数字信号处理的 VLSI 设计

$GF(2^m)$ 域上的 Digit-Serial 脉动结构（Systolic）的乘法器

小组成员：

顾晨昊： 17212020011

叶汉辰： 17212020116

张秉异： 17212020123

指导老师：

曾晓洋

目录

1. 设计要求

1.1 目标

设计一个GF(2^m)域上的 Digit-Serial 脉动结构（Systolic）的乘法器。

1.2 有限域概念

令 A(x)、B(x)为有限域上GF(2^m)的两个元素；G(x)为阶为 的不可约多项式，它是域生成多项式； p(x) = $A(x)B(x)\ mod\ G(x)$。其中每个多项式的表达形式如下：

$$A(x) = a_{m-1}x^{m-1} + a_{m-2}{}^{m-2} + \cdots + a_1 x + a_0$$

$$B(x) = b_{m-1}x^{m-1} + b_{m-2}{}^{m-2} + \cdots + b_1 x + b_0$$

$$G(x) = x^m + g_{m-1}x^{m-1} + g_{m-2}{}^{m-2} + \cdots + g_1 x + g_0$$

$$P(x) = p_{m-1}x^{m-1} + p_{m-2}{}^{m-2} + \cdots + p_1 x + p_0$$

其中，各多项式的系数都在有限域$GF(2)$上[1]。

1.3 有限域GF(2^m)的高位优先的乘法算法

Algorithm for MSB-first Multiplication in $GF(2^m)$[2]

Input: $A(x)$, $B(x)$, $G(x)$

Output: $P(x) = A(x)B(x) \bmod G(x)$

$t_j^{(0)} = 0$, $0 \le j \le m-1$

$t_{-1}^{(i)} = 0$, $1 \le i \le m-1$

for $i = 1$ to m do

  for $j = m-1$ **to** 0 **do**

    $t_j^{(i)} = t_{m-1}^{(i-1)} g_j + b_{m-i} a_j + t_{j-1}^{(i-1)}$

$P(x) = T^m(x)$

[1] 各项系数在 GF(2)域上也就是说，系数从{0,1}之中取值。

[2] 有限域GF(2^m)的高位优先的乘法算法的伪代码。

1.4 设计要求

①设计有限域$GF(2^{163})$域上的基于脉动（Systolic）阵列的 Digit-Serial 乘法器，要求 digit 分成 8，16，32 比特三种情况。

②其中，$GF(2^{163})$域上的生成多项式为：$G(x)=x^{163}+x^7+x^6+x^3+1$

2. 算法分析

2.1 有限域GF(2^m)的概念

有限域亦称伽罗瓦域（galois field），是仅含有限个元素的域，它是伽罗瓦（Galois,E）于 18 世纪 30 年代研究代数方程求根公式求解问题时引出的，有限域的特征数必为某一素数 p，因此它的含素域同构域 Zp。若 F 是特征为 p 的有限域，则 F 中元素的个数为p^n、n 为某一正整数，元素个数相同的有限域是同构的。因此，通常用GF(P^n)的乘法群是(p^n-1)阶的循环群。

下面引入群和域的概念：

定义 1：群的概念

设 G 是一个非空集合，若在 G 上定义一个二元运算 · 满足：

（1）结合律：对任何$a,b,c\in G$有$(a\cdot b)\cdot c=a\cdot(b\cdot c)$，则称 G 是半群，记作(G,·):若（G，∘）还满足：

（2）存在单位元 e 使得任何$a\in G$,有$e\cdot a=a\cdot e=a$

（3）对任何$a\in G$有逆元a^{-1}使$a^{-1}\cdot a=a\cdot a^{-1}=e$,称(G,·)是一个群

如果半群中也有也有单位元，则称含幺半群，如果群(G,·)适合交换律，对任何$a,b\in G$有$a\cdot b=b\cdot a$,则称 G 为可交换或阿贝尔（Abel）群。

把群的定义概括为 4 点：封闭性、结合性、单位元和逆元。

定义 2：域的概念

对于非空元素集合 F 及定义在其上的二元运算“+”“·”，若满足

（1）$<F,+>$为阿贝尔群，F 在“+”运算下的幺元记为 0；

（2）F 中所有的非 0 元素在“·”运算下构成阿贝尔群，其幺元记为 1；

（3）“·”运算“+”运算满足分配率：$a\cdot(b+c)=a\cdot b+a\cdot c$,其中$a,b,c\in F$。

那么称代数系统$<F,+,\cdot>$为域(Field)。如果集合 F 为有限集，则称$<F,+,\cdot>$是有限域。

我们给出一个简单的例子:$F=\{0,1,2,3,4,5,6,7,8\}$。F 在模 8 加运算下构成阿贝尔群，对应幺元记为 0。同时 F 的非 0 元素在模 8 乘下也构成阿贝尔群，对应幺元为 1。记元素个数为 n 的有限域GF(n)。可以证明，有限域GF(n)的阶数 n 必为质数或某一质数的幂。我们关心域$GF(2)=\{0,1\}$及其扩展域$GF(2^m)$。这是因为$GF(2^m)$域上的加法运算和乘法运算可以通过逻辑与 AND 或异或逻辑 XOR 来完成，在设计中极易实现。

(a_i,b_i)	add	(a_i,b_i)	multiple
(0,0)	0	(0,0)	0
(0,1)	1	(0,1)	0
(1,0)	1	(1,0)	0
(1,1)	0	(1,1)	1

从上表中可以看出GF(2)域上两个数的加法与逻辑 XOR 一致[3]，两个数的乘法与逻辑 AND 一致。

定义 3：$GF(2^m)$域上的多项式

假设$p(x)=x^m+p_{m-1}x^{m-1}+\cdots+p_1x+p_0$为 GF(2)域上阶数为 m 的即约多项式，其中$p_i\in GF(2)$。如果可以被p(x)除尽的形如$x^k+1$多项式的最小次数为$k=2^{m-1}$,则称p(x)为$GF(2^m)$域的本原多项式。记本源多项式的根为α，即有$\alpha^m=p_{m-1}\alpha^{m-1}+\cdots+p_1\alpha+p_0$。

由本源多项式的定义可知，集合$\{1,\alpha,\alpha^2,\ldots,\alpha^{2^m-1}\}$恰好不重复地构成整个$GF(2^m)$域，因而α称为$GF(2^m)$域的生成元素，因而p(x)也称为$GF(2^m)$域的生成多项式。当$i>m-1$时，$\alpha^i$总可以通过上面的式子变换成$1,\alpha,\alpha^2,\ldots,\alpha^{m-1}$的组合，因而$\{1,\alpha,\alpha^2,\ldots,\alpha^{m-1}\}$构成$G(2^m)$域的一组基，称为标准基（Standard Basis）,若将α记为 x,则$GF(2^m)$域上的任一元素 A 可以唯一地记为如下的多项式形式：

[3] 在 GF（2）域中,1+1 的值为 0，因为其结果要对 2 取模。

$$A(x) = a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \cdots + a_1 x + a_0$$

2.2 MSB-first 算法分析

有限域运算中包括有限域加法、乘法以及求逆等运算，其中求逆运算可以转换为有限域乘法运算，有限域加法则十分简单。因此有限域乘法运算自然就是最为关键的运算。有限域乘法器从结构上主要分为比特串行结构（Bit-serial）和并行结构（Bit-parallel）两种，串行结构电路简单，但包含寄存器，运算速度慢，一般适合低码率应用场合。并行结构都是纯组合逻辑，速度快，适合于高码率应用场合。也可以分为脉动结构和非脉动结构两种。如果从算法上分，大致可以分为基于标准基（Standard）,正规基（Normal Base）和对偶基（Dual Base）的三种结构的乘法器。

