上海交通大學

SHANGHAI JIAO TONG UNIVERSITY


# VLSI 数字通信原理与设计课程大作业

# 基于随机计算的乘累加计算电路

林羽 519021910467

王清训 519021910593

李卓壕 519021911248

# 目录

# 1 实验背景与要求

深度神经网络（Deep Neural Network，DNN）广泛地被应用在计算机视觉、自然语言处理、语音识别等各个领域，且取得了显著的成就。然而随着网络深度的增加，其计算量急剧增加，越来越多的研究者开始关注 DNN 的硬件加速器设计。DNN 的乘累加运算占总运算量的 90% 以上，由此导致的巨大硬件资源开销和功耗使得其在嵌入式设备等硬件资源受限的平台上的部署变得困难。另一方面，由于摩尔定律走到尽头，难以单纯依靠更先进的工艺来缩小芯片面积。因此，迫切需要新颖的替代计算范例克服这个障碍。随机计算 (Stochastic Computing,SC) 作为一种新型计算范式，为解决上述问题提供了可能。利用随机计算，可以用极低的硬件开销为代价实现一些常见的计算电路，例如乘法器和放缩加法器等。本实验将设计、实现、优化基于随机计算的乘累加电路。

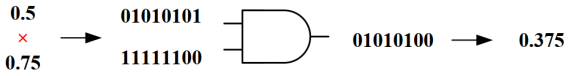



图 1: SC 序列乘法实现

SC 将 [0,1] 范围内的二进制数转化为一串随机的 0,1 序列表示，序列中 1 所占的比例即为数的大小。例如，序列 1,0,1,0，1,1,0,0，1,0,1,0,1,0,1,0 均可以表示数 0.5。应用 SC 后，原来很复杂的运算可以由简单的电路来实现。上图说明了 SC 乘法的具体过程：通过将二进制数输入（0.5，0.75）转化为 SC 序列后，逐比特通过与门得到输出序列，根据与门性质，输出的序列表示的数（0.375）即为两数相乘的结果。因此乘法在 SC 中仅需要一个与门即可实现，从而大大降低面积和功耗。

本实验要求实现基于随机计算的乘累加电路，并尝试多种不同的随机计算单元。阅读文献，设计实现基于 Sobol 序列的随机计算乘累加电路，具体要求如下：

- MATLAB 实现 LFSR+MUX、Sobol+MUX 和 Sobol+ADD 共 3 种 8 输入 sc 乘累加计算单元，比较他们的精度。
- 设计并实现基于 Sobol 序列的 8 输入 sc 乘累加计算单元 A1，并提高其比特级并行度，设计实现并行架构 A2。

# 2 matlab 中的实现与仿真验证

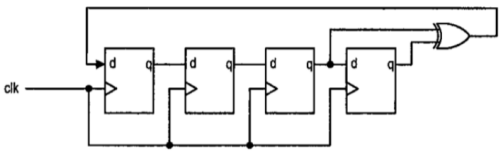
用 MATLAB 实现实验所需的 8 输入 SC 乘累加计算单元，随机数发生器分别采用 LFSR 和 Sobol，加法器分别利用 MUX 和 ADD 实现。将 SC 序列转回二进制之后统计精度，评估精度的指标采用平均绝对误差 (Mean AbsoluteError，MAE)，定义为下式，要求 n 至少为 1000。

$$MAE = \frac{1}{n}(\sum_{i=1}^{n} |SCResult - PreciseResult|) \tag{1}$$

## 2.1 LFSR+MUX

### 2.1.1 软件设计思路

首先介绍 LFSR 的设计思路。LFSR 的参考结构如下图。LFSR 就是线性反馈移位寄存器，是移位寄存器的一种，通常用于在数字电路中产生伪随机数，寄存器中的初始值叫做种子，种子应该是非零的，因此在 matlab 中通过随机函数生成随机种子并保证若随机数恰好是零则转换为非零数。移位寄存器链的多个抽头用作 XOR 门的输入，输出反馈到首位。每个周期最后一位寄存器的还会输出为 LFSR 序列的一位。为保证两个 LFSR 生成的随机数拥有较低相关性，往往采用不同的 LFSR 的结构。LFSR 结构的变化可以采取两种方式，**本原多项式的改变和寄存器数量的改变**，在软件设计过程中我对两种方式都进行了尝试但是效果基本一样，最终统一使用了改变寄存器数量的方式。共采用了 5 种结构对应 LFSR+MUX 过程中的五次 SC 序列生成，寄存器数量分别为 **5**、**6**、**7**、**8**、**16**，对应的本原多项式分别为 $x^5+x^3+1$、$x^6+x^5+1$、$x^7+x^4+1$、$x^8+x^6+x^5+x^4+1$、$x^16+x^13+x^12+x^11+1$。




4bit LFSR 电路图

图 2: LFSR 参考结构

因为通常 n-bit 的定点数需要 $2^n$ 长度的 SC 序列来确保准确率，考虑到 5bit 的位宽，因此将 SC 序列长度定为 32，此时需要的 LFSR 的长度也是 n 比特，因此每次 LFSR 生成的序列长度都为 5，即一个 5bit 的小数，然后将此数与输入比较 (若比输入大则输出 0 反之输出 1) 得到 1bit 的 SC 序列，经过 32 次重复过程，得到 32bit 长度的 SC 序列。

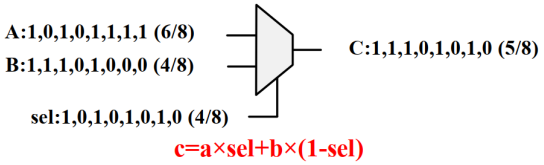



图 3: MUX 实现过程

接下来介绍 MUX 的软件设计思路。MUX 是实现 SC 序列放缩加法的一种方式。实现过程如上图。其中，$c=a\wedge sel+b\wedge(1-sel)$, 由于 SC 序列的相与就是乘法的意思，当 sel 是 0.5 转化成的 SC 序列时，通过 MUX 就实现了 $c=\frac{a+b}{2}$ 的运算，当然最终结果是除以 2 的放缩，最终结果还需要将 2 乘回来。

最终整体的设计为，四对输入中的每一对分别用 5、6 寄存器数量的 LFSR 生成 SC 序列，得到四个乘法输出，两两用 MUX 相加，sel 分别用 7、8 寄存器数量的 LFSR 生成，得到两个加法输出后再将这两个输出用 MUX 相加，sel 用 16 寄存器数量的 LFSR 生成。根据输出中 $\frac{the\ number\ of\ "1"}{32}$ 得到十进制输出然后乘 4(两次放缩因此乘 4) 得到最终结果。精确结果则是通过四对 5bit 输入各自进行二级制乘法运算后 (乘法结果仍为 5bit，此处模拟了硬件设计中的乘法，乘法前后位宽不

