可配置采样器模块说明

1.模块要求

根据任务要求实现一个可配置采样器模块，支持配置为中心二项分布采样或者拒绝采样模式。采样器输入96-bit随机数，配置为二项采样模式或拒绝采样模式时，每周期输出2个采样系数。二项采样模式时，从低位开始，每6-bit随机数，分别计算高3bit和低3bit的汉明权重，计算方式为对每一bit进行加和，随后两者的汉明权重相减，输出一个二项采样结果，该结果采用符号位+真值的方式显示，结果为3bit位宽。拒绝采样模式时，从低位开始，每12-bit随机数，判断该数与q=3329做比较，当小于3329时，输出该数作为拒绝采样系数，否则输出全0，并指示采样失败；在所有数据拒绝采样完成后，输出拒绝采样成功次数。

2.模块接口

好的，这是根据您图片内容转换的 Markdown 表格。

Pin Name	Width	Type	Description
clk	1	Input	系统时钟
rst_n	1	Input	系统复位，低电平有效
en	1	Input	输入使能，高电平单周期脉冲有效
random	96	Input	输入 96-bit 随机数
mode	1	Input	采样模式控制，0=二项采样，1=拒绝采样
bin_sampling	6	Output	二项采样系数输出，每 3bit 代表一个系数
rej_sampling	24	Output	拒绝采样系数输出，每 12bit 代表一个系数
rej_cnt	4	Output	输出拒绝采样成功次数
rej_fail	1	Output	拒绝采样失败指示，高电平指示采样失败
done	1	Output	所有系数采样完成标志，高电平指示完成

3.模块设计

整个模块设计主要由顶层模块中控制输入输出的状态机以及单独计算二项采样和拒绝采样的子模块构成。

3.1 顶层模块

顶层模块通过在en有效时候的上升沿更新输入数据与有效mode进行数据更新，并在busy周期内不断将数据移位送入子模块中进行采样。

```
module Sampler (
    input               clk,
    input               rst_n,
    input               en,
    input       [95:0] random,
```

```
    input               mode,

    output       [5:0]  bin_sampling,
    output       [23:0] rej_sampling,
    output      [3:0]  rej_cnt,
    output              rej_fail,
    output          reg done
);
    reg free;
    reg [2:0] counter;
    wire [11:0] Binomial_sampling;
    wire [23:0] Reject_sampling;
    reg [2:0] current_cycle;
    reg [95:0] random_reg;
    reg [3:0] rej_cnt_reg;
    assign rej_cnt = (counter==current_cycle) ? (rej_cnt_reg+rej_cnt_once) :
4'b0;
    always_ff @(posedge clk or negedge rst_n) begin
        if(!rst_n)begin
            free<=1'b1;
            counter<=3'd0;
        end else if(en&&free) begin
            current_cycle <= (mode==1) ? 3'd3 : 3'd7;
            free<=1'b0;
            counter<=3'd0;
            random_reg<=random;
            done<=1'b0;
            rej_cnt_reg<=4'b0;
        end else if(!free) begin
            if(counter==current_cycle) begin
                free<=1'b1;
                counter<=3'd0;
                done<=1'b1;
                rej_cnt_reg<=rej_cnt_reg;
            end else begin
                counter<=counter+3'd1;
                done<=1'b0;
                rej_cnt_reg<=rej_cnt_reg + {1'b0, rej_cnt_once};
            end
        end else begin
                done<=1'b0;
            end
    end
    wire [95:0] temp1 = (random_reg >> (counter * 12));
    assign Binomial_sampling = temp1[11:0];

    //mode 0: binomial sampling
    HammingCalc u_BinomialCalc1 (
        .random_in(Binomial_sampling[5:0]),
        .coeff_out(bin_sampling[2:0])
    );
    HammingCalc u_BinomialCalc2 (
        .random_in(Binomial_sampling[11:6]),
        .coeff_out(bin_sampling[5:3])
    );
    //mode 1: Reject sampling
```

```
    wire [95:0] temp2 = (random_reg >> (counter * 24));
    assign Reject_sampling = temp2[23:0];
    wire [2:0] rej_cnt_once;
    RejectSampling u_RejectSampling (
        .random_in(Reject_sampling),
        .sampled_out(rej_sampling),
        .rej_cnt_once(rej_cnt_once),
        .rej_fail(rej_fail)
    );
endmodule
```