在这次的课程项目中，我们采用基于标准基的脉动式结构的乘法器，乘法的运算的顺序为高位优先。

基于多项式基的串行乘法器电路简单，无需基转换，根据乘法运算的两个子过程：多项式乘法和多项式取余运算的排列顺序的不同，我们将乘法器分为两种执行方式，低位优先（LSB-first）乘法、高位优先（MSB-first）乘法。

假设 T 为 A 和 B 的乘积，这里,A, B, P∈GF(2^m)，其中：

$A=\sum_{i=0}^{m-1} a_i x^i = a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \cdots + a_1 x + a_0$
$B=\sum_{i=0}^{m-1} b_i x^i = b_{m-1}x^{m-1} + b_{m-2}x^{m-2} + \cdots + b_1 x + b_0$
$T=\sum_{i=0}^{m-1} t_i x^i = t_{m-1}x^{m-1} + t_{m-2}x^{m-2} + \cdots + t_1 x + t_0$

假设 G(X)为GF(2^m)域上的本原既约多项式。$G(x) = x^m + g_{m-1}x^{m-1} + g_{m-2}x^{m-2} + \cdots + g_1 x + g_0$。在下面的表述中，除非特别声明，否则“+”代表逻辑 XOR，“·”代表逻辑 AND，“α”代表本原多项式的根。

在 MSB-first array type 乘法算法中，多项式乘法和模G(x)两种运算交叉进行，但是乘法运算从乘数 B 的最高位开始

初始状态：$t^{(0)} = 0$;
第一步：$t^{(1)} = A \cdot b_{m-1}$
第二步：$t^{(2)} = t^{(1)} \cdot x \ mod \ G(x) + A \cdot b_{m-2}$
第三步：$t^{(3)} = t^{(2)} \cdot x \bmod G(x) + A \cdot b_{m-3}$
……

第 m 步：$t^{(m)} = t^{(m-1)} \cdot x \bmod \mathrm{G(x)} + \mathrm{A} \cdot b_0$

结果：$\mathrm{p} = \mathrm{p}^{(\mathrm{m})}$

2.3 算法伪代码表示

将上述 MSB-first 算法的步骤整合成伪代码表示为：

Input:$\mathrm{A(x)}, \mathrm{B(x)}, \mathrm{G(x)}$

Output:$\mathrm{P(x)} = \mathrm{A(x)B(x)}\ mod\ G(x)$

$$t_j^{(0)} = 0, 0 < j < m - 1$$

$$t_{-1}^{(i)} = 0, 0 < \mathrm{i} < \mathrm{m} - 1$$

For i = 1 to m do

For j = m – 1 to 0 do

$$t_j^{(i)} = t_{m-1}^{(t-1)} g_j + b_{m-\mathrm{i}} a_j + t_{j-1}^{(i-1)}$$

$$\mathrm{P(x)} = T^m(x)$$

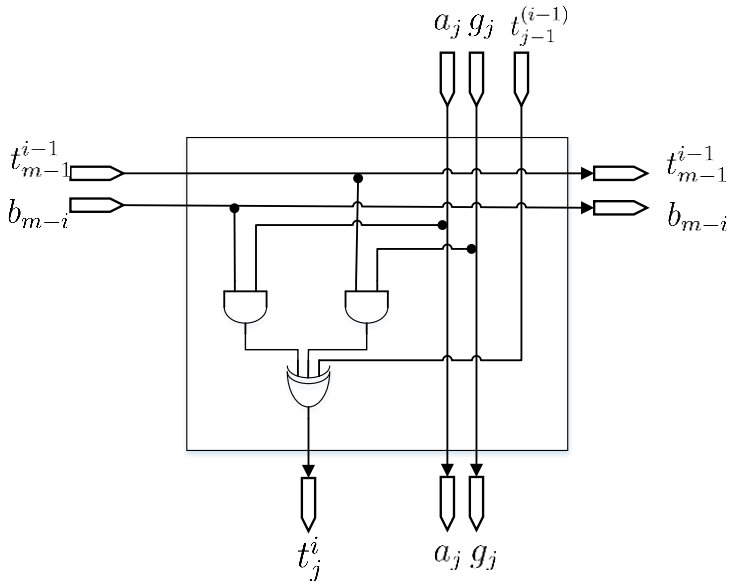
3. 设计思路

3.1 运算基本单元 PE

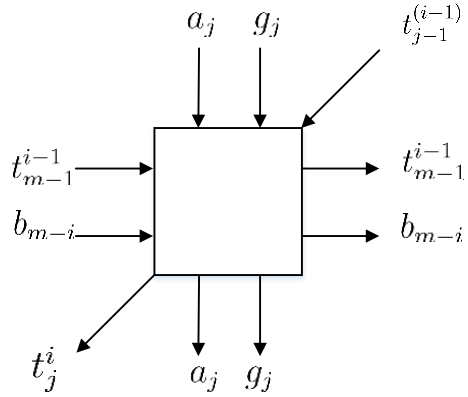
由算法运行的代码可知，循环内最基本的运算为：

$$t_j^{(i)} = t_{m-1}^{(t-1)} g_j + b_{m-\mathrm{i}} a_j + t_{j-1}^{(i-1)}$$

而且由于有限域的系数在 GF(2)中，所以满足$a_i, b_i, g_i \in \{0,1\}$，乘法可以用 AND 逻辑实现，加法可以用 XOR 逻辑实现，实现电路如下所示：

