

AC709 开发板

CM3452_1250MHz_JESD204B 采集与 双 DDR 缓存和 TCP 传输说明手册

武汉芯路恒科技有限公司
教学产品项目组 编写

修订说明：

V1.0.0	2026-11-6	第一版本发行，cm3452 和 cm3453 共用一套驱动

版权所有 © 2026 武汉芯路恒科技有限公司

、小梅哥均为武汉芯路恒科技有限公司注册商标， 本手册中提到的其他任何商标，其所有权利属其拥有者所有。未经本公司书面许可，任何单位和个人都不得擅自摘抄、复制、翻译本文档内容的部分或全部，不得以任何形式传播。

免责声明

本文档并未授予任何知识产权的许可，并未以明示或暗示，或以禁止发言或其它方式授予任何知识产权许可。除芯路恒在其产品的销售条款和条件中声明的责任之外，武汉芯路恒概不承担任何法律或非法律责任。武汉芯路恒对芯路恒产品的销售和／或使用不作任何明示或暗示的担保，包括对产品的特定用途适用性、适销性或对任何专利权、版权或其它知识产权的侵权责任等，均不作担保。芯路恒对文档中包含的文字、图片及其它内容的准确性和完整性不承担任何法律或非法律责任，芯路恒保留修改文档中任何内容的权利，恕不另行通知。芯路恒不承诺对这些文档进行适时的更新。

联系我们

厂家名称：武汉芯路恒科技有限公司
联系地址：武汉东湖新技术开发区大学园路长城园路 8 号光谷精工科技园 A 栋 301-5 室
联系电话： 027-59265620
技术网站：https://fpga.cn/
微信公众号（提供在线客服）：小梅哥电子
在线商城（淘宝）：https://xiaomeige.taobao.com
在线商城（天猫）：https://xiaomeigesm.tmall.com/

目录

1. ACFH3452_1250MHz_JESD204B 采集与双 DDR 缓存和 TCP 传输设计说明手册 3
1.1 ACFH3452 模块简介 3
1.2 工程目标： 5
1.3 工程目录结构 6
1.4 整体数据流 7
1.5 时钟说明与数据格式 7
1.5.1 时钟参数及吞吐预算 7
1.5.2 lane 到通道的重排 9
1.6 上电初始化与 JESD 链路 10
1.6.1 供电时序与自动启动 10
1.6.2 AD9528 与 ADC 初始化表 11
1.6.3 PHY 和 RX 配置流程 15
1.7 自定义 RTL 模块逐项解析 16
1.7.1 top_tcp 顶层连接 16
1.7.2 cm3453_tcp_frontend 前端封装 17
1.7.3 ad9528_cm3453_init_rom 初始化 ROM 19
1.7.4 spi_3wire_dual_csb 三线 SPI 引擎 20
1.7.5 cm3453_spi_ctrl 自动初始化与手动访问 21
1.7.6 cm3453_phy_config 均衡配置 22
1.7.7 cm3453_jesd_transport 样点重排 23
1.7.8 cm3453_dual_capture 同步采集与分流 24
1.7.9 cm3453_axi_writer 突发写入器 26
1.7.10 cm3453_dual_capture_bd 接口适配 28
1.7.11 cm3453_capture_monitor 故障留痕 31
1.8 Block Design 与约束文件 32
1.8.1 PS 和 PL 的连接关系 32
1.8.2 AXI 控制 IP 两层模块 32
1.8.3 HPC1 与 CDC 约束 33
1.9 JESD PHY 与 RX IP 配置 34
1.9.1 JESD204 PHY 34

1.9.2 JESD204 RX.....34
1.10 SDK 应用程序逐项解析.....35
1.10.1 main 与平台初始化.....35
1.10.2 echo 网络连接与 TCP 组帧.....36
1.10.3 CMD_Handle 命令服务.....37
1.10.4 Restart_ADC_Sample 采集准备.....38
1.10.5 Data_Recive_Trans 缓存回读与发送.....39
1.10.6 ADC_Info 与链接脚本.....40
1.10.7 驱动和启动支持代码.....41
1.11 AXI 寄存器与状态字.....42
1.12 TCP 通信协议与常用命令.....43
1.12.1 请求和响应格式.....43
1.12.2 常用命令速查.....45
1.12.3 错误码和连接恢复.....46
1.13 硬件准备与实验平台连接.....47
1.14 上电工作流程与最小自检.....48
1.14.1 串口打印.....48
1.14.2 网络调试助手调试与测试.....49
1.15 功能验证.....51
1.16 思考与总结.....52

