编码做了 32 位乘法器的设计，通过比较基 2 与基 4 两种 Booth 编码设计的乘法器理解超大规模集成电路（VLSI）中位级运算架构优化方式。

2 乘法器描述

本项目设计了一种支持 32 位原码无符号数与 32 位补码有符号数两种情况的乘法运算单元 `mul`，网表接口如下

```
module mul
(
    input wire mul_clk,           // 时钟信号
    input wire resetn,            // 复位信号，低电平有效
    input wire run,               // 开始运算信号
    input wire mul_signed,        // 有符号乘法标志
    input wire [31:0] x,          // 
    input wire [31:0] y,          // 
    output reg [63:0] result,      // 完成信号
    output reg complete
);

```

其中，`run` 信号作为启动运算信号。`mul_signed` 为 1 则表明输入为 32 位补码，否则位 32 位无符号数。输出为 64 位输出，可以满足所需要的输出范围。`complete` 为完成信号，会持续至少一个周期。

3 乘法的 Booth 算法

Booth 算法针对补码做运算，可以同时对符号位与数值位做运算。为了同时满足上述两种情况，需要对输入 32 位数做扩展，基本原理如下，具体方式在设计中体现。

```
reg signed [32:0] x_e = {{mul_signed & x_e[31]}, x};
```

3.1 基 2 算法

3.1.1 原理

假设两 n 位有符号数 X 、 Y 补码分别为 $[X]_{comp}$ 、 $[Y]_{comp}$ ，则 $[X]_{comp} = x_{n-1} \cdots x_1 x_0$ 、 $[Y]_{comp} = y_{n-1} \cdots y_1 y_0$ 。不难得到 Y 的值可表示为（令 $y_{-1} = 0$ 为辅助位）

$$Y = -y_{n-1} 2^{n-1} + \sum_{i=0}^{n-2} y_i 2^i = \sum_{i=0}^{n-1} (y_{i-1} - y_i) 2^i$$

这样乘积 $[X \times Y]_{comp}$ 可以表示为

$$[X \times Y]_{comp} = \left[X \times \sum_{i=0}^{n-1} (y_{i-1} - y_i) 2^i \right]_{comp}$$

考虑到小数点位置不影响运算，因此不妨计算下式

$$\left[X \times \frac{Y}{2^n} \right]_{comp} = \left[X \times \sum_{i=0}^{n-1} (y_{i-1} - y_i) 2^{i-n} \right]_{comp}$$

展开上式子，可以得到递推公式

$$[P_{i+1}]_{comp} = [2^{-1}(P_i + X(y_{i-1} - y_i))]_{comp}$$

其中， P_i 为上次的部分积， P_{i+1} 为本次的部分积。令 $P_0 = 0$ ，可以得到 $[X \times Y]_{comp} = [P_n]_{comp}$ 。

由递推公式，可以知道，求得 $[P_i]_{comp}$ 后，根据 $y_i y_{i-1}$ 可以给出 $[P_{i+1}]_{comp}$ 如下

$$[P_{i+1}]_{comp} = \begin{cases} [2^{-1}(P_i + X)]_{comp}, & (y_i y_{i-1} = 01) \\ [2^{-1}(P_i - X)]_{comp}, & (y_i y_{i-1} = 10) \\ [2^{-1}P_i]_{comp}, & (y_i y_{i-1} = 00 \text{ or } 11) \end{cases}$$

而 $[2^{-1}(P_i \pm X)]_{comp}$ 可以通过执行 $[P_i]_{comp} + [\pm X]_{comp}$ 后算术右移一位实现。重复 n 次后可以得到最终结果 $[X \times Y]_{comp}$ 。

实现时，需要一 $(2n + 1)$ 的寄存器，先将 Y 装入低 $(n + 1)$ 位（包括辅助位），之后在高位做运算，之后整体右移即可。

3.1.2 硬件描述实现

首先定义相关寄存器，采用了 67 位，是因为符号扩展会使得两个 33 位数，加上辅助位总计 67 位。为了方便做 $[P_i]_{comp} + [\pm X]_{comp}$ 运算，直接使用 67 位的 X 。

```
reg signed [66:0] x_e;
reg signed [66:0] x_inv;
reg signed [66:0] temp;
```