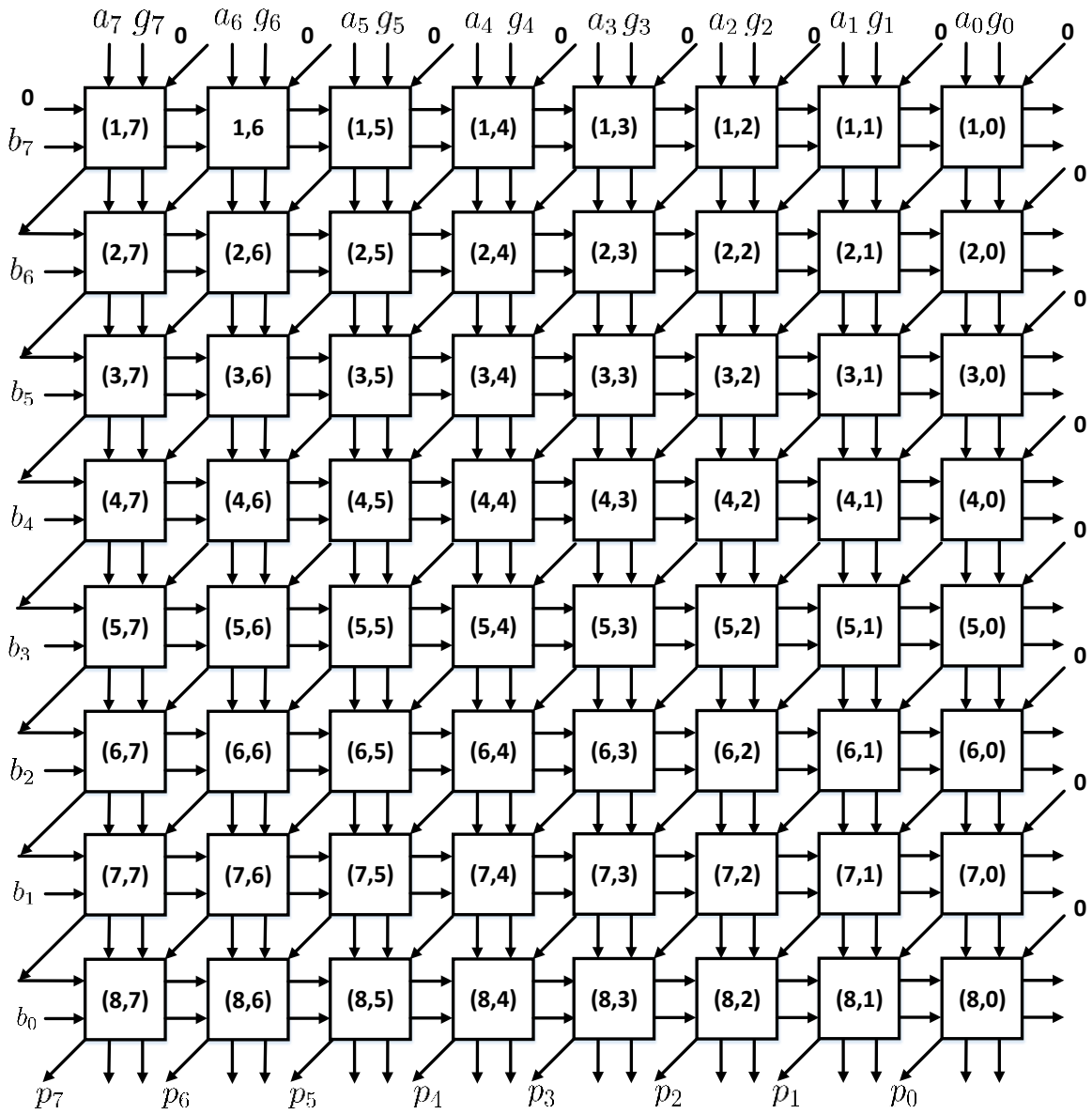


根据信号的传输流的方向，可以将上述电路封装为：

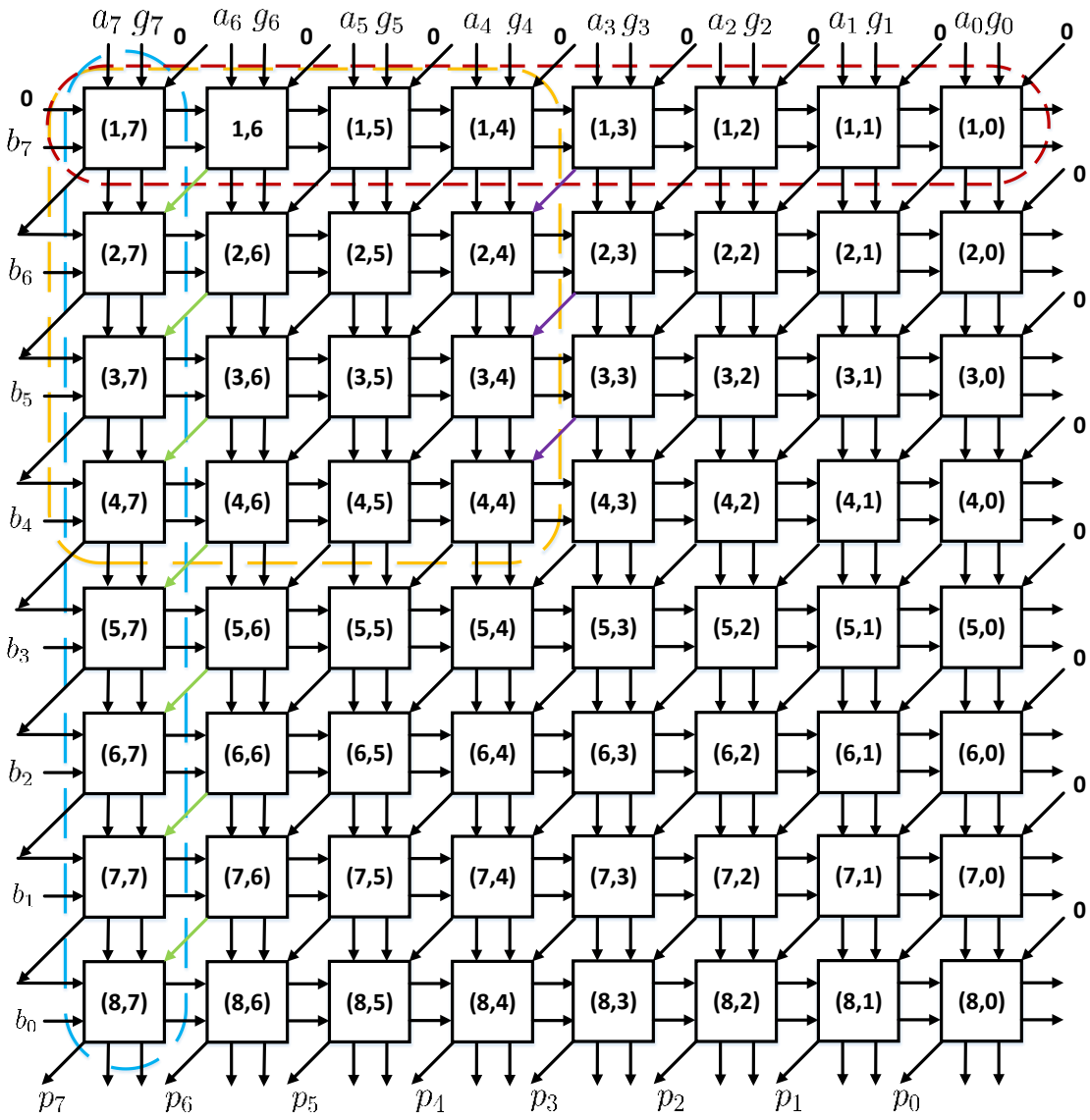


3.2 脉动阵列

考虑当 m=8 时，这个电路循环展开，使用 systolic 脉动阵列的思想，电路展开的方式如下所示：



由下图所示，在考虑 digital-serial 的时候，主要考虑是按照二维的方式进行 digital-serial 的数据输送还是采用一维的方式进行 digital-serial 的数据输送。



（1）一维数据输送：在一维数据输送中，我们主要考虑是对操作数b_i进行 digital-serial 数据输送，还是对操作数a_i进行 digital-serial 方式的数据输送。上图中，红色虚线框代表对操作数b_i进行 digital-serial 数据输送，该方式的数据流中，只有顺序的数据依赖，没有逆向的数据依赖，是一种可行的数据输送方式。而蓝色的虚线框代表对操作数a_i进行 digital-serial 方式的数据输送，但是该方式中存在反向的数据依赖关系，所以不适合 digital-serial 的方式。

（2）二维数据输送：在二维数据输送中，如图中的黄色框所示，组成的一个 4x4 的方阵，改方阵也存在方向的数据依赖关系，方向的数据依赖由图中紫色线所表示。所以也不适合 digital-serial 的方式，除非修改算法的循环表达式或者算法的循环结构才能进一步消除电路的数据依赖关系。