1. ACFH3452_1250MHz_JESD204B 采集与双 DDR 缓存和 TCP 传输设计说明手册

工程源码	|-- AC709 开发板 |-- AC709_CM3452_1250mhz_TCP_DUAL_DDR
相关视频课程	
参考官方文档	

章节导读

本工程基于 AC709 载板、ACZ7035 核心板和 CM3452 双通道高速 ADC 模块，实现双通道 1.25 GSPS、14 bit 数据采集。FPGA 通过 JESD204B 接收 ADC 数据，完成样点重排和符号扩展，将两个通道的数据分别缓存到 PS DDR3 和 PL DDR3，再由 Zynq PS 通过千兆以太网 TCP 接口回传至上位机。

1.1 ACFH3452 模块简介

CM3452 芯片是北京士模微电子有限责任公司研发的一款双通道 14 位高速 ADC，最高采样率为每通道 1.25 GSPS。

ACFH3452 模块提供两路 SMA 模拟输入，每通道采用两只巴伦组成平衡网络，将单端交流信号转换为差分信号，再经耦合电容和阻抗匹配电路送入 ADC。

模块采用 AD9528 与 100 MHz 差分振荡器构建时钟系统，通过 SPI 配置器件，并经 FMC HPC 接口将 JESD204B 高速串行数据送入 FPGA。

