北京时间 **Thursday, January 1, 2026**，深夜。

**IC 工程级审查模式：全链路重构指令 (Full-Stack Refactoring Protocol)。**

收到。这是最正确的学习姿势。

你现在不需要当“打字员”，你需要当“翻译官”——把下面的人类语言（需求文档）翻译成Verilog 语言（代码）。

这是你需要手写的三个文件的详细“白话文需求书”。请仔细研读每一个字，**先在脑子里（或纸上）画出图，再动手写代码。**

### 第一部分：核心引擎 lfsr.v (伪随机数发生器)

**设计目标**：造一个能自己产生乱数的 8 位寄存器。

#### 1. 端口定义 (Port Definition)

* **输入**：
  + clk：心脏跳动的节拍。
  + rst\_n：强制重置信号（0 表示复位）。
* **输出**：
  + data\_out：8 位宽 ([7:0])，输出当前的随机数。

#### 2. 内部元件 (Internal Components)

* 你需要定义一个 8 位的**寄存器** (reg) 来存储当前的数值。给它起个名字，比如 r\_lfsr。

#### 3. 逻辑行为 (Logic Description)

* 创建一个 always 块，它只在 clk 的**上升沿** (posedge) 或 rst\_n 的**下降沿** (negedge) 触发。
* **复位逻辑**：
  + 如果 rst\_n 是 0，必须给寄存器赋一个**非零的初值**（例如 8'hFF）。*注意：LFSR 如果是全 0 就死机了，永远转不起来，所以必须给初值。*
* **正常工作逻辑**：
  + 如果 rst\_n 是 1，执行移位操作：
  + 算出**反馈位 (Feedback)**：把寄存器的 **第7位、第5位、第4位、第3位** 拿出来做**异或 (XOR, ^)** 运算。
  + **移位**：把原来的 r\_lfsr 向左推一位（第 6 位变第 7 位，第 5 位变第 6 位...）。
  + **填补**：把刚才算出来的“反馈位”，填补到最右边空出来的 **第 0 位**。
  + *(提示：这步可以用拼接语法 {r\_lfsr[6:0], feedback} 来实现)*

#### 4. 输出连接 (Output Assignment)

* 用 assign 把内部寄存器 r\_lfsr 的值直接连到外部引脚 data\_out。

### 第二部分：顶层主板 top.v (系统集成与胶水逻辑)

**设计目标**：这是一块“主板”，你要把买来的 CPU (UART) 和自己造的显卡 (LFSR) 插上去，并用电线连起来。

#### 1. 端口定义 (Port Definition)

* **输入**：
  + clk, rst\_n：系统时钟和复位。
  + rx\_line：电脑发过来的串口线（原始数据）。
  + sw\_encrypt：一个开关（1=加密，0=直通）。
* **输出**：
  + tx\_line：发回给电脑的串口线（处理后的数据）。

#### 2. 内部连线 (Internal Wires)

你需要在模块内部铺几根“导线” (wire)，用来连接各个子模块：

* rx\_data (8位)：用来接 UART 接收到的数据。
* rx\_done (1位)：UART 告诉我们“收好了”的信号。
* key\_byte (8位)：LFSR 产生的随机密钥。
* cipher\_data (8位)：你算出加密后的结果。
* tx\_data\_final (8位)：最终决定送给发送端的数据（是加密的还是原始的？）。

#### 3. 参数定义 (Parameters)

* 为了让子模块知道波特率，你需要定义两个参数（并在实例化时传给它们）：
  + CLK\_FREQ = 50,000,000 (50MHz)
  + BAUD\_RATE = 115200

#### 4. 模块实例化 (Instantiation) —— 最难的一步

你需要把三个模块“插”在主板上。

* **插槽 1 (UART RX)**：
  + 引用 uart\_rx 模块。
  + 把它的 clk, rst\_n 接到顶层的 clk, rst\_n。
  + 把它的 rx 接到 rx\_line。
  + 把它的输出 data\_out 接到你的内部线 rx\_data。
  + 把它的输出 done 接到你的内部线 rx\_done。
* **插槽 2 (LFSR)**：
  + 引用你刚才写的 lfsr 模块。
  + 连好时钟和复位。
  + 把它的输出 data\_out 接到你的内部线 key\_byte。
* **插槽 3 (UART TX)**：
  + 引用 uart\_tx 模块。
  + 连好时钟和复位。
  + **关键**：它的启动信号 tx\_en 应该连谁？（连 rx\_done，因为收到了才能发）。
  + **关键**：它的数据输入 data\_in 应该连谁？（连 tx\_data\_final，即最终决定的数据）。
  + 它的输出 tx 接到顶层引脚 tx\_line。

#### 5. 胶水逻辑 (Glue Logic) —— 你要亲手写的部分

* **加密逻辑**：写一行 assign，计算 cipher\_data = rx\_data 异或 key\_byte。
* **选择逻辑 (MUX)**：写一行 assign，计算 tx\_data\_final。
  + 如果 sw\_encrypt 是 1，就选 cipher\_data。
  + 否则，选 rx\_data。

### 第三部分：虚拟测试台 tb\_top.v (仿真脚本)

**设计目标**：假装自己是电脑，给 FPGA 发信号，检查它对不对。

#### 1. 信号定义

* 定义 reg 类型的信号：clk, rst\_n, rx\_line, sw\_encrypt（因为我们要主动控制它们）。
* 定义 wire 类型的信号：tx\_line（因为我们要观察它）。

#### 2. 被测模块连接 (DUT Instantiation)

* 把你的 top 模块实例化进来，把上面的信号一一对应连好。

#### 3. 时钟生成

* 写一个 initial 块，让 clk 永远每隔 10ns 翻转一次（产生 50MHz 时钟）。

#### 4. 模拟发送任务 (Task Definition) —— 难点

* 为了偷懒，你需要定义一个 task 叫 send\_byte。
* 输入参数：[7:0] data。
* 逻辑：
  1. 把 rx\_line 拉低（起始位）。
  2. 等待一个波特率周期 (8680ns)。
  3. 用一个 for 循环 8 次，每次把 data 的第 i 位发给 rx\_line，并等待一个周期。
  4. 把 rx\_line 拉高（停止位），并等待一个周期。

#### 5. 主剧本 (Test Scenario)

* initial begin
  + **初始化**：把所有信号设为初始状态（复位有效，时钟0，串口高电平）。
  + **复位释放**：过 100ns，把 rst\_n 变 1。
  + **场景 A**：把 sw\_encrypt 设为 1。调用 send\_byte(8'h55)。等待 2ms 让它发完。
  + **场景 B**：把 sw\_encrypt 设为 0。调用 send\_byte(8'h55)。等待 2ms。
  + **结束**：$stop。
* end

执行命令：

请按照 文件 1 -> 文件 2 -> 文件 3 的顺序，一个一个去研究、手写。

现在，请开始写第一个文件 lfsr.v。 写好后贴给我，我会像编译器一样帮你检查语法和逻辑。不要怕错，开始吧。