所以经过分析论证，我们小组选择红色虚线框中的 digital-serial 的数据输送方式。

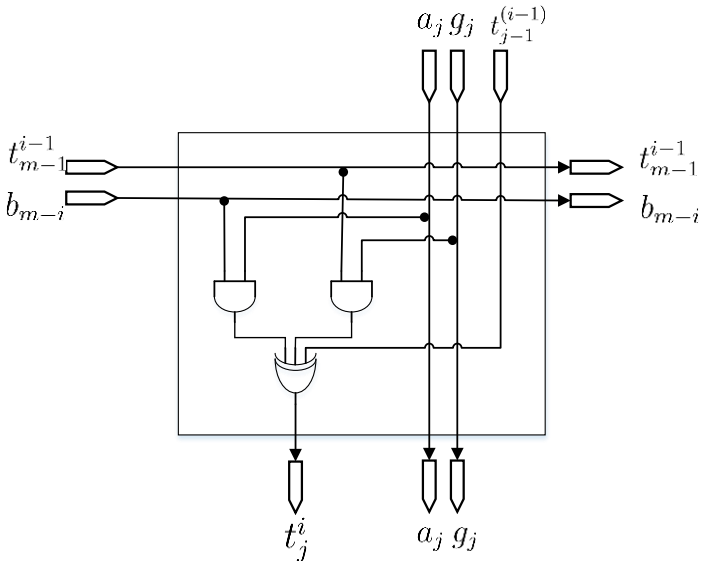
4. 硬件实现

在本设计中，我们用 perl 脚本生成了不同 bit 和不同阶数 m 的 RTL 代码。

以 digit 分为 8bit 为例。

4.1 basic_element 模块

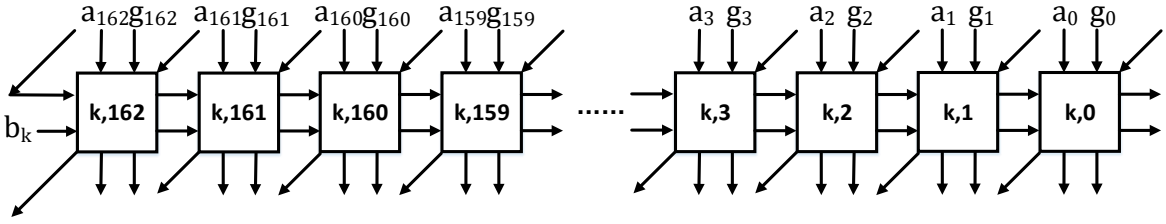
该模块即实现运算基本单元 PE，如下图所示：



不加赘述。

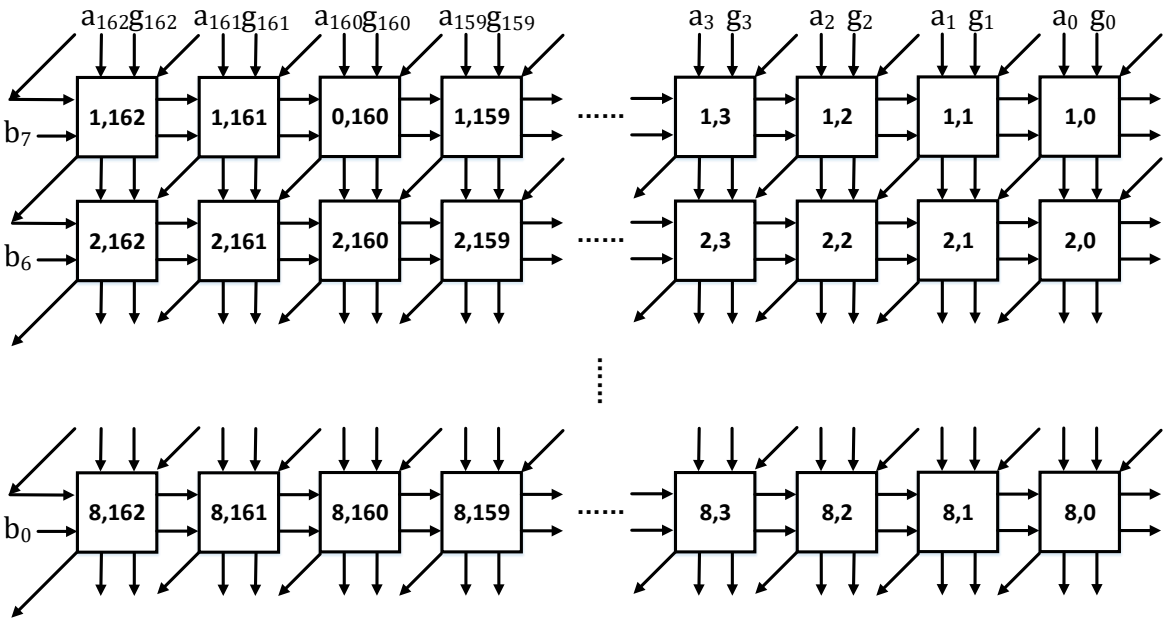
4.2 one_bit 模块

根据前述分析，我们采用了对操作数b_i进行 digital-serial 数据输送的方式。one_bit 模块是每一 bit 的实现。如下图所示。

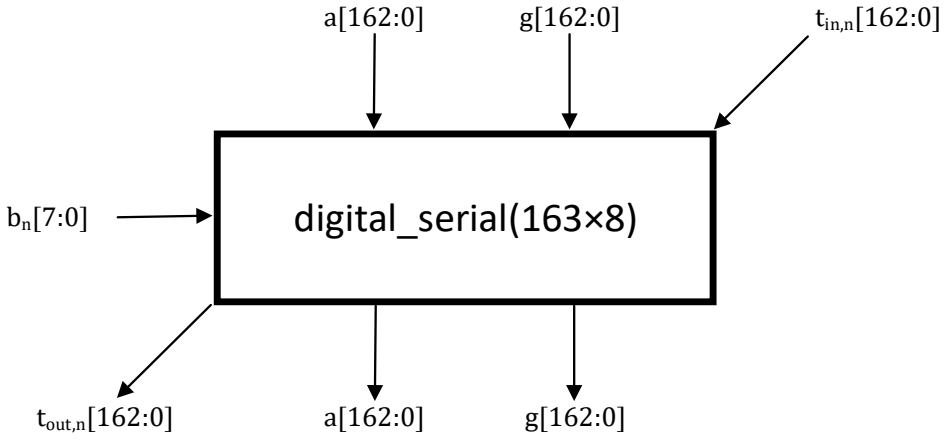


4.3 digital_serial 模块

digital_serial 模块包含若干比特个（bit = 8 or 16 or 32）one_bit 模块，是 163×bit 的 PE 的方阵。如下图所示。