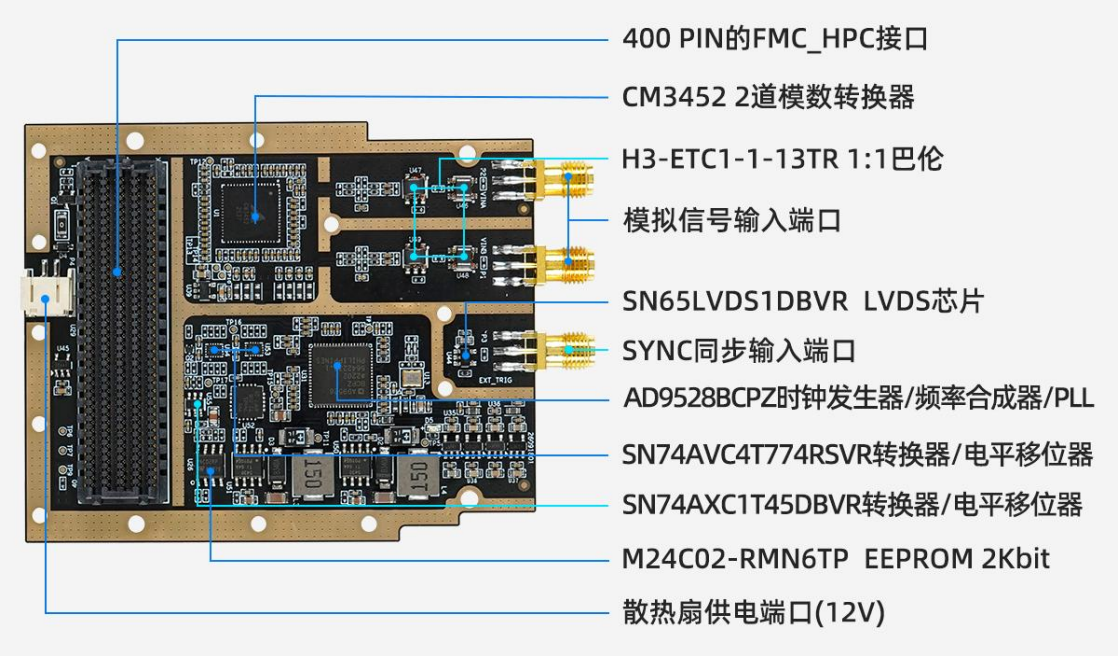


图 1-1 ACFF3452 模块实物图

ACFH3452 模块与 FPGA 的连接采用 FMC_HPC 接口，该模块接口图如下图所示。

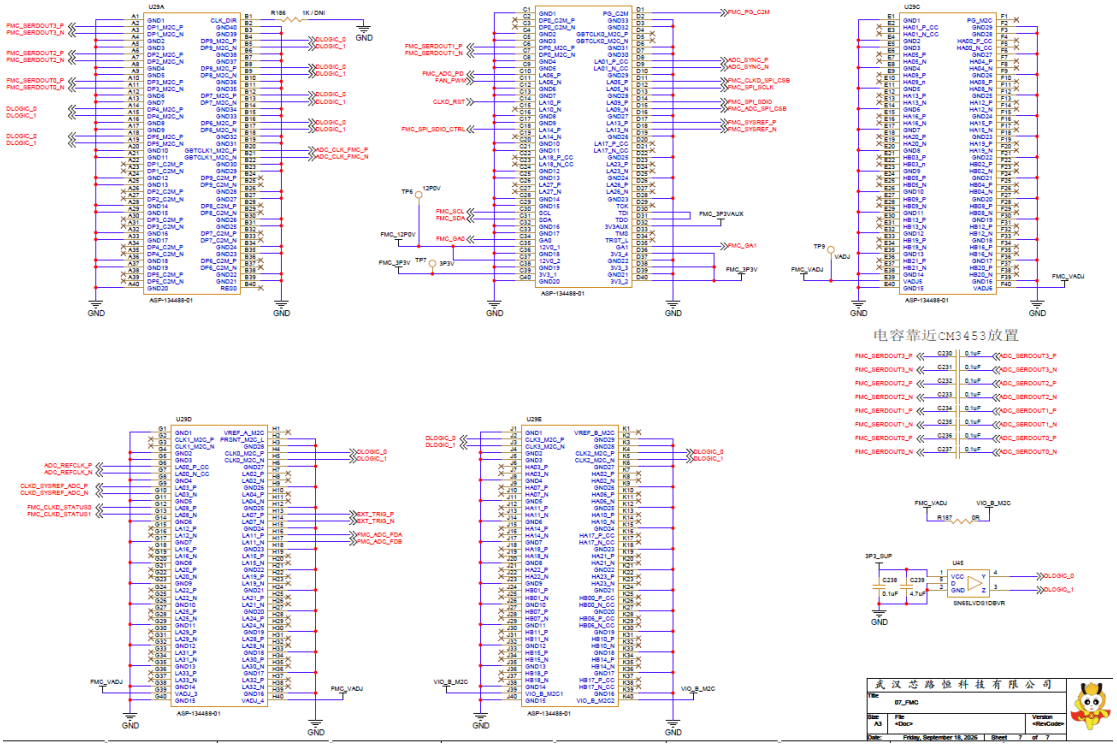


图 1-2ACFH3452 模块 FMC_HPC 接口原理图

模块采用 AD9528 与 100 MHz 差分振荡器构建时钟系统，使用该模块时，首先需要通过 SPI 协议来配置模块上的 ADC 芯片 CM3452 和时钟芯片 AD9528 相关寄存器，比如配置时钟芯片的时钟分频倍频等。

寄存器配置完成后，CM3452 芯片会根据 AD9528 提供的参考工作时钟（最高 1250MHz）进行数据采集转换，并通过 FMC HPC 接口将 JESD204B 高速串行数据送入 FPGA。

CM3452 默认差分满量程为 1.4 Vpp，默认采用二进制补码时，数据范围为 −8192～8191，每码约对应 85.45 μV。该换算针对 ADC 差分引脚，SMA 输入电压还需考虑前端传输损耗。