变) 将四个乘法结果相加得到，最后计算 MAE, 重复 2000 次。

### 2.1.2 仿真结果

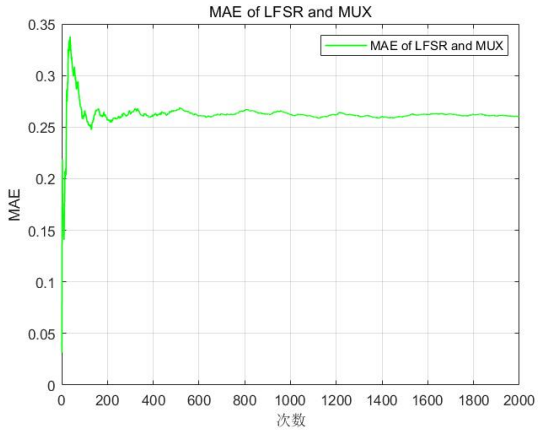



图 4: MAE of LFSR and MUX

## 2.2 Sobol+MUX

### 2.2.1 软件设计思路

MUX 设计与前文一致不做赘述。接下来主要介绍 Sobol 的软件设计。首先我们选择一个数量级 d，然后确定一个 d 级的本原多项式。

$$P(x) = x^d + a_1 x^{d-1} + \ldots\ldots + a_{d-1} + 1 \tag{2}$$

根据本原多项式得到 c 序列。

$$c_i = a_1 c_{i-1} \oplus a_2 c_{i-2} \oplus \ldots\ldots a_{d-1} c_{i-d+1} \oplus c_{i-d} \oplus (c_{i-d+1} >> d) \tag{3}$$

在实验中，由于输入序列为 5bit 长度，因此 d 取了 5，本原多项式定为 $P(x) = x^5 + x^3 + 1$，只有 $a_2 = 1$，其他 a 为 0。$c_{1-5}$ 根据下式确定。

$$c_i = \frac{m_i}{2^i} = 0.c_{i1}c_{i2}c_{i3}c_{i4}c_{i5}(m_i < 2^i and m_i is odd) \tag{4}$$

m 作为自己选择的奇数有多种选择，不同选择最终得到的 SC 序列也不同，因此可以通过随机选择不同的 $m_{1-5}$ 组合来随机得到不同的 Sobol 序列从而进一步降低序列相关性。

输出序列由下式迭代确定。

$$x_{n+1} = x_n \oplus c_r \tag{5}$$

其中 r 是正整数 n 转化为二进制后的最低位 0 的位数，比如 001 的最低位零为第 2bit，因此 r 是 2，110 则为 1。从 $x_0 = 0$、$n = 0$ 开始迭代，每一个 $x_i$ 都是一个 5bit 小数，最终得到 32 长度 Sobol 序列，即 32 个 5bit 小数，Sobol 序列的特点是序列元素都是以 $2^n$ 为分母以奇数为分子的小数以及 0，当 Sobol 序列长度为 32 时，序列一定是了 $\frac{0-31}{32}$ 的一个排列。

用这套规则在 matlab 中迭代计算得到 $x_{0-31}$ 就是所需的 32 长度的 Sobol 序列。然后将 32 个小数 (Sobol 序列中的每一个元素) 依次与输入相比得到 32 长度的 SC 序列。

每次的 SC 序列生成都是随机的初始化 $m_{1-5}$ 保证了序列相关性最低，与 LFSR+MUX 相比仅将输入序列和 MUX 的 sel 都改为通过 Sobol 生成就得到了 Sobol+MUX，其他处理相同。

#### 2.2.2 仿真结果

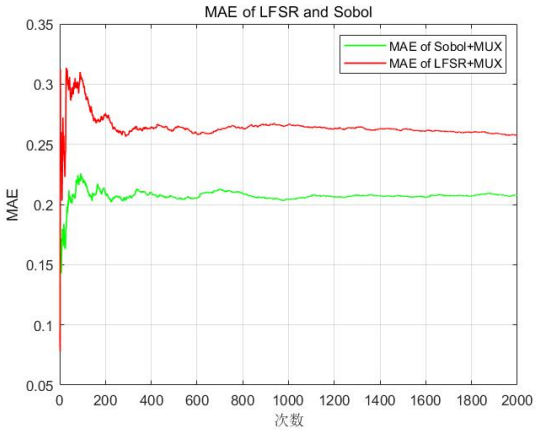



图 5: MAE of LFSR and Sobol with MUX

### 2.3 Sobol+ADD

#### 2.3.1 软件设计思路

序列生成与 Sobol+MUX 相同，改变的仅仅是最后的加法环节。ADD 的相加过程是 4 对乘法的输出直接根据 (1 的数量/32) 转化为二进制数然后进行二进制相加，与 MUX 相比，省了 sel 的 SC 序列生成也因此避免了由此而造成的精度降低。

#### 2.3.2 仿真结果

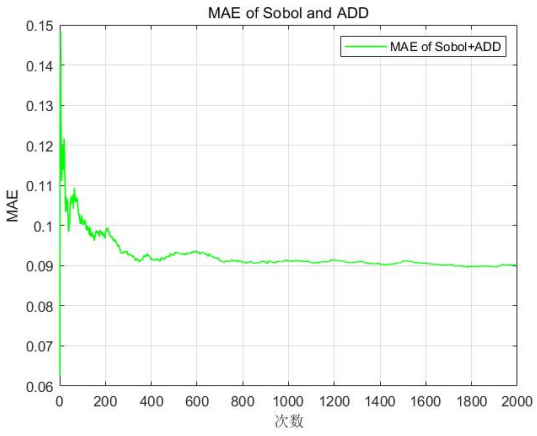



图 6: MAE of Sobol and ADD

### 2.4 比较与分析

最终的 MAE 分别为:**LFSR+MUX 大致为 0.26;Sobol+MUX 大致为 0.21;Sobol+ADD 大致为 0.09，都满足了实验要求的精度**。

从图5可以看到将序列生成方式从 LFSR 改变为 Sobol 后精度得到了提升，MAE 降低了大约 0.05，说明了相对于 LFSR，Sobol 作为一种相关性更低的低密度序列具有更好的数学特性，可以得到更好的精度。