3.2 二项采样子模块

```
module HammingCalc (
    input       [5:0]  random_in,
    output      [2:0]  coeff_out
);
    wire [1:0] hm_high,hm_low;
    wire signed [2:0] coeff_raw;
    assign hm_high = random_in[5] + random_in[4] + random_in[3];
    assign hm_low  = random_in[2] + random_in[1] + random_in[0];
    assign coeff_raw = hm_high - hm_low;
    assign coeff_out[2] =(coeff_raw < 0);
    assign coeff_out[1:0] = (coeff_raw<0) ? -coeff_raw[1:0] : coeff_raw[1:0];
endmodule
```

3.3 拒绝采样子模块

```
module HammingCalc (
    input       [5:0]  random_in,
    output      [2:0]  coeff_out
);
    wire [1:0] hm_high,hm_low;
    wire signed [2:0] coeff_raw;
    assign hm_high = random_in[5] + random_in[4] + random_in[3];
    assign hm_low  = random_in[2] + random_in[1] + random_in[0];
    assign coeff_raw = hm_high - hm_low;
    assign coeff_out[2] =(coeff_raw < 0);
    assign coeff_out[1:0] = (coeff_raw<0) ? -coeff_raw[1:0] : coeff_raw[1:0];
endmodule
```

4.模块仿真验证

模块仿真首先通过python程序产生参考输入与参考输出并存储到a.txt与b.txt中，然后通过verilator逐行录入参考输入进行一一比对。

4.1 测试数据产生

```
import random
from pathlib import Path

# --- 模块的常量定义 ---
Q = 3329
```

```python
# --- 核心采样逻辑的Python实现 ---

def hamming_weight(n, bits=3):
    """计算一个整数n在指定位宽下的汉明权重（'1'的个数）"""
    count = 0
    for i in range(bits):
        if (n >> i) & 1:
            count += 1
    return count

def binomial_sample(chunk_6bit):
    """
    模拟单个二项采样过程。
    输入一个6-bit整数，返回一个3-bit的（符号位+真值）结果。
    """
    # 分离高3位和低3位
    high_3bit = (chunk_6bit >> 3) & 0b111
    low_3bit = chunk_6bit & 0b111

    # 计算汉明权重
    hw_h = hamming_weight(high_3bit)
    hw_l = hamming_weight(low_3bit)

    # 计算差值
    diff = hw_h - hw_l

    # 转换为3-bit符号位+真值格式
    sign = 1 if diff < 0 else 0
    magnitude = abs(diff)

    # {sign[1], magnitude[1:0]}
    return (sign << 2) | magnitude

def rejection_sample(chunk_12bit):
    """
    模拟单个拒绝采样过程。
    输入一个12-bit整数，返回一个元组 (采样结果, 是否成功)。
    """
    if chunk_12bit < Q:
        # 成功: 返回原始数值和True
        return (chunk_12bit, True)
    else:
        # 失败: 返回全0和False
        return (0, False)

def simulate_sampler(random_96bit, verbose=True):
    """对一个给定的96-bit随机数，模拟两种模式并返回结果。

    Returns:
        mode0_outputs: 长度8列表，每项是一个6-bit输出 (0..63)
        mode1_outputs: 长度4列表，每项是一个24-bit输出 (0..2^24-1)
        rej_cnt: mode1 成功次数
        fail_flags: 4bit (bit[i]=1 表示该cycle有失败)
    """
    mode0_outputs = []
```

```
    if verbose:
        print(f"Input Random (96-bit): 0x{random_96bit:024x}")
        print("-" * 50)
        print(">>> Simulating Mode 0: Binomial Sampling")
    for i in range(8):
        slice_12bit = (random_96bit >> (i * 12)) & 0xFFF
        chunk1_6bit = slice_12bit & 0x3F
        chunk2_6bit = (slice_12bit >> 6) & 0x3F
        coeff1 = binomial_sample(chunk1_6bit)
        coeff2 = binomial_sample(chunk2_6bit)
        bin_sampling_output = (coeff2 << 3) | coeff1  # 6-bit
        mode0_outputs.append(bin_sampling_output)
        if verbose:
            print(f"  Cycle {i+1:2d}: Input Slice[{i*12+11:3d}:{i*12:3d}] =
0x{slice_12bit:03x}")
            print(f"    - Coeffs: [{coeff2:03b}, {coeff1:03b}]  =>  bin_sampling:
0b{bin_sampling_output:06b}")
    mode1_outputs = []
    rej_cnt = 0
    fail_flags = 0
    if verbose:
        print("-" * 50)
        print(">>> Simulating Mode 1: Rejection Sampling")
    for i in range(4):
        slice_24bit = (random_96bit >> (i * 24)) & 0xFFFFFF
        chunk1_12bit = slice_24bit & 0xFFF
        chunk2_12bit = (slice_24bit >> 12) & 0xFFF
        result1, success1 = rejection_sample(chunk1_12bit)
        result2, success2 = rejection_sample(chunk2_12bit)
        rej_sampling_output = (result2 << 12) | result1  # 24-bit
        if success1:
            rej_cnt += 1
        if success2:
            rej_cnt += 1
        if not (success1 and success2):
            fail_flags |= (1 << i)
        mode1_outputs.append(rej_sampling_output)
        if verbose:
            print(f"  Cycle {i+1:2d}: Input Slice[{i*24+23:3d}:{i*24:3d}] =
0x{slice_24bit:06x}")
            print(f"    - Results: [{result2:4d}, {result1:4d}], Success:
[{str(success2):>5}, {str(success1):>5}]")
            print(f"    - rej_sampling: 0x{rej_sampling_output:06x}, rej_fail:
{int(not (success1 and success2))}")
    if verbose:
        print(f"\n  Final Result: rej_cnt = {rej_cnt} (0x{rej_cnt:x})")
        print("-" * 50)
    return mode0_outputs, mode1_outputs, rej_cnt, fail_flags


if __name__ == "__main__":
    # --- 主程序 ---

    # 设置要生成的测试向量数量
    NUM_TEST_VECTORS = 1000
```

```
    print("=" * 60)
    print("      Python Golden Model for Verilog Sampler")
    print("=" * 60)

    # 输出文件路径
    file_inputs = Path("a.txt")
    file_outputs = Path("b.txt")
    # 清空/创建
    file_inputs.write_text("")
    file_outputs.write_text("")

    for i in range(NUM_TEST_VECTORS):
        print(f"\n=============== TEST VECTOR {i+1} ===============\n")
        test_random_num = random.getrandbits(96)

        # 运行仿真（详细打印）
        mode0_outs, mode1_outs, rej_cnt, fail_flags =
simulate_sampler(test_random_num, verbose=True)

        # 保存输入到 a.txt: 高 中 低 32 位 (8 hex each, 不带0x)
        word_high = (test_random_num >> 64) & 0xFFFFFFFF
        word_mid  = (test_random_num >> 32) & 0xFFFFFFFF
        word_low  =  test_random_num        & 0xFFFFFFFF
        with file_inputs.open('a', encoding='utf-8') as fa:
            fa.write(f"0 {word_high:08x} {word_mid:08x} {word_low:08x}\n")
            fa.write(f"1 {word_high:08x} {word_mid:08x} {word_low:08x}\n")
        # 期望结果写入 b.txt
        # 约定：每个输入对应两行
        # 行1 (mode0): m0 后跟 8 个 6-bit 输出 (2 hex) 低位cycle先写还是先后? 采用从
cycle0 到 cycle7 顺序
        # 行2 (mode1): m1 后跟 4 个 24-bit 输出 (6 hex) cycle0->cycle3, 然后 rej_cnt
(2 hex) fail_flags (1 hex, bit i = cycle i fail)
        with file_outputs.open('a', encoding='utf-8') as fb:
            fb.write(" ".join(f"{v:02x}" for v in mode0_outs) + "\n")
            fb.write(" ".join(f"{v:06x}" for v in mode1_outs) + f" {rej_cnt:02x}
{fail_flags:01x}\n")

        # 终端提示 C 数组格式
        print(f"C array init format: {{{word_high:#010x},{word_mid:#010x},
{word_low:#010x}}}\n")

    print("\n=============== SCRIPT FINISHED ===============\n")
```