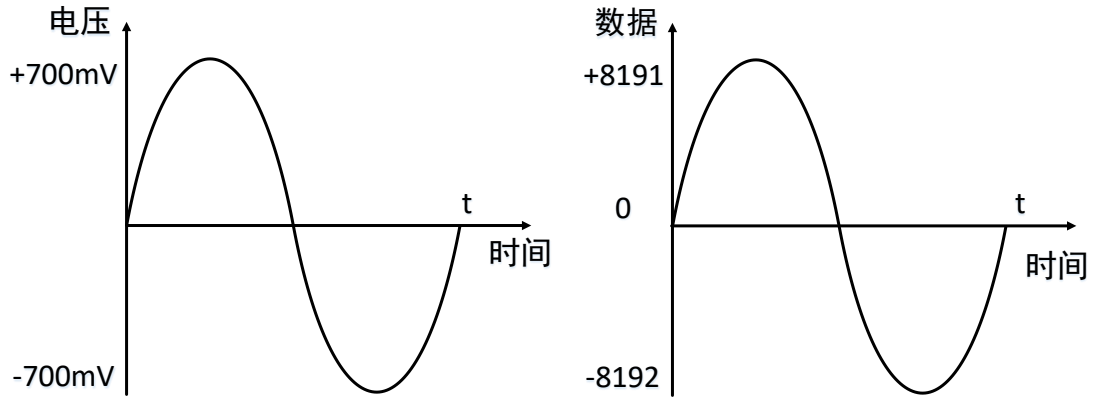


图 1-3 CM3452 正弦波模拟电压值（左）、数据（右）

1.2 工程目标：

1. 完成 AD9528 与 CM3452 的自动初始化。FPGA 上电后执行 FMC 电源稳定等待，通过约 5 MHz 三线 SPI 按预设寄存器表配置 AD9528 和 CM3452，并读取器件 ID、AD9528 PLL2 状态及 CM3452 JESD PLL 状态，为高速采集建立时钟和器件工作条件。
2. 建立匹配 1.25 GSPS 采样率的时钟链路。利用模块上的 100 MHz VCXO，经 AD9528 PLL2 产生 3.75 GHz VCO 时钟，通过 M1 分频形成 1.25 GHz 分配时钟；向 ADC 提供 1250 MHz 采样时钟，向 FPGA GTX 提供 125 MHz MGT 参考时钟。
3. 实现四路 JESD204B 高速串行数据接收。采用 M=2、L=4、F=1、K=32、N=16、N′ =16、Subclass 0、关闭扰码的链路配置，通过 GTXE2、QPLL、JESD204 PHY 和 JESD204 RX IP 接收 4×12.5 Gbps 串行数据，完成接收复位、链路同步与多 lane 数据对齐。其中 ADC 有效分辨率为 14 bit，工程按 16 bit 传输容器处理样点。
4. 完成 JESD 数据到双通道连续样点的重排。在约 312.5 MHz 的 rx_core_clk 时钟域内，将 JESD RX 输出的 128 bit 数据重排为两个通道，每通道每拍包含 4 个连续样点。从各 16 bit 容器中提取 [15:2] 的 14 bit 有效数据，并进行二进制补码符号扩展，形成统一的 16 bit 有符号样点。
5. 实现双通道同步启动、无抽取的双 DDR 数据捕获。两个通道在同一次捕获过程中同时保存数据：第一通道通过 PS 的 HP0～HP3 四个 64 bit AXI 接口写入 PS DDR3，第二通道通过 256 bit AXI 接口写入 PL MIG DDR3。利用异步 FIFO 完成跨时钟域缓存，通过 AXI burst 写入和写响

应检查确认数据已完成存储。

6. 实现上位机控制与 TCP 样点回传。Zynq PS 运行 standalone 与 lwIP 应用，使用固定地址 192.168.0.2、TCP 端口 5000 接收上位机命令。支持配置采集长度、选择回传通道、启动采集、访问 AD9528/CM3452 寄存器及读取状态；捕获完成后，从 DDR 分片读取并发送数据，双通道模式按 通道 A 样点、通道 B 样点 交错回传。

7. 建立完整的状态监测与异常恢复机制。监测时钟锁定、RX 时钟活动、JESD 数据有效、帧错误、GTX 编码错误、DDR 校准和捕获完成状态；记录捕获期间首次链路故障的阶段及原因，并处理存储溢出、操作超时和 TCP 断线，为实验调试与故障恢复提供依据。

8. 支持从数字测试码到实际模拟信号的分层验证。通过恒定满量程、棋盘格和软件递增数据等测试，分别检查样点位映射、符号扩展、通道顺序、双 DDR 存储及 TCP 数据重组；验证通过后恢复正常 ADC 数据模式，进行实际模拟信号采集。