定义操作次数寄存器，一共需要位移 33 次。

```
reg [5:0] cnt;
```

过程语句如下

```
always @(posedge mul_clk or negedge resetn)
```

下面为全部的过程块，下逐步分析。

首先是复位情况，复位信号为低电平有效，需要全部清零。

```
if (!resetn) begin
    complete <= 0;
```

```

x_e <= 67'd0;
x_inv <= 67'd0;
temp <= 67'd0;
cnt <= 6'd0;
end

```

非复位情况下，又有正常运行与非运行情况。

如果正常允许，`cnt` 会递增，并且在第一个周期（`cnt` 为 0）时，将 `x`、`y` 做相应处理填入 `e_x`、`e_inv` 与 `temp` 三个寄存器。如果计数到 34，说明完成计算（第一个周期预处理，一共移位 33 次，故需要 34 个周期），实际上由于使用非阻塞赋值，结合 `complete` 的高电平持续，一共一次计算全部耗时 36 个周期（可以由 `tb` 文件测量）。如果未完成计算，会根据 `temp` 的低两位，即 $y_i y_{i-1}$ ，进行高位的运算以及算术移位（用 '`>>>`' 实现）。

```

cnt <= cnt + 1;
if (cnt == 0) begin
    x_e <= {mul_signed & x[31], x, 34'd0};
    temp <= {{mul_signed & y[31]}, y, 1'b0};
    x_inv <= ~{mul_signed & x[31], x, 34'd0} + 1;
end else if (!complete) begin
    case (temp[1:0])
        2'b01: begin
            temp <= (temp + x_e) >>> 1;
        end
        2'b10: begin
            temp <= (temp + x_inv) >>> 1;
        end
        default: temp <= temp >>> 1;
    endcase
end
if (cnt == 34) begin
    complete <= 1;
    result <= temp[64:1];
    temp <= 66'd0;
end

```

如果没有正常运行，那么做类似复位的处理即可。

```

complete <= 0;
result <= 64'd0;
x_e <= 67'd0;
x_inv <= 67'd0;
cnt <= 6'd0;

```

全部代码见附录。

3.2 基 4 算法

与基 2 算法类似，在此略去推导，仅仅给出表格。

表 基 4 编码

y_{2i+1}	y_{2i}	y_{2i-1}	部分积 P_i
0	0	0	0
0	0	1	$+X$
0	1	0	$+X$
0	1	1	$+2X$
1	0	0	$-2X$
1	0	1	$-X$
1	1	0	$-X$
1	1	1	0

不难得到对应的代码如下。

```
// 69 位寄存器，因为基 4 需要计算 $\pm 2Y$ ，同时需要再一次扩展符号位
reg signed [68:0] x_e;
reg signed [68:0] x_inv;
reg signed [68:0] temp;
// 操作次数寄存器
reg [4:0] cnt;
always @(posedge mul_clk or negedge resetn) begin
    if (!resetn) begin // 复位操作
        complete <= 0;
        x_e <= 69'd0;
        x_inv <= 69'd0;
        temp <= 69'd0;
        cnt <= 5'd0;
    end else begin
        if (run) begin
            cnt <= cnt + 1; // 正常运行时 cnt 递增
            if (cnt == 0) begin
                x_e <= {{2{mul_signed & x[31]}}, x, 35'd0};
                temp <= {{2{mul_signed & y[31]}}, y, 1'b0}; // 扩展两次符号位
                x_inv <= ~{{2{mul_signed & x[31]}}, x, 35'd0} + 1;
            end else if (!complete) begin // 计算与移位
                case (temp[2:0])
                    3'b001, 3'b010: temp <= (temp + x_e) >>> 2;
                    3'b011: temp <= (temp + (x_e <<< 1)) >>> 2;
                    3'b100: temp <= (temp + (x_inv <<< 1)) >>> 2;
            end
        end
    end
end
```

```

3'b101,3'b110: temp <= (temp + x_inv) >>> 2;
default: temp <= temp >>> 2;
endcase
end
if (cnt == 18) begin // 计算完成
    complete <= 1;
    result <= temp[64:1];
    temp <= 69'd0;
end
end else begin // 非正常运行时做复位操作
    complete <= 0;
    result <= 64'd0;
    x_e <= 69'd0;
    x_inv <= 69'd0;
    cnt <= 5'd0;
end
end
end

```

4 仿真 (Simulation)