然后使用了 ADD 的加法方式后，精度更是大幅提高，MAE 降低到了 0.1 以下。二进制加法是一种不复杂的硬件运算，没有很大的需要去用 SC 序列实现，甚至相比于 SC 序列加法二进制加法更加地简洁方便。因此综合使用 SC 序列乘法和二进制加法既可以保证运算的简单降低硬件消耗，又可以保证较高的精度。

## 3 Verilog A1 实现

A1 乘累加计算单元采用上述 2.3 中的结构，其中 SC 序列采用串行的方式生成。本部分将逐一说明 A1 各模块的设计思路，以及验证方式和结果。

### 3.1 硬件设计思路

A1 乘累加计算单元主要包含 SC 序列生成和乘累加计算两个部分。其中 SC 序列生成模块包含 Sobol 序列生成器，而乘累加计算的部分分为乘法模块和加法模块。除此之外，电路由顶层模块对上述几个部分进行调用和封装，最终实现功能。各模块的实现细节将在后文进行说明。

### 3.2 电路图

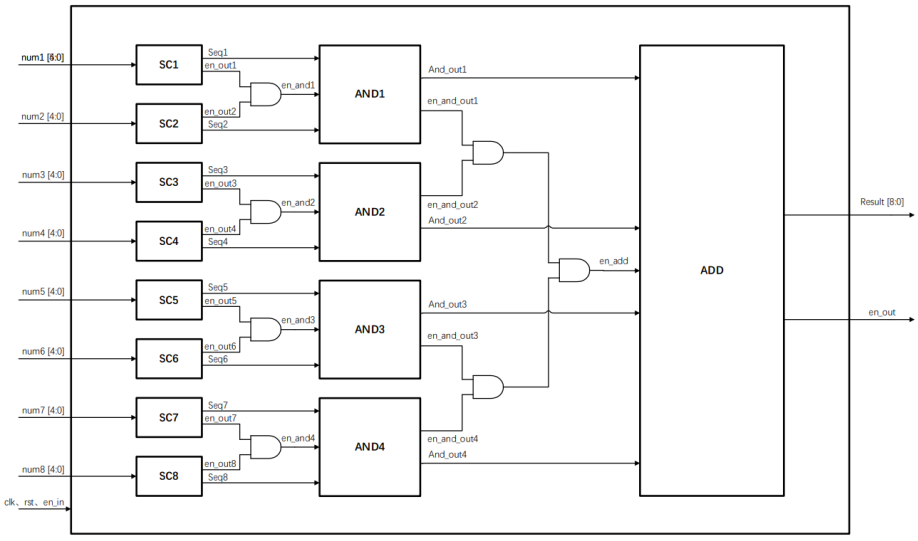



图 7: A1 架构图

A1 乘累加计算单元电路的总体架构如上图所示。电路的输入包含 8 个 5bit 输入数据、使能信号、时钟以及 reset 信号，电路的输出由计算结果和输出使能组成。对于八个输入数据，分别由

八个相关性较低的 SC 序列生成模块进行 SC 序列的生成。生成后的 SC 序通过经过四个二输入乘法模块两两相乘。最后，由四输入加法模块将乘积由 SC 序列转换为二进制的形式并相加，完成计算过程。

## 3.3 RTL 各模块介绍

### 3.3.1 串行 Sobol 序列发生器

Sobol 序列的生成原理已经在 2.3 中进行说明，这里不做赘述。与 MATLAB 软件仿真不同的是，在 Verilog 中，小数不能以浮点数的形式表示，进行电路运算时需要将输入数值放大 32 倍，以 5bit 带宽的形式进行输入。此时，输入范围为 0-31 的整数，精度为 $\frac{1}{32}$。另外，考虑到最终产生的 SC 序列只有 32 位，Sobol 序列的本原多项式次数仅为 6。为简化电路计算，不同 Sobol 序列的 $m_1$ 至 $m_6$ 在电路设计时直接给出。经测试，此时不同序列见的相关性足以满足电路功能需求（具体在后文进行说明）。

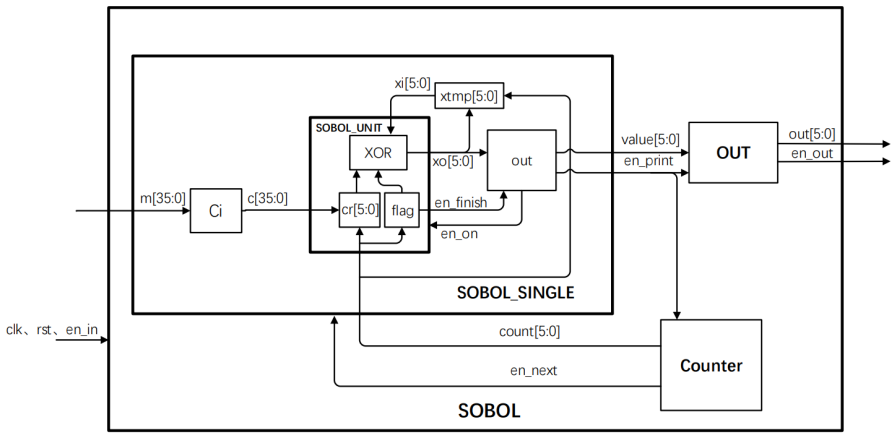



图 8: Sobol 序列发生器架构图

如上图所示，Sobol 序列产生模块由三层函数进行封装。首先由 SOBOL UNIT 函数进行 Sobol 算法的基本运算，接着由 SOBOL SINGLE 函数控制 SOBOL UNIT 函数循环运算的输入和输出，最后由 SOBOL 函数控制 Sobol 序列的长度，将其整理为 32x6bit 的一组进行输出。下面将对三个函数的实现细节进行说明。

SOBOL UNIT 函数主要完成式 (5) 中表述的异或计算。借助 Sobol 序列下标 count，仅需简单的条件逻辑就可以找出此时最低位 0 所在的位置，进而确定 $c_r$ 的取值。flag 信号标记 $c_r$ 的寻找进度，当 $c_r$ 确定时，xo 输出通过组合逻辑进行计算，flag 信号也同时拉高表示此时输出有效。此部分代码详见文件 *sobol_unit.v*。

SOBOL SINGLE 函数将 SOBOL UNIT 的输出结果保存在 xtmp 寄存器中，并作为 SOBOL UNIT 下一周期的输入信号。当 SOBOL UNIT 输出有效时，SOBOL SINGLE 将输出结果向上一级传递。在输出期间关闭对 SOBOL UNIT 的使能，以保证信号的稳定。此外，该模块还完成 $m_i$ 到 $c_i$ 的转换，根据式 (4) 当 $i = 6$ 时 $m_6$ 需要除以 64。为避免整数除法带来的误差，此处将 $m_i$ 放大 64 倍，故单个 $m_i$ 的位宽位 6。考虑代码的简洁性，将 $m_1$ 到 $m_6$ 合并为位宽为 36 的信号。此部分代码详见文件 *sobol_single.v*。