1.3 工程目录结构

表 1-1 工程目录结构

文件/IP	主要作用
top_tcp.v	连接 AC709 HPC1、ADC 参考时钟、4 路 GTX、SPI、同步信号、PS DDR 与 PL DDR。fan_pwm 固定为 1，保证采集和空闲期间风扇满功率。
cm3453_tcp_frontend.v	协调 50 MHz 控制域、100 MHz JESD AXI-Lite 域和恢复的 RX 时钟域；汇总状态并输出样点帧。
cm3453_jesd_transport.v 、cm3453_jesd204_rx 、cm3453_jesd_phy	JESD 数据路径 完成四 lane 接收、AXI-Stream 解包、lane 到通道映射和 14 bit 样点恢复。
cm3453_spi_ctrl.v 、spi_3wire_dual_csb.v 、ad9528_cm3453_init_rom.v	上电初始化 AD9528/CM3452，提供手动读写和重训触发，SPI 约 5 MHz。
cm3453_dual_capture.v/_bd.v 、cm3453_axi_writer.v 、cm3453_capture_monitor.v	异步 FIFO 跨域，CH1 走 PS HP0..HP3，CH2 走 PL MIG；记录首个故障阶段和 sticky 原因。
main.c 、 echo.c 、	固定网络配置，解析 8 字节命令帧，启动捕获、管理 DCache、

CMD_Handle.c、ADC_Info.h	以 1460 字节分片回传样点。

1.4 整体数据流

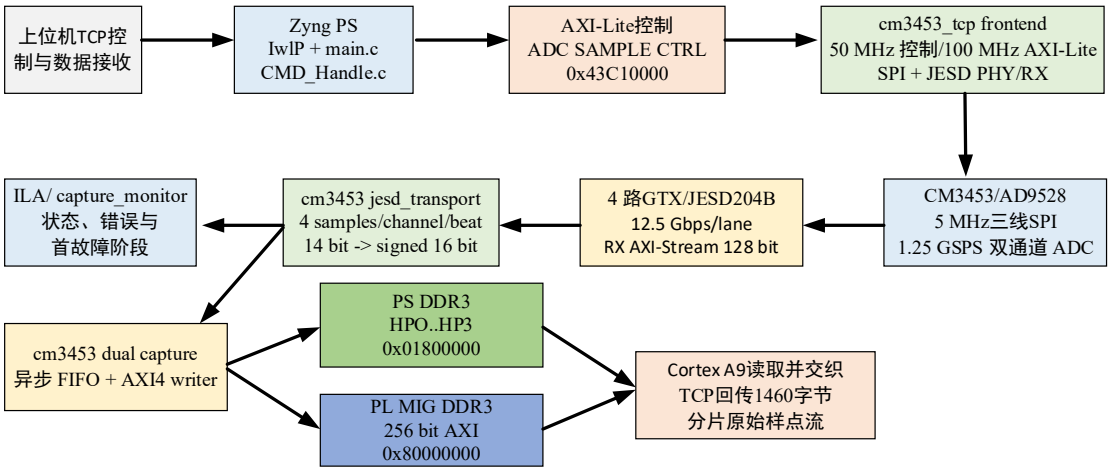


图 1-4 双通道采集与 TCP 回传数据流

1. PC 通过 网口下发寄存器、状态和采集命令；
2. 控制模块通过 5 MHz 三线 SPI 配置 AD9528 与 CM3452。
3. ADC 的 4 条 JESD lane 进入 GTX PHY，
4. 经 JESD204 RX 解码后形成 128-bit AXI4-Stream 数据，
5. cm3453_jesd_transport 将 14bit 原始数据拼接为 16 位有符号数据。
6. cm3453_dual_capture 将通道 A 写入 PS 侧 DDR，将通道 B 写入 PL 侧 DDR。
7. PS 在捕获完成后才开始 TCP 回传。