这里采用专门用于仿真的 Questa Sim 软件做仿真，仿真结果均正确，说明功能性结果正确，相关 `testbench` 见附件。

仿真截图如图 1、图 2。

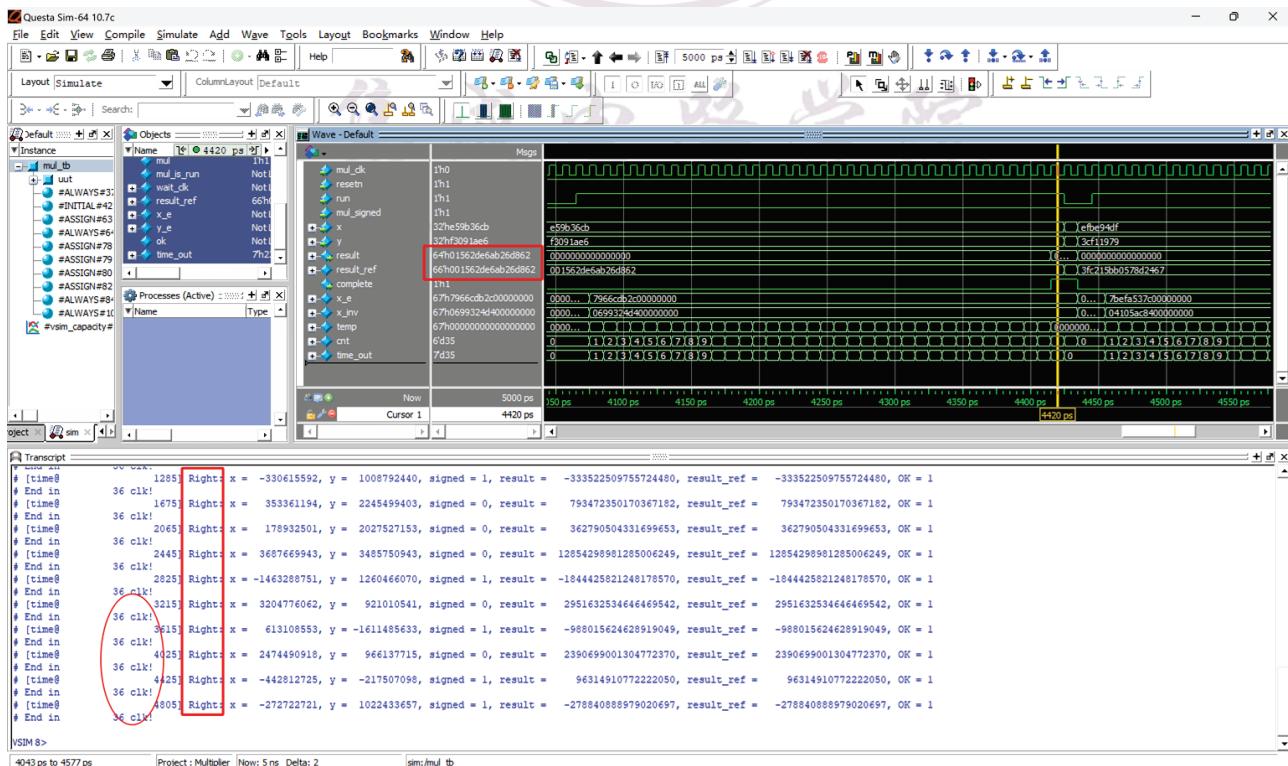


图 1 基 2 算法仿真

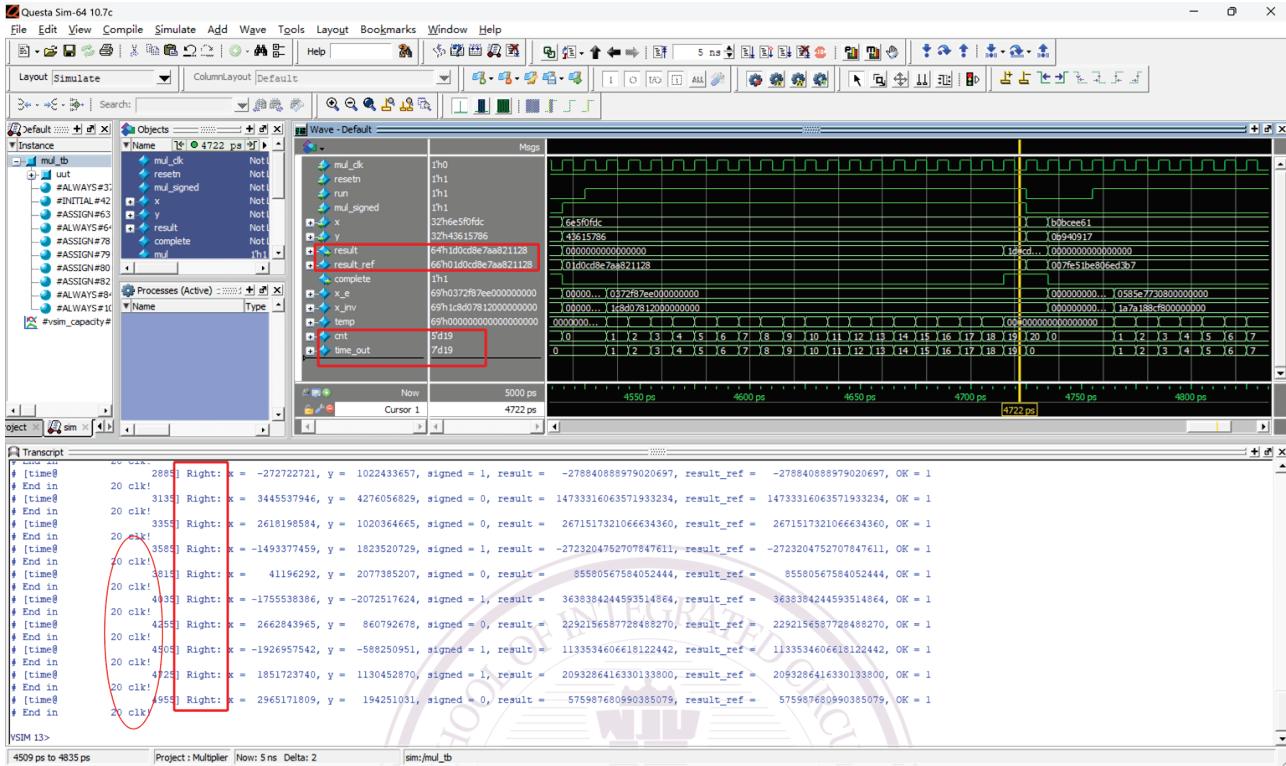


图 2 基 4 算法仿真

5 综合与实现 (Synthesis and Implementation)

由于并不能直接对乘法器做综合设计，这样意义不大，故将其放置在一个小型 CPU 的模块内。外接模块从 RAM 调取数据交至乘法器做计算。板卡型号为 xc7a200tfgbg676-2。

在 Vivado 中建立相关文件如图 3 所示，为了方便，将两个算法均写入 `multiplier.v` 文件中，使用条件编译指令即可，详情见附件。