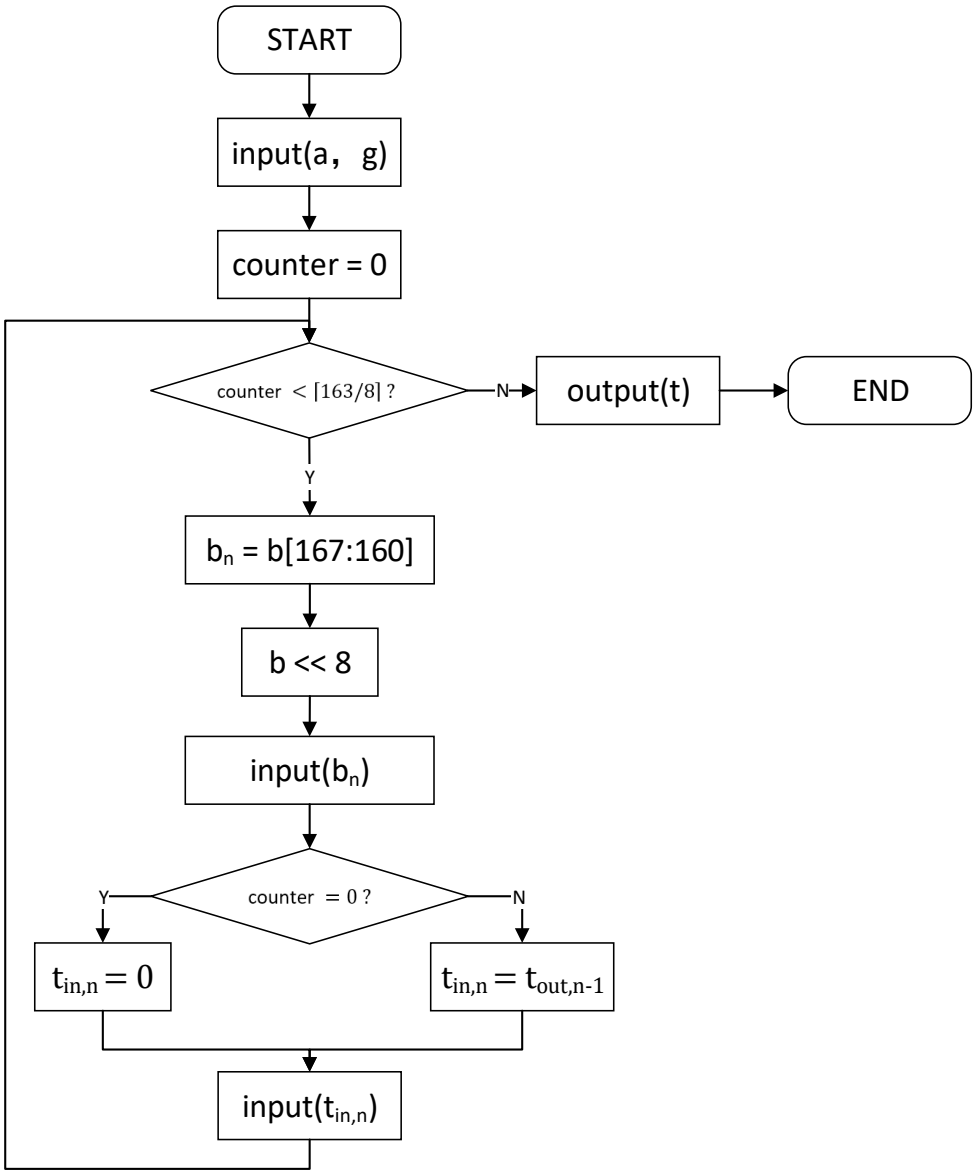


可以将上述电路封装为。



4.4 8socWrapper 模块

该模块控制 digital_serial 在不同时钟下的输入，并在计算完成时输出结果。其基本流程如下图所示。



在我们的设计中，digital_serial 模块的 a 和 g 是固定的 163bit 输入。

设立一个计数器，自 0 计数到⌈163/8⌉ − 1，代表本设计需要⌈163/8⌉个时钟周期完成，在第⌈163/8⌉ + 1个时钟周期输出结果。每个时钟周期，将 b 的高 8 位取出，并将其左移 8 位，把取出的高 8 位送入 digital_serial 模块。

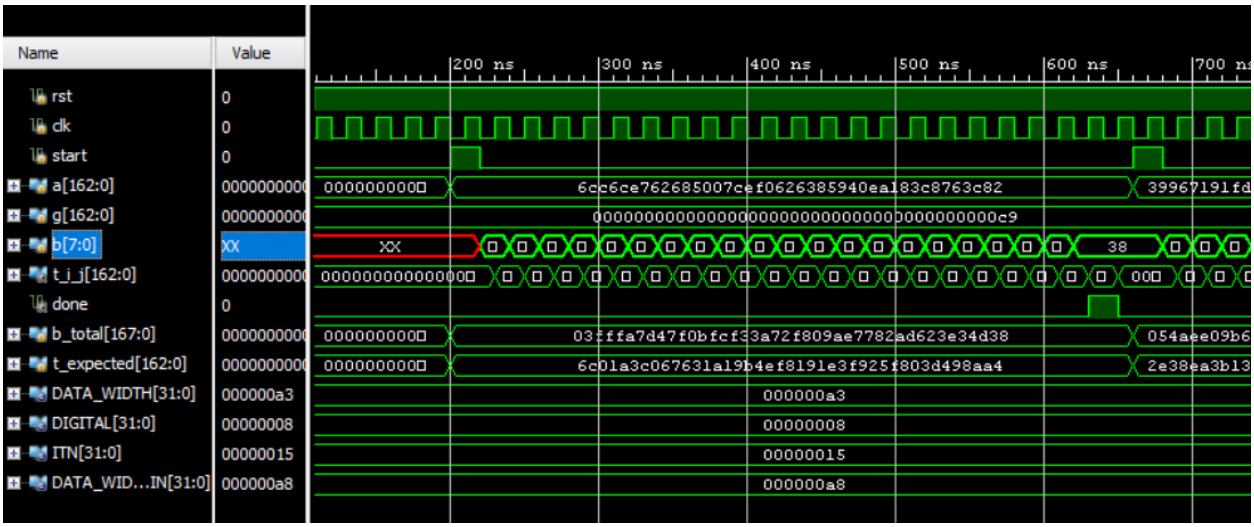
在计数的第 1 个时钟周期，$t_{in,n}$ 为 0，其他时钟周期 $t_{in,n}$ 取的是上一个时钟周期的输出 $t_{out,n-1}$。