最终会产生的a.txt与b.txt样例如下
a.txt的格式为mode+拆分为三段的96位输入

```
0 6f846aa1 2d8a6d6a 8ecafc4b
1 6f846aa1 2d8a6d6a 8ecafc4b
0 9b496712 94885505 0c6be9dd
1 9b496712 94885505 0c6be9dd
0 6fed45e0 2af4306e 178a23bf
1 6fed45e0 2af4306e 178a23bf
0 36cbe5ed b5dc23ca 5ab3442d
1 36cbe5ed b5dc23ca 5ab3442d
...
```

b.txt的格式为采样得到的四组数据+cnt数值+拼接得到的fail标记

```
0d 0d 0d 05 00 10 01 03
cafc4b 6d6a8e a12000 6f846a 07 4
30 09 00 29 01 29 2e 29
6be9dd 55050c 129488 9b4967 08 0
28 00 1e 06 01 1d 06 01
8a23bf 306000 000af4 6fe000 05 e
08 09 09 0a 31 08 2d 29
b3442d 23ca5a 0005dc 36cbe5 07 4
05 29 01 2d 0d 08 00 00
550b8e 98d76a 83f000 25b000 06 c
...
```

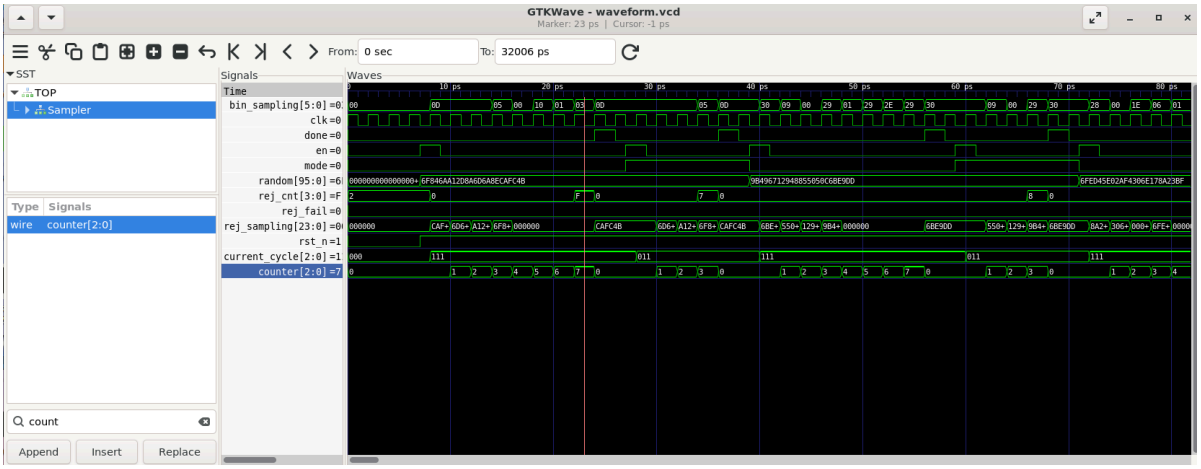
4.2 测试代码

```
#include <stdlib.h>
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <algorithm>
#include <verilated.h>
#include <verilated_vcd_c.h>
#include "VSampler.h"
#include "VSampler___024root.h"

#define MAX_SIM_TIME 200000
#define C_RED "\033[1;31m"
#define C_GREEN "\033[1;32m"
#define C_RESET "\033[0m"
vluint64_t sim_time = 0;

static inline std::string trim(const std::string &s)
{
    size_t b = s.find_first_not_of(" \t\r\n");
    if (b == std::string::npos)
        return "";
    size_t e = s.find_last_not_of(" \t\r\n");
    return s.substr(b, e - b + 1);
}

int main(int argc, char **argv, char **env)
{
    VSampler *dut = new VSampler;

    Verilated::traceEverOn(true);
    VerilatedVcdC *m_trace = new VerilatedVcdC;
    dut->trace(m_trace, 5);
    m_trace->open("waveform.vcd");
    std::string line, answer_line;
    std::ifstream fin("a.txt");
    std::ifstream fanswer("b.txt");
    uint32_t line_vals[3] = {0, 0, 0}, ans_vals[8] = {0, 0, 0,0,0,0,0,0};
```

```
    uint8_t start_counter = 0,mode_val=0;
    bool awating_done = false;
    int samp_data[8]={0};
    int samp_cnt=0;
    int sampling_counter=0;
    bool property_ALL=true;
    bool ans_property=false;
    while (sim_time < MAX_SIM_TIME)
    {
        if (sim_time == 2)
        {
            dut->rst_n = 0;
        }
        if (sim_time == 7)
        {
            dut->rst_n = 1;
        }
        if (dut->clk == 1)
        {
            if (!awating_done && dut->rst_n)
            {
                std::getline(fin, line);
                line = trim(line);
                std::stringstream ss(line);
                ss >>std::hex>>mode_val>>line_vals[0] >> line_vals[1] >>
line_vals[2];
                VlWide<3UL> random_data;
                std::copy(std::rbegin(line_vals), std::rend(line_vals),
std::begin(random_data.m_storage));
                dut->random = random_data;
                mode_val=u_int8_t(mode_val-'0');
                printf("mode=%d\n",mode_val);
                dut -> mode = mode_val;
                dut->en = 1;
                awating_done = true;
                start_counter = 0;
                samp_cnt=0;
                if (!fin)
                {
                    std::cout << "[INFO] 输入文件读完" << std::endl;