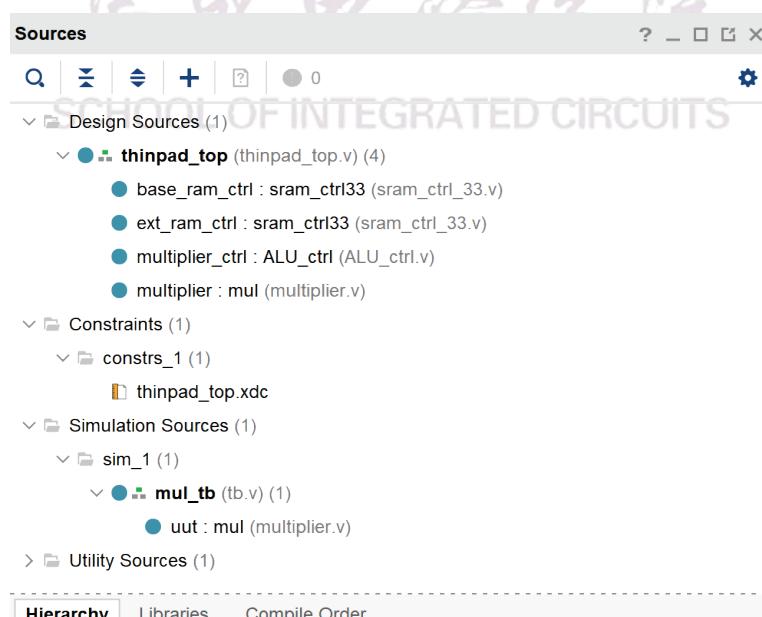


图 3 工程结构

考虑到两种设计的运算周期不同，需要更改相应的时钟，在 `xdc` 文件中做 `clk` 的设定。由于周期比为 $36:20 = 9:5$ ，因此时周期比为 $5:9$ 。

```
# Algorithm_1 booth radix-2
create_clock -period 5.0 -name clk_200M -waveform {0.0 2.5} [get_ports clk_main]
# Algorithm_2 booth radix-4
create_clock -period 9.0 -name clk_111M111 -waveform {0.0 4.5} [get_ports clk_main]
```