5. 行为级仿真

在仿真中，我们用 perl 脚本生成随机的 a,b 的值，并将硬件仿真的结果与 perl 脚本生成的结果的对比，验证设计电路在不同输入下的输出正误。

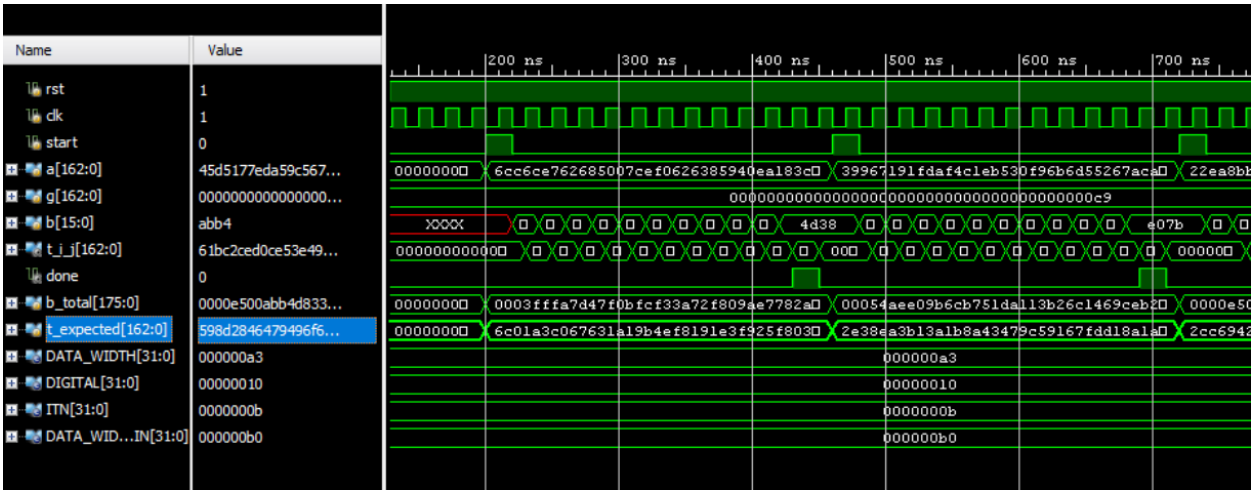
结果如下。

5.1 8bit 仿真结果



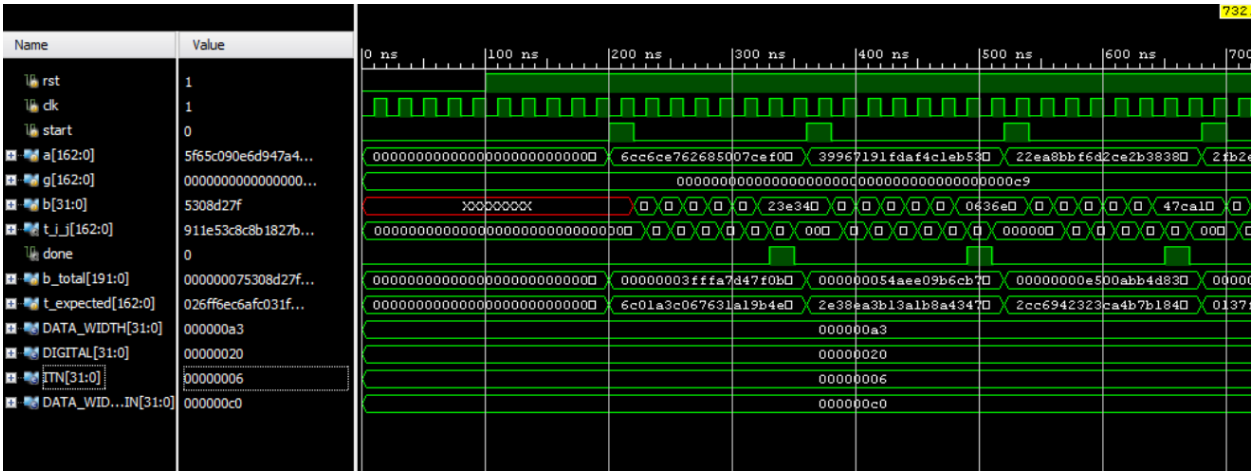

```
launch_simulation: Time (s): cpu = 00:00:01 ; elapsed = 00:00:16 . Memory (MB): peak = 1849.727 ; gain = 0.000
run 100 us
this data is right
this data is right
this data is right
this data is right
this data is right
this data is right
this data is right
this data is right
this data is right
$finish called at time : 4800 ns : File "D:/project/zyboLearning/8bit_generate/8bit_generate.srcs/sim_1/new/simu.v" Line 629
```

5.2 16bit 仿真结果



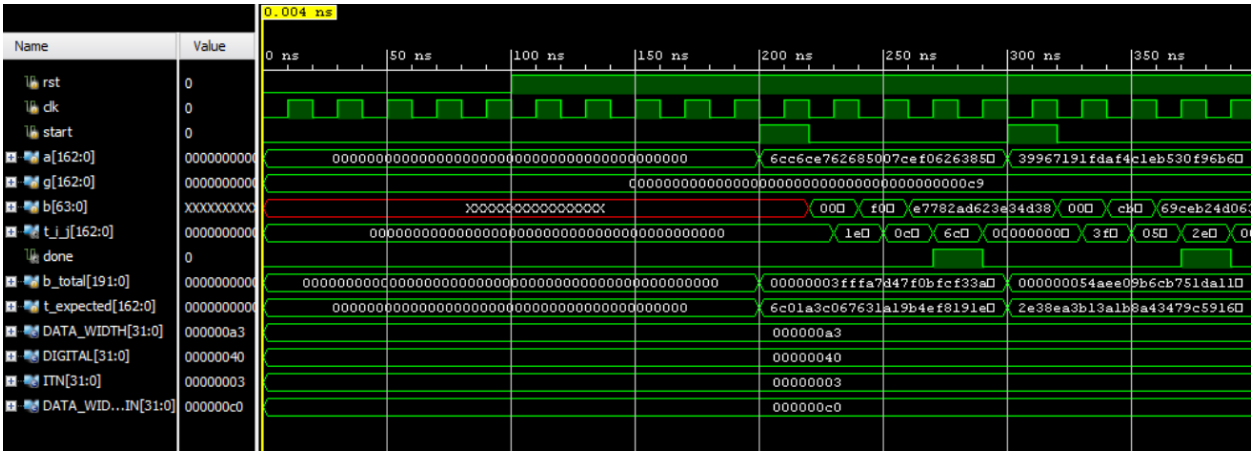

```
this data is right
this data is right
this data is right
this data is right
this data is right
this data is right
this data is right
this data is right
this data is right
this data is right
$finish called at time : 26200 ns : File "D:/project/zyboLearning/16bit_generate/16bit_generate.srcs/sim_1/new/simu16bit.v" Line 3579
```

5.3 32bit 仿真结果




```
this data is right
this data is right
this data is right
this data is right
this data is right
this data is right
this data is right
this data is right
this data is right
this data is right
$finish called at time : 16200 ns : File "D:/project/zyboLearning/32bit_generate/32bit_generate.srcs/sim_1/new/simu32bit.v" Line 2579
run: Time (s): cpu = 00:00:02 ; elapsed = 00:00:08 . Memory (MB): peak = 1849.727 ; gain = 0.000
```

5.4 64bit 仿真结果




```
this data is right
this data is right
this data is right
this data is right
this data is right
this data is right
this data is right
$finish called at time : 10200 ns : File "D:/project/zyboLearning/64bit_generate/64bit_generate.srcs/sim_1/new/simu64bit.v" Line 1979
run: Time (s): cpu = 00:00:03 ; elapsed = 00:00:17 . Memory (MB): peak = 1849.727 ; gain = 0.000
```

6. FPGA 实现及 SoC 验证

6.1 Vivado 综合，功耗，频率

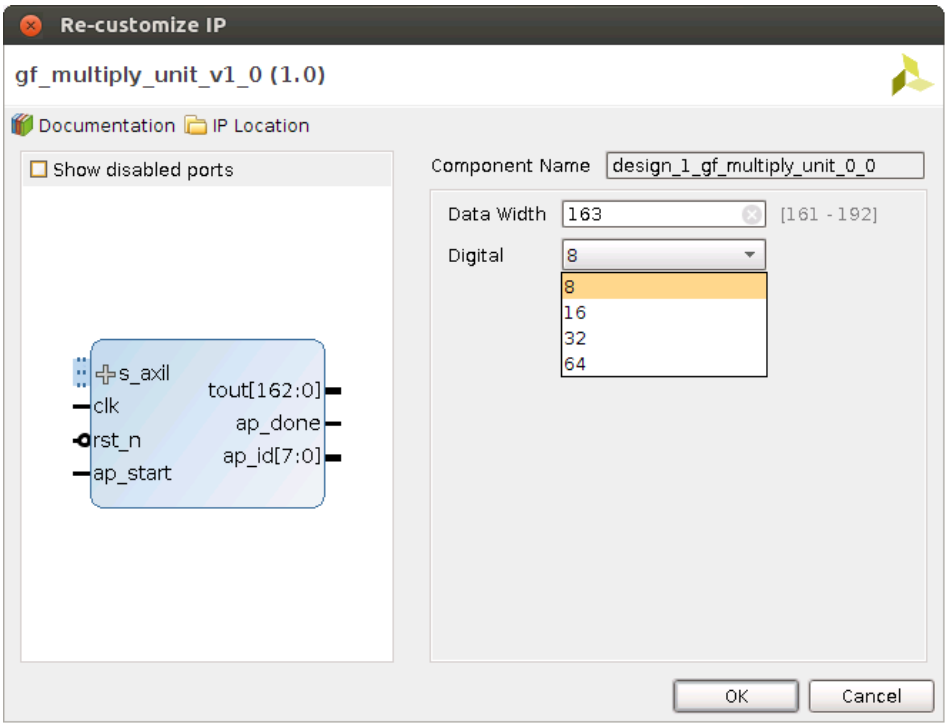
Vivado 下对乘法器模块进行综合，资源占用、功耗、频率如下表所示。

	LUTs	Registers	Power (W)	Freq (MHz)
8 bit	1331	178	0.376	260
16 bit	2631	178	0.389	142
32 bit	5240	178	0.458	75

6.2 IP 封装

将参数化设计实现的乘法器模块在 Vivado 中封装成 IP 核，其 GUI 例化界面如下图所示。其中 Data Width 为数据宽度，Digital 为不同 Digital 位数的选择，可选范围为 8、16、32、64 比特。

封装好的 IP 核的接口有时钟 clk 及其同步复位 rst_n 输入，控制接口开始信号 ap_start、结束信号 ap_done，数据接口 s_axil。s_axil 符合 AXI4 Lite 总线接口标准，用于传输初始数据 a、b、g 以及运算结果 t。