SOBOL 将有效的 SOBOL SINGLE 运算结果进行输出，同时更新 Sobol 序列的下标 count 以便下一 Sobol 值的计算。同样的，在输出时将 SOBOL SINGLE 的使能置零，保持信号的稳定。此部分代码详见文件 *sobol.v*。

### 3.3.2 SC 序列产生模块

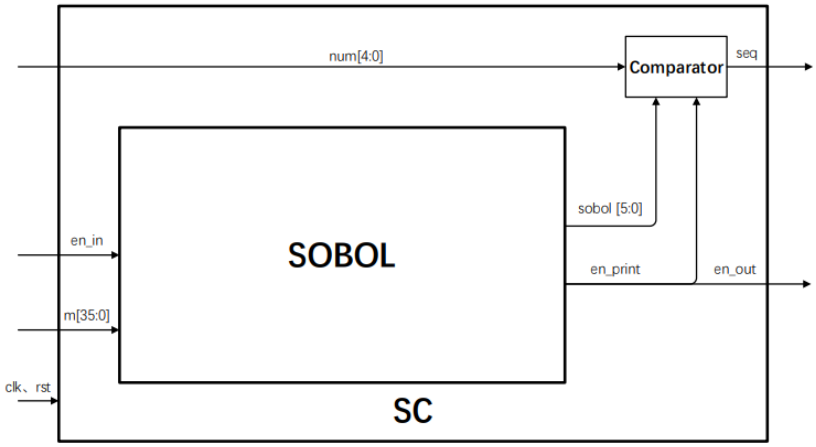



图 9: SC 序列产生模架构图

如上图所示，SC 序列产生模块包含 Sobol 序列生成器和比较器，当 SOBOL 函数的输出有效时，读取输出结果并与输入数据 num 进行比较。若前者较大输出 1，较小输出 0。需要注意的是，受除法计算的限制产生的 Sobol 值位宽为 6，而输入数据的位宽限制为 5。比较时，取 Sobol 序列的高五位与 num 信号进行比较可获得更精确的 SC 序列。每读取一个有效 Sobol 序列可比较产生一位 SC 序列。一个完整的 SC 序列需要进行 32 次比较，也就是 32 个运算周期。此 SC 序列产生模块实现了长度为 32 的 SC 序列在 32 运算周期内单 bit 输出，满足设计要求。此部分代码详见文件 *SC.v*。

### 3.3.3 乘法模块

根据 SC 序列计算原理，二元乘法由与门实现。当 SC 序列产生模块输出有效时，AND 函数进行单比特的与逻辑运算并将输出使能拉高。由于该模块代码较为简单，故不进行电路图展示，代码详见文件 *AND.v*。

### 3.3.4 加法模块

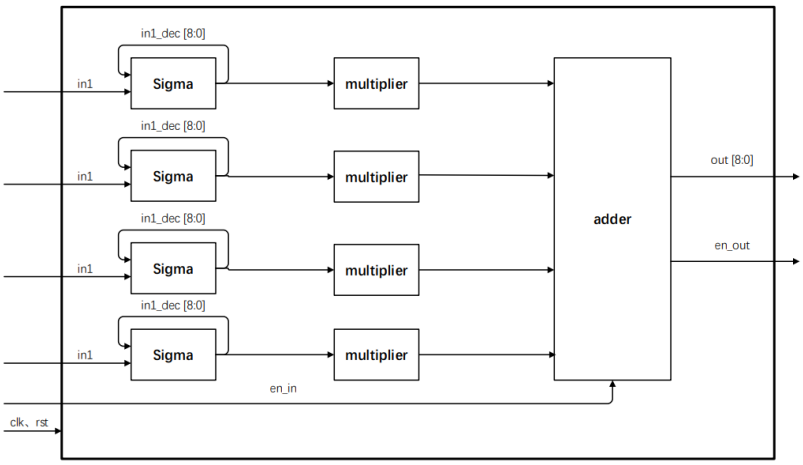



图 10: 加法模块架构图

如上图所示，加法模块 ADD 将四个乘积由 SC 序列转化为二进制再相加得到最终的乘累加计算结果。调用四个累加器在输入有效的时候累加读取的 SC 序列值，32 个周期后即可得到一组 SC 序列中 1 的个数，将累加结果除以 32 可以得到其二进制值。由于整数除法的固有误差，我们通过乘法器将累加结果进行放大。实现电路时，我们将放大倍数设置为 16 倍，具体原因将在后文测试时说明。乘法与加法的部分由组合电路实现，故 32 个周期后，加法模块完成长度为 32 的 SC 序列的转换并输出二进制加法结果，符合设计要求。此部分代码详见文件 $ADD.v$。

### 3.3.5 顶层模块

如图. 7所示，顶层模块调用八个 SC 序列产生模块、四个乘法模块和一个加法模块。对八个输入数据进行运算并最终输出计算结果和指示输出有效的信号。此外，顶层模块还为八个 Sobol 生成器赋予不同的 $m_i$ 值。实现时，不同 Sobol 序列生成器的 $m_i$ 为 1、3、5、7、9、11 这六个奇数的不同排列组合。此部分代码详见文件 $top.v$。

至此，A1 乘累加计算电路的所有模块实现细节描述完毕。

## 3.4 硬件设计验证

### 3.4.1 Testbench 设计

硬件设计验证的思路与软件一致，通过对 MAE 的计算来验证电路的正确性和准确性。由于在 Testbench 中仍然无法实现精确的除法和小数表示，测试时我们将放大后的输出结果进行绝对误差的累加，并以 1000 次计算为一组观察累加结果。

Testbench 实现主要包括随机数输入的产生，绝对误差的计算和累加。调用 $random 函数并将结果对 32 取模可以得到范围在 0-31 的随机数用于模拟放大 32 倍的 0-1 浮点数输入。通过确定输出值与理论计算值的大小关系可以计算出二者的绝对值差。当计数器计数至 1000 时输出累加结果。考虑输出结果放大了 16 倍且累加了 1000 次计算的误差，将累加结果除以 16000 即可得到 MAE 值。