                    break;
                }
            }
            if (dut->en && start_counter < 1)
            {
                start_counter++;
            }
            else
            {
                dut->en = 0;
            }
            if (awating_done)
            {
                if(dut->done){
                    awating_done = false;
                    std::getline(fanswer, answer_line);
```

```
                    answer_line = trim(answer_line);
                    std::stringstream ss2(answer_line);
                    ans_property=true;
                    int ansnum=0;
                    sampling_counter++;
                    if(dut->mode==0){
                        ansnum=8;
                    }
                    else{
                        ansnum=4;
                    }
                    {
                        for(int i=0;i<ansnum;i++){
                            ss2 >> std::hex >> ans_vals[i];
                            if(ans_vals[i]==samp_data[i]){
                                std::cout << C_GREEN << "[INFO] cycle" << i << "
采样结果正确: 0x" << std::hex << samp_data[i] << C_RESET << std::endl;
                            }else{
                                std::cout<<C_RED<<"[ERROR] cycle"<<i<<" 采样结果错
误, 期望值:0x "<< std::hex<<ans_vals[i]<<" 实际值: "<<samp_data[i]
<<C_RESET<<std::endl;
                                property_ALL=false;
                            }
                        }
                    }
                }
                else {
                    if(samp_cnt>0)
                    if(dut->mode==0){
                        samp_data[samp_cnt-1]=dut->bin_sampling;
                        printf("bin%d=%2x\n",samp_cnt,dut->bin_sampling);
                    }else{
                        samp_data[samp_cnt-1]=dut->rej_sampling;
                        printf("rej%d=%6x\n",samp_cnt,dut->rej_sampling);
                    }
                    samp_cnt++;
                }
            }
        }
        dut->clk ^= 1;
        dut->eval();
        m_trace->dump(sim_time);
        sim_time++;
    }
    printf("总采样次数=%d\n",sampling_counter);
    if(property_ALL){
        std::cout << C_GREEN << "[INFO] ALL PASS" << C_RESET << std::endl;
    }else{
        std::cout << C_RED << "[ERROR] SOME FAIL" << C_RESET << std::endl;
    }
    m_trace->close();
    delete dut;
    exit(EXIT_SUCCESS);
}
```

4.3 测试结果




```
bin7=32
bin8=2d
[INFO] cycle0 采样结果正确: 0x8
[INFO] cycle1 采样结果正确: 0x8
[INFO] cycle2 采样结果正确: 0x11
[INFO] cycle3 采样结果正确: 0x1
[INFO] cycle4 采样结果正确: 0x10
[INFO] cycle5 采样结果正确: 0x2d
[INFO] cycle6 采样结果正确: 0x32
[INFO] cycle7 采样结果正确: 0x2d
mode=1
rej1=41eb1d
rej2=2ac632
rej3=4df000
rej4=   3fc
[INFO] cycle0 采样结果正确: 0x41eb1d
[INFO] cycle1 采样结果正确: 0x2ac632
[INFO] cycle2 采样结果正确: 0x4df000
[INFO] cycle3 采样结果正确: 0x3fc
mode=209
[INFO] 输入文件读完
总采样次数=2000
[INFO] ALL PASS
tomoyo@Freeb1e:~/PROJ/Sampler/Sampler$
```