实际使用时，仅需设置一个时钟即可，另一个需要注释掉。

5.1 基 2 算法

如图，时序没有违例情况。

Design Timing Summary

Setup	Hold	Pulse Width
Worst Negative Slack (WNS): 0.583 ns	Worst Hold Slack (WHS): 0.152 ns	Worst Pulse Width Slack (WPWS): 2.000 ns
Total Negative Slack (TNS): 0.000 ns	Total Hold Slack (THS): 0.000 ns	Total Pulse Width Negative Slack (TPWS): 0.000 ns
Number of Failing Endpoints: 0	Number of Failing Endpoints: 0	Number of Failing Endpoints: 0
Total Number of Endpoints: 1229	Total Number of Endpoints: 1229	Total Number of Endpoints: 611

All user specified timing constraints are met.

图 4 基 2 设计时序

整体功耗为 0.197 W，其中动态功耗 0.058 W。

Summary

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power: 0.197 W
Design Power Budget: Not Specified
Process: typical
Power Budget Margin: N/A
Junction Temperature: 25.4°C
Thermal Margin: 59.6°C (31.6 W)
Ambient Temperature: 25.0 °C
Effective θJA: 1.9°C/W
Power supplied to off-chip devices: 0 W
Confidence level: Low
[Launch Power Constraint Advisor](#) to find and fix invalid switching activity

On-Chip Power

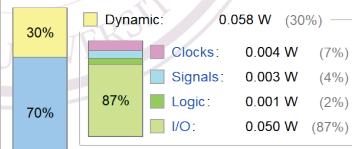


图 5 基 2 设计总功耗

需要关注乘法器的功耗，如图，动态功耗为 0.003 W。

Utilization	Name	Clocks (W)	Signals (W)	Data (W)	Clock Enable (W)	Set/Reset (W)	Logic (W)	I/O (W)
0.058 W (30% of total)	N_thinpad_top							
0.046 W (23% of total)	Leaf Cells (197)							
0.003 W (2% of total)	multiplier (mul)	0.001	0.001	0.001	<0.001	<0.001	0.001	<0.001
0.003 W (2% of total)	Leaf Cells (539)							
0.003 W (11% of total)	multiplier_ctrl (ALU_ctrl)	0.002	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
0.001 W (1% of total)	base_ram_ctrl (srram_ctrl33)	0.001	0.001	0.001	<0.001	<0.001	<0.001	<0.001
0.001 W (1% of total)	ext_ram_ctrl (srram_ctrl33_0)	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
<0.001 W (<1% of total)	base_ram_data_Iobuf[1]_inst (IOBUF)	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
<0.001 W (<1% of total)	base_ram_data_Iobuf[2]_inst (IOBUF)	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
<0.001 W (<1% of total)	base_ram_data_Iobuf[3]_inst (IOBUF)	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
<0.001 W (<1% of total)	base_ram_data_Iobuf[4]_inst (IOBUF)	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
<0.001 W (<1% of total)	base_ram_data_Iobuf[5]_inst (IOBUF)	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
<0.001 W (<1% of total)	base_ram_data_Iobuf[6]_inst (IOBUF)	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001

图 6 基 2 设计乘法器动态功耗

5.2 基 4 算法

同样，无时序违例。

Design Timing Summary

Setup	Hold	Pulse Width
Worst Negative Slack (WNS): 4.055 ns	Worst Hold Slack (WHS): 0.141 ns	Worst Pulse Width Slack (WPWS): 4.000 ns
Total Negative Slack (TNS): 0.000 ns	Total Hold Slack (THS): 0.000 ns	Total Pulse Width Negative Slack (TPWS): 0.000 ns
Number of Failing Endpoints: 0	Number of Failing Endpoints: 0	Number of Failing Endpoints: 0
Total Number of Endpoints: 1229	Total Number of Endpoints: 1229	Total Number of Endpoints: 611

All user specified timing constraints are met.