### 3.4.2 验证结果

如图. 11所示，Modelsim 监视界面从左至右分别为当前计算次数、电路计算结果、理论计算结果和当前绝对误差的累加值。由图可见，计算值与理论值较为相近。

```
# count=        993result=  3ideal=     2MAE=       1188
# count=        994result=  7ideal=     5MAE=       1189
# count=        995result=  9ideal=     9MAE=       1191
# count=        996result= 12ideal=    12MAE=       1191
# count=        997result=  7ideal=     5MAE=       1191
# count=        998result= 16ideal=    15MAE=       1193
# count=        999result= 12ideal=    12MAE=       1194
# count=       1000result= 25ideal=    23MAE=       1194
# count=          1result= 18ideal=    16MAE=          2
# count=          2result=  5ideal=     5MAE=          4
# count=          3result=  6ideal=     7MAE=          4
# count=          4result=  9ideal=     8MAE=          5
# count=          5result=  8ideal=     7MAE=          6
# count=          6result= 13ideal=    13MAE=          7
# count=          7result= 10ideal=     9MAE=          7
# count=          8result= 13ideal=    11MAE=          8
# count=          9result=  2ideal=     2MAE=         10
# count=         10result= 20ideal=    20MAE=         10
```

图 11: A1 计算结果展示

如图. 12所示，对多组 1000 次计算的 MAE 进行输出。经过多次测试，绝对误差的累加值约在 1100 与 1250 之间，对应 MAE 值约为 0.069-0.078。该 MAE 值远低于设计指标 0.8 且与 2.3 中 MATLAB 仿真值相近。

```
# MAE=       1161
# MAE=       1181
# MAE=       1137
# MAE=       1173
# MAE=       1175
# MAE=       1205
# MAE=       1192
# MAE=       1173
# MAE=       1167
# MAE=       1213
# MAE=       1176
# MAE=       1206
# MAE=       1172
# MAE=       1145
# MAE=       1154
# MAE=       1207
```

图 12: A1 MAE 结果展示

特别说明的是，理论上考虑加法模块 SC 序列转二进制是除法的精度，电路计算结果应该方法 32 倍以上。但是，考虑 Sobol 算法生成序列的相关性和电路实现的局限性，计算结果放大的同时也伴随着绝对误差的放大。适当的方法倍数需要兼顾除法的精确性又要防止误差的过度放大。经过多次测试，我们最终发现将放大倍数调整为 16 时可以取得最优的 MAE。此外，电路调试过程中还发现，增加 ibit 输入数据的位宽可以使 MAE 值显著降低。

至此，A1 乘累加计算电路的设计与验证全部完成，符合实验要求。

# 4 Verilog A2 实现

提高计算机系统并行度的方法有很多，包括指令级并行（ILP，Instruction-Level-Parallelisim）、线程级并行（TLP，Thread-Level-Parallelisim），数据级并行（DLP，Data-Level-Parallelisim），在这些指导思想下衍生出了超标量，流水线，多核等具体的优化方法。在 VLSI 硬件设计中，我们通常采用**展开**的方法以提高系统 DLP [3] [4]。

A2 乘累加计算单元基于前文提到的 A1 电路结构，其中 SC 序列生成器每周期并行生成 2 个 Sobol 序列，通过提高硬件开销的方式换取处理性能的提升。本部分讲逐一说明 A2 各模块的设计思路，以及验证方式和结果。 [2]

## 4.1 硬件设计思路

A2 并行结构的性能提升主要来源于对 A1 电路图 (Fig. 7) 中 SC 生成单元进行 2 阶展开。展开后，SC 模块每周期将会并行生成 2 个序列比特。SC 同样包括前文提到的 Sobol 序列生成器。乘累加计算的部分和 A1 类似。除此之外，电路由顶层模块对 A2 进行封装，最终实现并行发生功能。个模块的实现细节将在进行说明。

## 4.2 电路图

A2 的电路图如 Fig. 13所示，我会对每一部分进行详细说明：

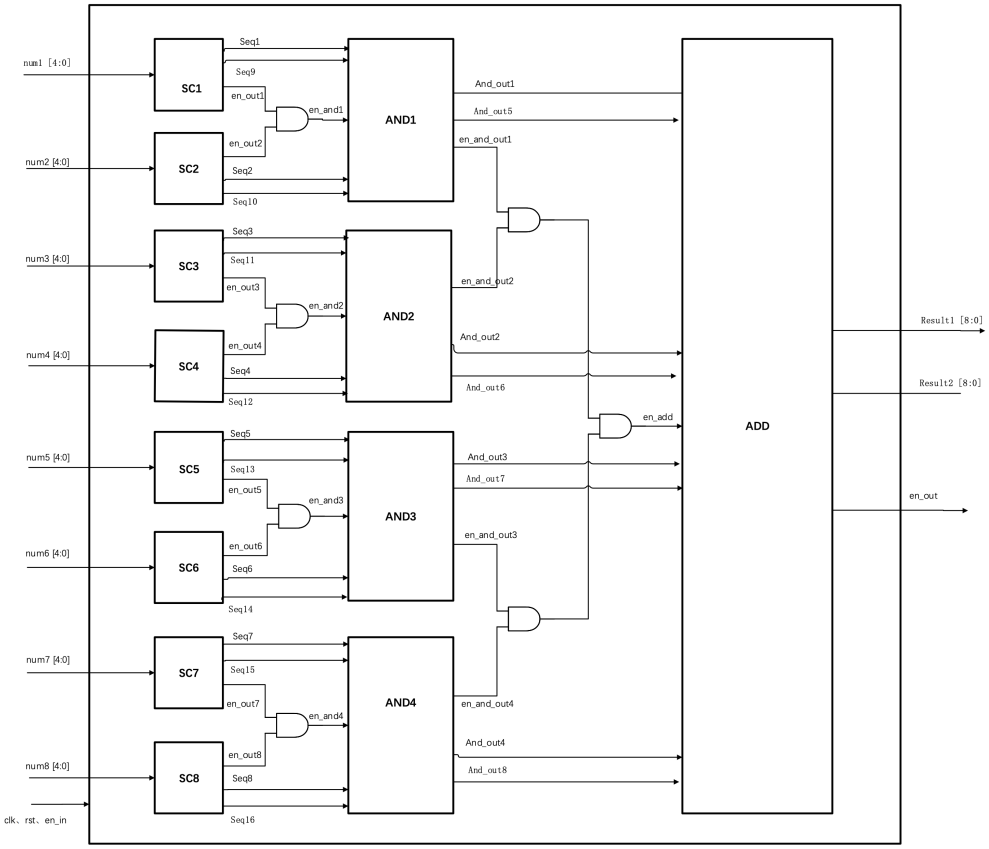



图 13: A2 Architecture Diagram

注意到，A2 的架构的核心在于对 SC 随机序列发生器进行了展开，因此其并行度得到了提高，这样，A2 的处理速度相对于 A1 有了 2× 的增益。图13所示的其他部分 AND 和 ADD 乘累加电路和 A1 类似。整个架构图包括 8 个 5bit 输入数据，使能信号，reset 信号。输出包括 2 个计算结果`result1`和`result2`，以及一个输出使能信号。实现功能和 A1 类似，具体可以参照 Chapter3.2 的表述。