1.5 时钟说明与数据格式

1.5.1 时钟参数及吞吐预算

时钟链路，100 MHz VCXO 经 AD9528 产生 ADC 采样时钟与 GTX 参考时钟；GTXCDR 生成 rX_core_clk。

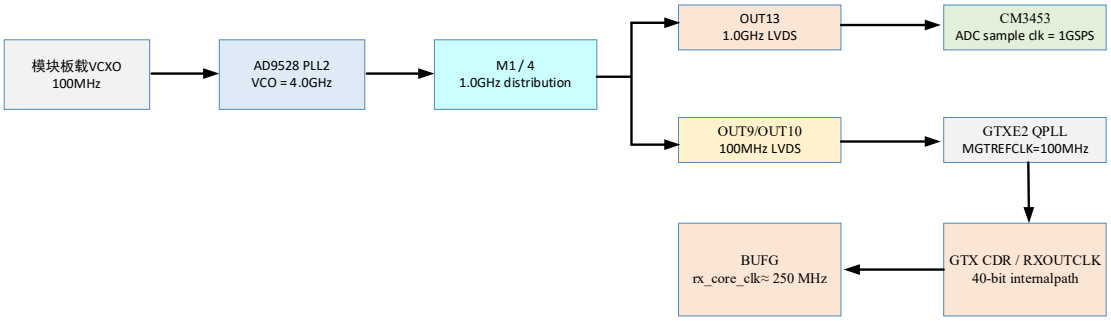


图 1-5 AD9528、ADC 与 GTX 时钟流向

表 1-2 时钟关系

信号/节点	频率	来源/去向
模块板载差分晶振 VCXO	100 MHz	进入 AD9528 PLL2
AD9528 PLL2 VCO	3.75 GHz	AD9528 PLL2；M1=/3，distribution=1.25 GHz
ADC 采样	1250 MHz	AD9528 OUT13；每样点 0.8 ns
MGT refclk	125 MHz	OUT9/OUT10，/10；XDC period=8 ns
JESD lane	12.5 Gbps	4 lane serial output
RX user clock	约 312.5 MHz	GT RXOUTCLK/BUFG
控制时钟	50 MHz	板载 F15 输入及 BD clk_wiz 输出；SPI 和启动 FSM
JESD AXI-Lite	100 MHz 配置参数	BD clk_wiz 第二输出
MIG 参考	200 MHz	BD clk_wiz 第三输出至 clk_ref_i
PS AXI 控制	100 MHz	PS FCLK_CLK0
HP 写入	125 MHz	PS FCLK_CLK1
MIG UI	125 MHz	500 MHz DDR 时钟、4 比 1 PHY 比例推得
PL MIG 系统输入	100 MHz 差分	pl_sys_clk_clk_p/n；独立于 F15 控制时钟

JESD lane rate 由采样率、converter 数、JESD 容器位宽和 lane 数共同决定：

$$\begin{aligned} \text{lane_rate} &= Fs \times M \times N' \times (10/8) / L \\ &= 1.25\times10^{9} \times 2 \times 16 \times 10 / 8 / 4 \\ &= 12.5\ \text{Gbps} \end{aligned}$$

$$\text{rx_core_clk} \approx 12.5\ \text{Gbps} / 40\ \text{bit} = 312.5\ \text{MHz}$$

有效数据 $4 \times 32 = 128\text{bit}$

这里的 40 bit 是 GTX 8b/10b 解码后的内部并行宽度。不能把 32 bit/lane 的用户接口宽度直接当作串行 lane rate 的分母。

每通道存储速率为 $1.25\times10^{9} \times 2 = 2.5$ GB/s，双通道合计 5 GB/s。四个 64 位 125MHz HP 端口的接口理论合计 4 GB/s，MIG 256 位 125MHz 接口理论 4 GB/s；这些是接口上限，实际 DDR 效率、仲裁、刷新和 CPU 访问会消耗余量，必须通过溢出状态和连续采集实验验收。

MIG 工程参数为 TimePeriod = 2000 ps、32 位 DDR3、PHY Ratio = 4:1，内存器件配置为 MT41K256M16XX-125。不能把器件型号后缀直接当成本工程 DDR 工作速率。千兆以太网物理上限仅 125 MB/s 且还有协议开销，因此本工程采用先存储后发送。

1.5.2 lane 到通道的重排

表 1-3 每拍样点位域

sample_frames	位域内容	时间顺序
15:0 / 31:16 / 47:32 / 63:48	CH1 sample0 / 1 / 2 / 3	由低位向高位递增
79:64 / 95:80 / 111:96 / 127:112	CH2 sample0 / 1 / 2 / 3	与 CH1 对应样点同步
每个 16 位单元	raw[15:2]取 14 位，再复制符号位到 15:14	有符号范围−8192 至 8191