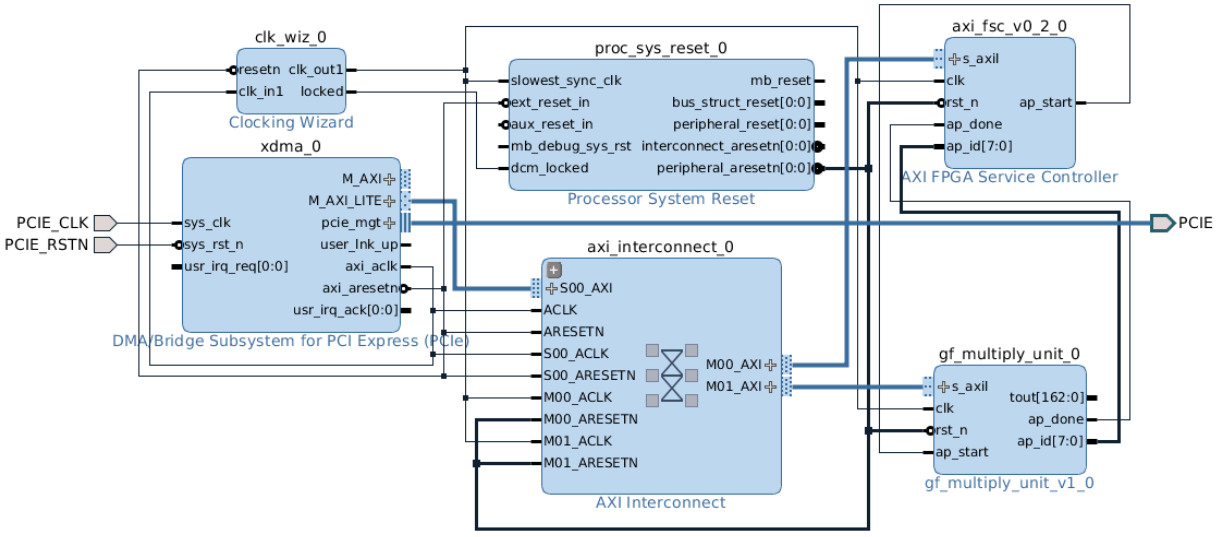
6.3 SoC 平台搭建

本设计中用于功能验证的 SoC 平台基于 Xilinx VC709 开发板搭建，开发板与主机使用 PCIe 总线通信，主机 CPU 通过 PCIe 对片上电路进行控制。

SoC 电路如下图所示，其中 xdma_0 模块为 PCIe 控制器，axi_interconnect_0 为片上 AXI4 Lite 总线，axi_fsc_v0_2_0 为计算模块控制器，clk_wiz_0 为 PLL 分频器，proc_sys_reset_0 为复位信号同步器。上述模块均为 Xilinx 提供或已有的设计。其中 gf_multiply_unit_0 为乘法器模块。

PCIe 接口为 xdma_0 提供了 100M 参考时钟 PCIE_CLK 和同步复位 PCIE_RSTN 输入。xdma_0 对该 100M 时钟进行了倍频得到 250M 时钟 axi_aclk 及其同步复位 axi_aresetn。axi_aclk 为片上总线提供了时钟。clk_wiz_0 对 axi_aclk 时钟进行分频得到乘法器模块及其控制器的时钟 clk。proc_sys_reset_0 对 axi_aresetn 复位进行同步得到乘法器模块及其控制器的同步复位 rst_n。

xdma_0 模块将 PCIe 总线转换为 AXI4 Lite 接口，CPU 通过 AXI4 Lite 总线对乘法器模块及其控制器上的寄存器进行读写，从而实现数据传输以及计算过程的控制。



6.4 SoC 实现

对 8、16、32bit 的情况分别进行了实现，资源占用、功耗、频率如下表所示。

	LUTs	Registers	Power (W)	Freq (MHz)
8 bit	21585	21012	4.359	250
16 bit	22880	21027	4.353	140
32 bit	25515	21061	4.359	70

6.5 SoC 验证

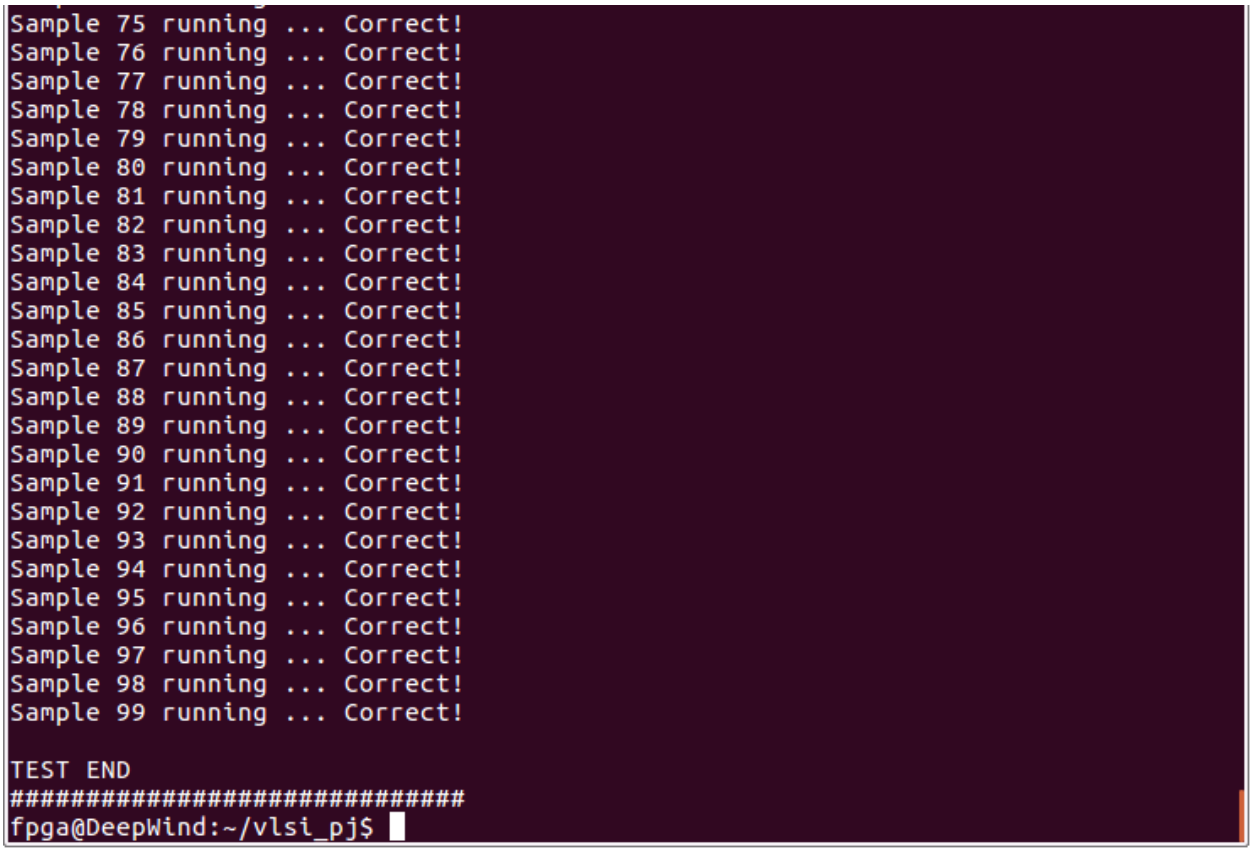
我们分别对 8、16、32 三种情况的乘法器，在主机运行脚本随机生成 100 组数据进行正确性测试，脚本关键部分代码如下所示。

```
# 随机生成 a、b
for((i=0; i<$data_width; i++))
do
        a[$i]=$(( $RANDOM % 2 ))
        b[$i]=$(( $RANDOM % 2 ))
        g[$i]=0
done
g[0]=1; g[3]=1; g[6]=1; g[7]=1

# 计算 t 正确值
```

```
for((i=0; i<$data_width_p1; i++))
do
	for((j=0; j<$data_width_p1; j++))
	do
		t_i_j=$(( $data_width_p1 * $i + $j ))
		t[$t_i_j]=0
	done
done

for((i=1; i<$data_width_p1; i++))
do
	for((j=$data_width; j>0; j--))
	do
		t_i_j=$(( $data_width_p1 * $i + $j ))
		t_i1_m=$(( $data_width_p1 * $(( $i - 1 )) + $data_width ))
		g_j1=$(( $j - 1 ))
		b_mi=$(( $data_width - $i ))
		a_j1=$(( $j - 1 ))
		t_i1_j1=$(( $data_width_p1 * $(( $i - 1 )) + $j - 1 ))

		t[$t_i_j]=$(( $(( ${t[$t_i1_m]} & ${g[$g_j1]} )) ^ $(( ${b[$b_mi]} &
${a[$a_j1]} )) ^ $(( ${t[$t_i1_j1]} )) ))
	done
done

# 向 FPGA 发送 a、b、g
for((i=0; i<=5; i++))
do
	value_a=""
	value_b=""
	value_g=""
	for((j=0; j<32; j++))
	do
		value_a=${a[$(($i * 32 + $j))]}$value_a
		value_b=${b[$(($i * 32 + $j))]}$value_b
		value_g=${g[$(($i * 32 + $j))]}$value_g
	done
	./reg_rw_8k /dev/xdma0_user 0x$(echo "obase=16; $(($i * 4 + 4096))"|bc) w
0x$(echo "obase=16; ibase=2; $value_a"|bc) >> run.log
	./reg_rw_8k /dev/xdma0_user 0x$(echo "obase=16; $(($i * 4 + 4128))"|bc) w
0x$(echo "obase=16; ibase=2; $value_b"|bc) >> run.log
	./reg_rw_8k /dev/xdma0_user 0x$(echo "obase=16; $(($i * 4 + 4160))"|bc) w
0x$(echo "obase=16; ibase=2; $value_g"|bc) >> run.log
Done
```

```
# 向 FPGA 发送开始命令
./reg_rw_8k /dev/xdma0_user 0x0000 w 0x100 >> run.log
status=$(./reg_rw_8k /dev/xdma0_user 0x0000 w | awk '/Read/{print $8}')
while [ $status = "0x00000200" ]
do
       status=$(./reg_rw_8k /dev/xdma0_user 0x0000 w | awk '/Read/{print $8}')
done

# 读取 FPGA 计算结果
for((i=0; i<=5; i++))
do
       result[$i]=$(./reg_rw_8k /dev/xdma0_user 0x$(echo "obase=16; $(($i * 4 +
4256))"|bc) w | awk '/Read/{print $8}')
done
```