## 4.3 RTL 各模块介绍

### 4.3.1 SC 随机序列发生器

A2 的 SC 序列发生器如 Fig. 14所示

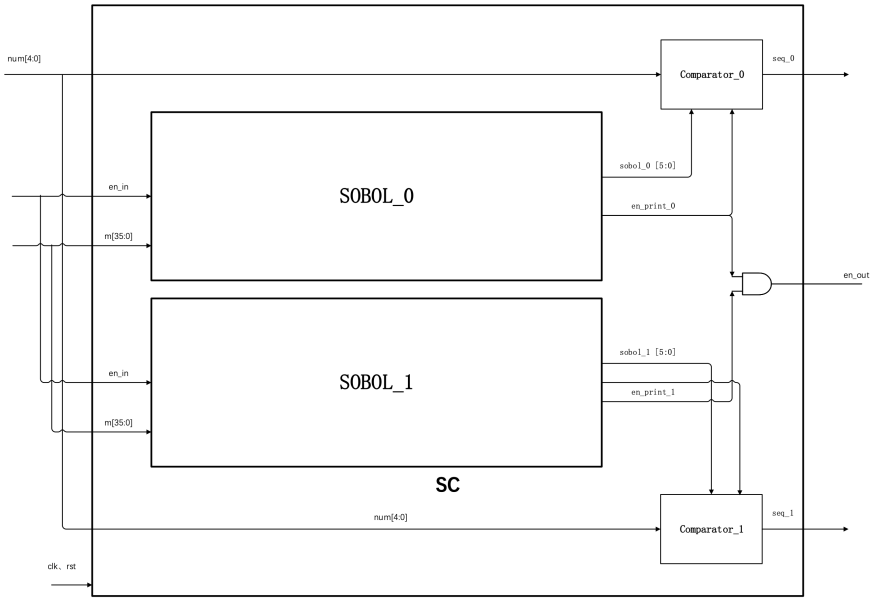



图 14: A2 SC Architecture

图14的结果即是 A1 中 SC 序列发生器的二阶展开。在这里，把`SOBOL`和`Comparator`看作 DFG 中两个节点，通过二阶展开后得到复制，分别具有相同功能的模块`SOBOL_1`和`SOBOL_0`。`SOBOL_1`负责计算奇数时刻的序列发生，输入`x(2k+1)`和输出`y(2k+1)`,`SOBOL_0`负责计算偶数时刻的序列发生，输入`x(2k+0)`和输出`y(2k+0)`. 原始架构中并不存在`SOBOL`和`Comparator`的路径延迟，因此也不必讨论展开图新路径的延迟分配问题。

### 4.3.2 Sobol 序列发生器

Sobol 序列发生器在图14中以`SOBOL_0`和`SOBOL_1`展示，具体的架构和 A1 类似，这里不做过多阐述。唯一需要注意的是，因为`SOBOL_1`和`SOBOL_0`分别处理奇数时刻和偶数时刻，为了复用 A1 代码，两者的`counter`对应的初始值有所区别。通过`SOBOL_UNIT`完成核心的 XOR 计算，更新`counter`，确定 Least-Zero 的位置下标。`counter`仍然在 0-31 循环，只不过步长（stride）被调整成了 2，从而实现逻辑意义上的奇数偶数区分。

根据 Chapter2.2 的 Sobol 序列生成原理，每次计算 $X_n$ 时，需要知道上一时刻 $x_{n-1}$ 的值，这意味着SOBOL_1和SOBOL_2存在数据依赖，我借鉴了编译器的设计方法，提出了 **forwarding** 的技术以避免数据依赖带来的功能错误，并提高并行度。

**Forwarding**：以SOBOL_1举例子，当生成了 $x_3$ 时，SOBOL_0也生成了 $x_4$，若没有 forwarding，SOBOL_1需要等到 $x_4$ 在下一周期传入并用于计算 $x_5$，通过增加一条直接从SOBOL_1的xtmp寄存器（见图 8）到SOBOL_0的xtmp的数据通路，两个 Sobol 发生器可以提前知道对方的此时的计算情况，从而减少了数据通路的 stall。实现部分参照代码SOBLE.v [1]

至此，核心部件 SC 发生器可以实现在一个周期中输出两个序列比特seq_0和seq_1，可以在 16 个周期内完成 32 比特的序列输出，满足 assignment 的条件。

### 4.3.3 AND

乘法模块和 A1 类似，十分容易，唯一的区别就是输入变成了 4 个数据，AND 会按序比特比较前一组两个输入和后一组两个输入数据，并将使能信号拉高。

### 4.3.4 ADD

加法模块和 A1 类似，实现原理可以参照 Chapter3.3.4. 唯一的区别就是输入数据变为了 8 个，具体的架构如图15所示。通过复制一套硬件实现 ADD 的并行输出。