相邻 lane 配对及 14 位符号扩展，lane0 与 lane1 构成 CH1，lane2 与 lane3 构成 CH2。原始 16 位容器的低 2 位不进入样点数值，因而 PC 不能对已符号扩展的回传数据再次右移 2 位。最低 16 位始终是当前拍该通道最早的 sample0。

cm3453_jesd_transport .v。transport 中的 sample_i、sample_q、sample_ch0、sample_ch1 只显示每拍第一个样点，不能用于替代 sample_frames 做满速采集。实际 DDR 路径使用完整 128 位总线；sample_valid 随 rx_tvalid 寄存一拍，数据和有效标志保持对应。

```
// M=2, L=4, F=1, N'=16: each frame carries one byte per lane.
// Observed ramp mode shows converter samples are adjacent lane pairs:
// channel 0 = {gt0, gt1}, channel 1 = {gt2, gt3}.
wire [15:0] frame0_ch0_raw = {rx_tdata[7:0],    rx_tdata[39:32]};
wire [15:0] frame0_ch1_raw = {rx_tdata[71:64],  rx_tdata[103:96]};
wire [15:0] frame1_ch0_raw = {rx_tdata[15:8],   rx_tdata[47:40]};
wire [15:0] frame1_ch1_raw = {rx_tdata[79:72],  rx_tdata[111:104]};
wire [15:0] frame2_ch0_raw = {rx_tdata[23:16],  rx_tdata[55:48]};
wire [15:0] frame2_ch1_raw = {rx_tdata[87:80],  rx_tdata[119:112]};
wire [15:0] frame3_ch0_raw = {rx_tdata[31:24],  rx_tdata[63:56]};
wire [15:0] frame3_ch1_raw = {rx_tdata[95:88],  rx_tdata[127:120]};

wire [13:0] frame0_ch0 = frame0_ch0_raw[15:2];
wire [13:0] frame0_ch1 = frame0_ch1_raw[15:2];
wire [13:0] frame1_ch0 = frame1_ch0_raw[15:2];
wire [13:0] frame1_ch1 = frame1_ch1_raw[15:2];
wire [13:0] frame2_ch0 = frame2_ch0_raw[15:2];
wire [13:0] frame2_ch1 = frame2_ch1_raw[15:2];
wire [13:0] frame3_ch0 = frame3_ch0_raw[15:2];
```

```
  wire [13:0] frame3_ch1 = frame3_ch1_raw[15:2];

  wire [15:0] frame0_ch0_ext = {{2{frame0_ch0[13]}}, frame0_ch0};
  wire [15:0] frame0_ch1_ext = {{2{frame0_ch1[13]}}, frame0_ch1};
  wire [15:0] frame1_ch0_ext = {{2{frame1_ch0[13]}}, frame1_ch0};
  wire [15:0] frame1_ch1_ext = {{2{frame1_ch1[13]}}, frame1_ch1};
  wire [15:0] frame2_ch0_ext = {{2{frame2_ch0[13]}}, frame2_ch0};
  wire [15:0] frame2_ch1_ext = {{2{frame2_ch1[13]}}, frame2_ch1};
  wire [15:0] frame3_ch0_ext = {{2{frame3_ch0[13]}}, frame3_ch0};
  wire [15:0] frame3_ch1_ext = {{2{frame3_ch1[13]}}, frame3_ch1};
```

完整双通道四样点打包。

```
  always @(posedge clk) begin
    if (rst) begin
      sample_i <= 16'h0000;
      sample_q <= 16'h0000;
      sample_ch0 <= 14'h0000;
      sample_ch1 <= 14'h0000;
      sample_frames <= 128'h0;
      sample_valid <= 1'b0;
    end else begin
      sample_valid <= rx_tvalid;
      if (rx_tvalid) begin
        sample_i <= frame0_ch0_ext;
        sample_q <= frame0_ch1_ext;
        sample_ch0 <= frame0_ch0;
        sample_ch1 <= frame0_ch1;
        sample_frames <= {
                    frame3_ch1_ext,   frame2_ch1_ext,   frame1_ch1_ext,
frame0_ch1_ext,
         frame3_ch0_ext, frame2_ch0_ext, frame1_ch0_ext, frame0_ch0_ext
        };
      end
    end
  end
```