图 7 基 3 设计时序

总功耗情况如下，动态功耗为 0.032 W，降低了 44.8%。

Summary

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:

0.171 W

Design Power Budget:

Not Specified

Process:

typical

Power Budget Margin:

N/A

Junction Temperature:

25.3°C

Thermal Margin:

59.7°C (31.6 W)

Ambient Temperature:

25.0 °C

Effective 9JA:

1.9°C/W

Power supplied to off-chip devices: 0 W

Confidence level: Low

[Launch Power Constraint Advisor](#) to find and fix invalid switching activity

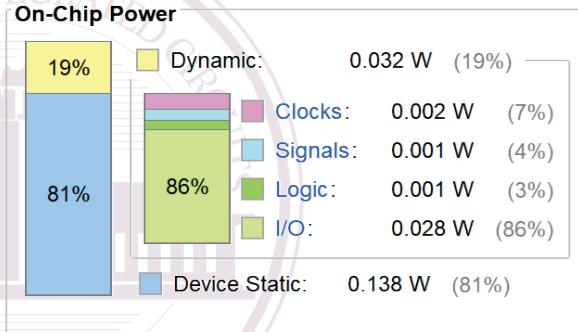


图 8 基 4 设计总功耗

其中，乘法器的动态功耗为 0.002W，有所降低，降低约 33.3%。

Hierarchical								
Utilization	Name	Clocks (W)	Signals (W)	Data (W)	Clock Enable (W)	Set/Reset (W)	Logic (W)	I/O (W)
0.032 W (19% of total)	thinpad_top							
0.026 W (15% of total)	Leaf Cells (197)							
0.002 W (1% of total)	multiplier (mul)	0.001	0.001	0.001	<0.001	<0.001	0.001	<0.001
0.002 W (1% of total)	Leaf Cells (539)							
> 0.001 W (1% of total)	multiplier_ctrl (ALU_ctrl)	0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
> 0.001 W (1% of total)	base_ram_ctrl (sram_ctrl33)	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
> 0.001 W (<1% of total)	ext_ram_ctrl (sram_ctrl33_0)	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
> 0.001 W (<1% of total)	base_ram_data_Iobuf[1]_inst (Iobuf)	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
> 0.001 W (<1% of total)	base_ram_data_Iobuf[2]_inst (Iobuf)	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
> 0.001 W (<1% of total)	base_ram_data_Iobuf[3]_inst (Iobuf)	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
> 0.001 W (<1% of total)	base_ram_data_Iobuf[4]_inst (Iobuf)	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
> 0.001 W (<1% of total)	base_ram_data_Iobuf[5]_inst (Iobuf)	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
> 0.001 W (<1% of total)	base_ram_data_Iobuf[6]_inst (Iobuf)	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
> 0.001 W (<1% of total)	base_ram_data_Iobuf[7]_inst (Iobuf)	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001

图 9 基 4 设计乘法器动态功耗

6 总结

可以看到，其实乘法器的功耗本身已经很低。

由于基 4 设计需要更少的周期数，因此，时钟频率降低，降低了 44.4%，如果根据

$$P_{dynamic} \propto CV^2f$$

做估算，可以得知，功耗也应当降低 44.4%，这与电路的总动态功耗的减少是相一致的。

但显然，具体的功耗已经在小数点后 4 位，Vivado 无法给出更为精确的估计，但仍然可以得知，降低了 0.001 W 左右。结合电路内部寄存器数量的变化与逻辑的变化，这是可以接受的。



集成电路学院

SCHOOL OF INTEGRATED CIRCUITS

附 录

工具与环境：

Windows 64bit

Vivado : 2024.2

Questa Sim : 10.7

相应代码文件在 Vivado 工程中

VLSI-DSP\Soc_Multiplier\Soc_Multiplier.srcs

其中，**\sources_1** 内为工程源码，**sim_1** 内为 **tb**，**constrs_1** 内为 **xdc** 约束文件。

为方便查阅，**VLSI-DSP>MainSrc** 中给出了上述的乘法器模块与 **tb** 文件，**VLSI-DSP\Report** 中给出了两种设计对应生成的全部报告。



集成 电 路 学 院

SCHOOL OF INTEGRATED CIRCUITS