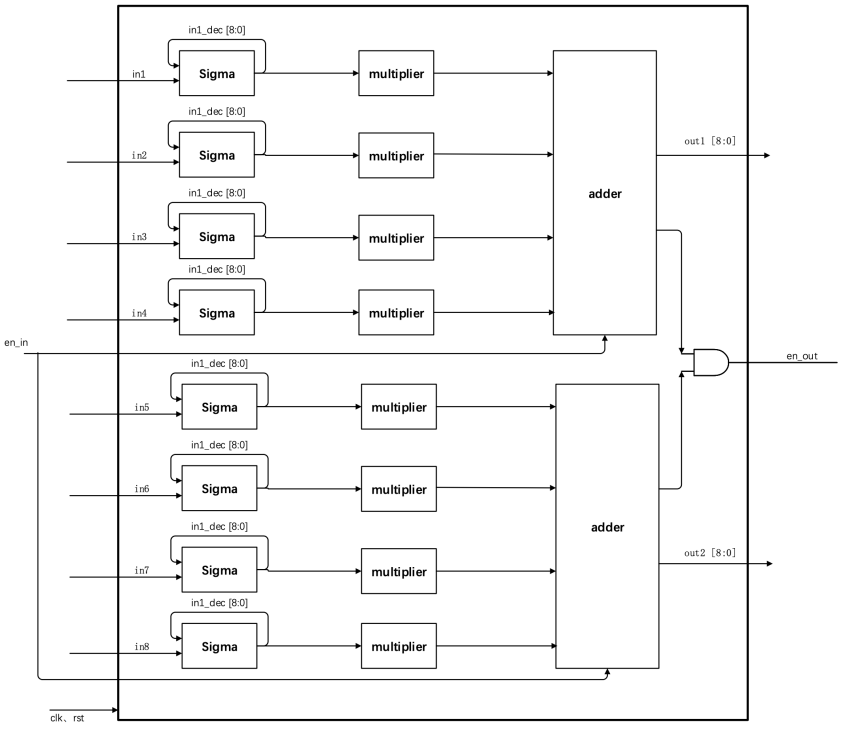



图 15: A2 ADD Architecture

### 4.4 硬件设计验证

验证硬件系统的正确性的方法和 A1 相似，`testbench`也格外类似，具体的设计思路可以参照 Chapter3.4.1。这里就不做详细阐述了。

验证在 Modelsim 10.2c 中进行。

### 4.5 验证结果

A2 系统正确性的验证如下：

如图.16所示，Modelsim 监视界面从左至右为当前计算次数，并行输出的两个结果`result1`和`result2`，绝对误差的累加。和 A1 一样，对多组 1000 次计算的误差累加值进行算术平均（如图17），可以得到对应的 MAE 大约在 **0.075 - 0.0875**，这与 A1 和 MATLAB 的软件结果（MAE ≈ 0.09）的结果很吻合，同时也远低于 assignment 要求的 0.8 的设计指标。

```
# count=        985result1= 12result2=        12MAE=       1274
# count=        986result1= 15result2=        16MAE=       1276
# count=        987result1= 16result2=        23MAE=       1276
# count=        988result1= 16result2=        19MAE=       1276
# count=        989result1= 13result2=        20MAE=       1277
# count=        990result1= 20result2=        20MAE=       1277
# count=        991result1=  8result2=        14MAE=       1278
# count=        992result1= 22result2=        22MAE=       1278
# count=        993result1= 10result2=        15MAE=       1279
# count=        994result1= 17result2=        22MAE=       1279
# count=        995result1= 13result2=        18MAE=       1279
# count=        996result1=  6result2=        14MAE=       1280
# count=        997result1= 22result2=        30MAE=       1280
# count=        998result1= 35result2=        36MAE=       1283
# count=        999result1= 17result2=        24MAE=       1284
# count=       1000result1= 15result2=        20MAE=       1285
```

图 16: A2 Test Result

```
VSIM 11> run -all
# MAE=       1210
# MAE=       1203
# MAE=       1297
# MAE=       1208
# MAE=       1369
# MAE=       1325
# MAE=       1307
# MAE=       1299
# MAE=       1178
# MAE=       1284
# MAE=       1340
# MAE=       1170
# MAE=       1220
# MAE=       1354
# MAE=       1233
# MAE=       1269
# MAE=       1349
# MAE=       1303
# MAE=       1295
# MAE=       1298
# MAE=       1213
# MAE=       1356
```

图 17: A2 MAE Result

至此完成了 A2 所有的设计和验证部分。[演示视频]

# 5 Vivado 综合结果

## 5.1 综合环境

综合的环境配置如下：

- Operating System Kernel：Windows10(Kernel:19042.1586). VMware12.5
- Software Version：Vivado 2017.3
- 电路板：ARM Cortex-VC709

## 5.2 A1 综合结果

图18展示了 A1 的综合结果，我们从 Schematic 可以看到综合出的电路图和前端设计的电路图逻辑功能一致，一定程度上又证明了我们的功能正确性。

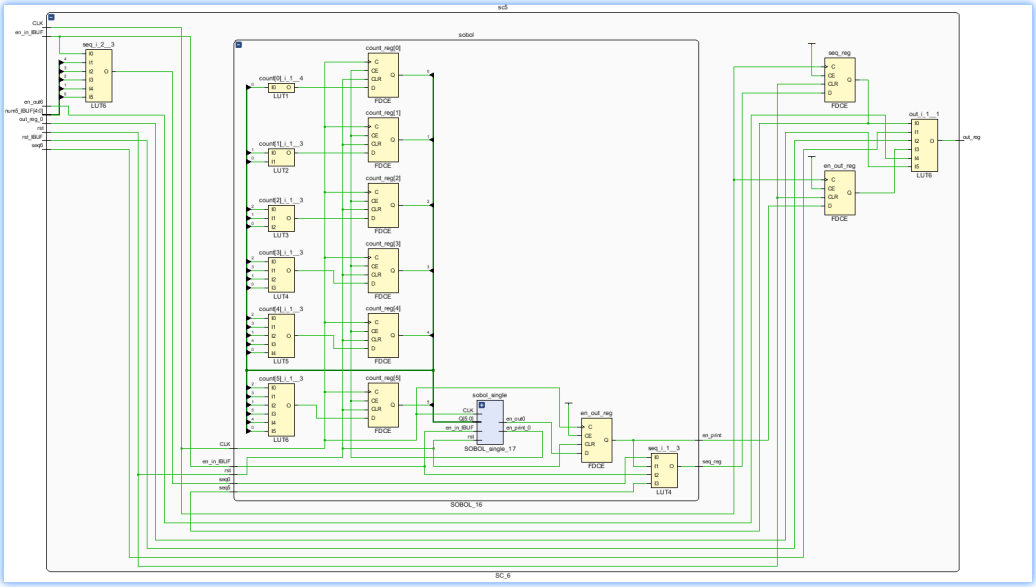


图 18: A1 Synthesis Diagram Result

- **性能**表现如图19：

  在 Vivado 设置时间约束条件，具体如下：

```
# define clock and period
create_clock -period 10.000 -name clk -waveform {0.000 5.000} [get_ports
    clk]
# frequency=10MHz
```

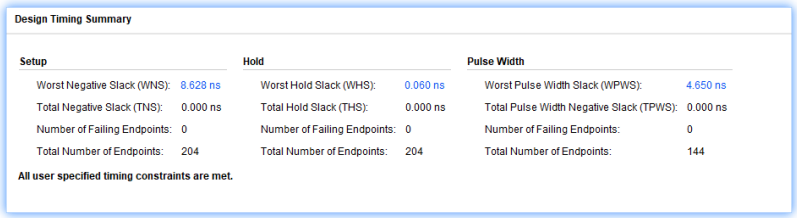



图 19: A1 Synthesis Timing Report

- **功耗**表现如图20:

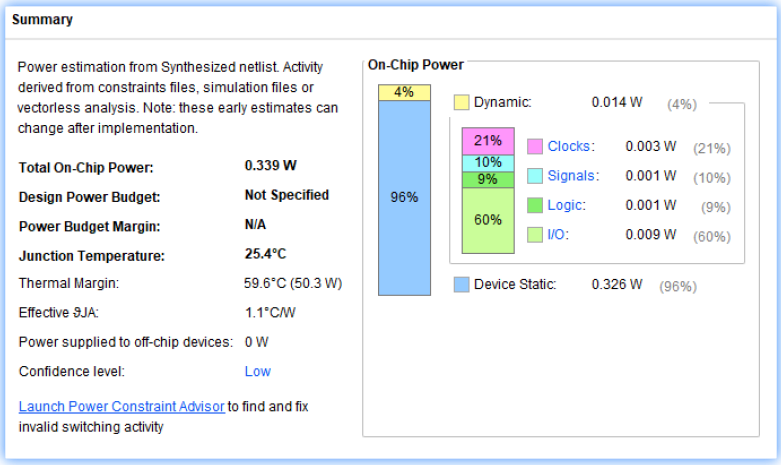



图 20: A1 Synthesis Power Report

- **资源占用**表现如图21:

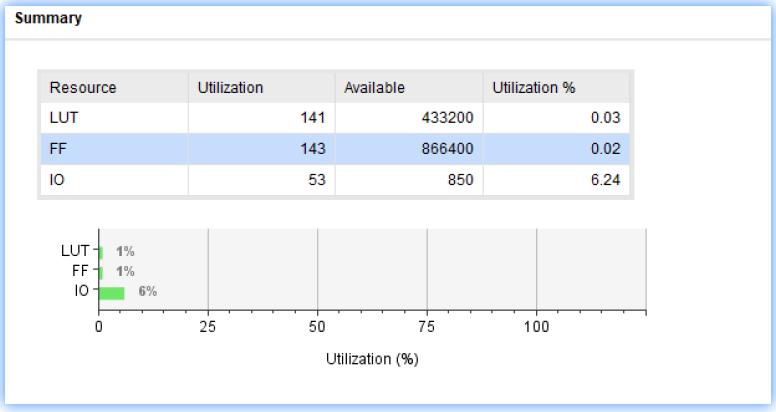



图 21: A1 Synthesis Resoures Report

## 5.3 A2 综合结果

A2 和 A1 的主要区别体现在综合结果资源开销更大，功耗更高。结果表示，A2 也能工作在和 A1 相同的频率上，但因为 A2 是两阶并行架构的，一个时钟周期输出了两个序列比特，从而可以体现出 A2 在计算速度上的提升。

具体的综合结果如下：

**性能**表现如图22：

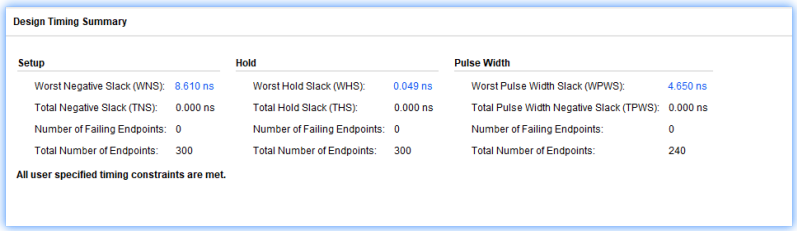



图 22: A2 Synthesis Timing Report

A2 的时间约束和 A1 相同。

**功耗**表现如图23：

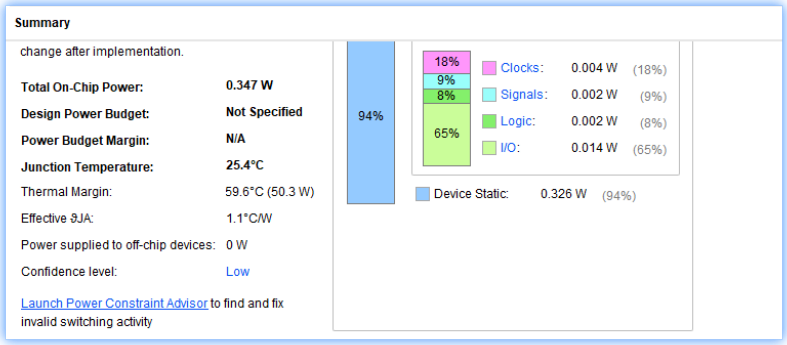



图 23: A2 Synthesis Power Report

**资源占用**表现如图24：

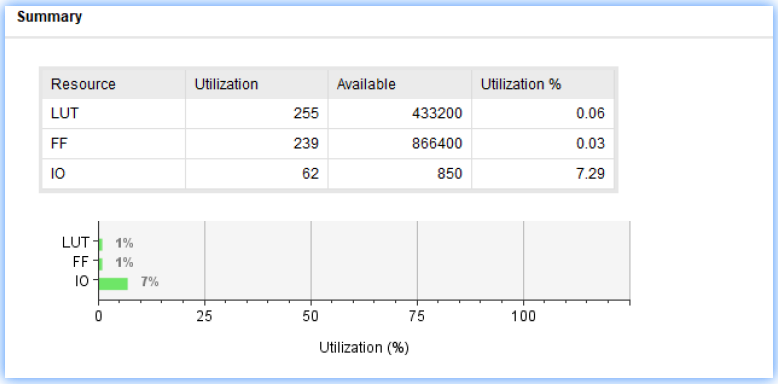



图 24: A2 Synthesis Resource Report

综上所述，A1,A2 的综合结果可以总结成表1：

表 1: A1&A2 Synthesis Results

| Arch | Timing | Power | Resourse |
|---|---|---|---|
| A1 | **10MHz** | **0.338W** | LUT(**141**), FF(**143**), IO(**53**) |
| A2 | **10MHz**(2-bit per clock cycle) | **0.347W** | LUT(**255**), FF(**239**), IO(**62**) |

# 6 成员分工

组长：林羽
组员：王清训 李卓壕
**各自分工：**
林羽：A1 硬件设计以以及仿真验证
王清训：matlab 软件设计以及仿真验证
李卓壕：A2 硬件设计以及仿真验证，A1/A2 综合

# 参考文献

[1] Alaghi, A., and Hayes, J. P. Fast and accurate computation using stochastic circuits. In *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)* (2014), IEEE, pp. 1–4.

[2] Bratley, P., and Fox, B. Implementing sobols quasirandom sequence generator (algorithm 659). *ACM Transactions on Mathematical Software 29*, 1 (2003), 49–57.

[3] Hennessy, J. L., and Patterson, D. A. *Computer architecture: a quantitative approach*. Elsevier, 2011.

[4] Hwang, K., and Faye, A. Computer architecture and parallel processing.