运行结果如下图所示。



三种情况下 100 组数据均测试通过，说明乘法器模块 SoC 验证通过。

7. DC 综合，功耗，面积

在 DC 综合中，我们测试了 TSMC 65nm 下 的最高频率，及其对应的功耗，面积。如下表所示。

	Area/um^2	Power/mw	Frequency/MHz
4bit	10827.360067	4.2252	675
8bit	18671.400208	5.7480	467
16bit	34058.520227	7.5541	289
32bit	62305.200116	7.4711	161

根据表中数据，可以计算不同 bit 下的吞吐率和能效比。电路的吞吐率指的是单位时间电路处理的 bit 数，能效比为电路处理单位 bit 数所需要的能量。

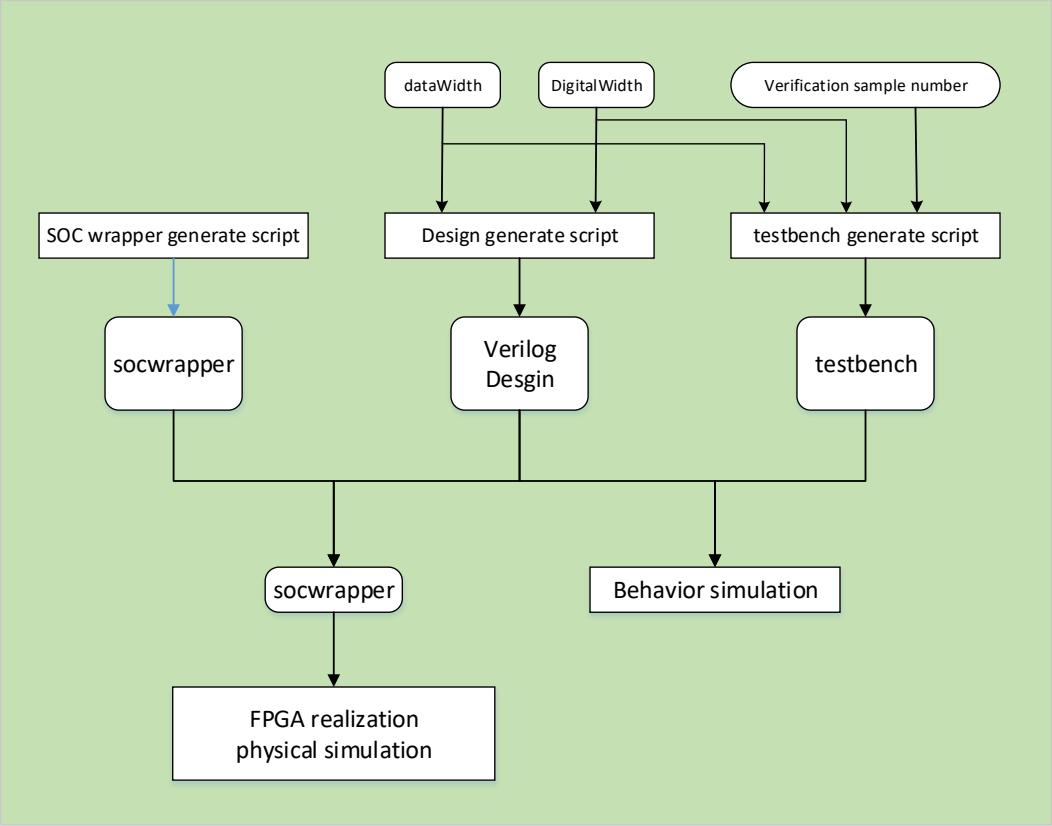
	Frequency/MHz	吞吐率/Gbits/s	能效比/pJ/bit
4bit	675	2.51	1.68
8bit	467	3.37	1.71
16bit	289	3.98	1.89
32bit	161	4.07	1.84

8. 总结

在这次设计中，我们主要设计实现了 $GF(2^{163})$ 域上的基于脉动（Systolic）阵列的 Digit-Serial 乘法器，设计出了 8,16,32 比特的三种情况。

在设计脉动阵列时，我们主要考虑到了数据的相关性，选择了一种可以消除数据相关性的实现方式，改设计主要使用了的 PE 单元组成了很规整的脉冲阵列。经过我们的行为级仿真验证和 FPGA 仿真验证，证明了我们的设计的正确性，同时通过 DC 综合证明了我们的设计具有很高的吞吐率和能效比，功耗在 mW 级别，达到了低功耗的设计要求，

不仅如此，我们此次设计的亮点在于设计的整个流程通过脚本实现了自动化。在自动化设计的流程当中，在设定数据位宽和 digital-Serial 的长度之后，脚本可自动生成相应的代码和测试激励。另外为了在 SOC 系统上进行验证，我们设计出生成 socWrapper 的代码脚本，socWrapper 的功能主要是将我们的设计封装为一个可以挂在 AXI 总线上的 IP，之后可以进行进一步的板级验证与调试。整个自动化设计的流程可以用流程框图表示如下：



在这次的课程项目中，我们小组分工合作，很快确定了算法和实现方案，尤其是选择了全自动化的实现方案，最后我们也到达了预期了效果，收获颇丰。