1.6 上电初始化与 JESD 链路

1.6.1 供电时序与自动启动

FPGA 配置结束后，前端将 PG_C2M_HPC 置 1，驱动模块相关电源与电平转换使能。前端先等待 1 秒，并在等待期间保持 ADC 掉电；随后释放 SPI 控制器。控制器自身再等待约 100 ms，执行 AD9528 软复位、67 条 ROM 配置、PLL 与链路检查。首次启动出现数秒等待应结合状态判断，不能立即认定 SPI 失效。

自动启动依次检查 AD9528 的 0x000C=0x56 和 0x000D=0x04、0x0508 的

(value & 0x82)==0x82、ADC 0x056F 的 bit7 锁定，然后触发 JESD 重训练并等待约 200 ms 检查。完整启动含首次最多 3 轮，每轮链路检查含首次最多 3 次；失败后仍返回空闲状态以接受诊断和重新初始化。

表 1-4 启动状态

boot_status	含义
00	复位或启动进行中
80	自动启动检查通过
E1/E2	AD9528 ID 低字节或高字节错误
E3	AD9528 PLL2 检查失败
E4	ADC JESD PLL 检查失败
E5	JESD 链路健康检查失败

1.6.2 AD9528 与 ADC 初始化表

表 1-5 AD9528 关键配置

寄存器	值	当前工程作用
0108	01	启用差分 VCXO 接收
0200 / 0201 / 0202	80 / D2 / 23	PLL2 及校准分频组合；校准计数 75
0203 / 0204	14 / 03	倍频及 R1 路径、M1 组合
031D / 0320	09 / 09	OUT9/10 按 10 分频，125 MHz
0329	00	OUT13 不再分频，1250 MHz
0501 / 0502 / 0503	FF / 01 / 80	输出掉电掩码；保留 9、10、13
000F	01	IO_UPDATE；按 ROM 顺序执行
0203	15	启动 VCO 校准后更新并等待 100 ms

ad9528_cm3453_init_rom.v 。

ROM 注释给出的时钟目标是 PLL2 VCO 3.75 GHz、M1 除 3 后得到 1.25 GHz 分配时钟。100 MHz VCXO 经倍频及 R1 除 4 形成 50 MHz PFD，N2 为 25。寄存器实际写入和相邻 IO_UPDATE、校准等待构成一个整体，不能只复制某一个分频值。

```
      // PLL2 = 3.75 GHz, M1 = /3, distribution input = 1.25 GHz.
      // With a 100 MHz VCXO input and doubler/R1=/4 path: N2 = /25.
      // VCO calibration divider must match N2 * M1 = 75.
      // Register 0x0201: A[7:6] = 3, B[5:0] = 18, so 4 * B + A = 75.
      // The board drives VCXO_IN/VCXO_IN# from the differential 100
MHz
      // SG3225VAN oscillator.  The AD9528 reset default selects a
```

```
        // single-ended VCXO receiver, so explicitly enable differential
mode.
        7'd0:  begin addr = 15'h0108; data = 8'h01; end
        7'd1:  begin addr = 15'h0500; data = 8'h10; end
        7'd2:  begin addr = 15'h0504; data = 8'hA0; end
        // Keep OUT9, OUT10 and OUT13 enabled. Power down OUT7 and OUT8
to reduce SYSREF spur coupling.
        7'd3:  begin addr = 15'h0501; data = 8'hFF; end
        7'd4:  begin addr = 15'h0502; data = 8'h01; end
        7'd5:  begin addr = 15'h0503; data = 8'h80; end
        7'd6:  begin addr = 15'h0504; data = 8'hBF; end
        7'd7:  begin addr = 15'h0200; data = 8'h80; end
        7'd8:  begin addr = 15'h0201; data = 8'hD2; end
        7'd9:  begin addr = 15'h0202; data = 8'h23; end
        // Enable the PLL2 doubler/R1-divider path so R1=/4 is actually
used.
        7'd10: begin addr = 15'h0203; data = 8'h14; end
        7'd11: begin addr = 15'h0204; data = 8'h03; end
        7'd12: begin addr = 15'h0205; data = 8'h1F; end
        7'd13: begin addr = 15'h0206; data = 8'h00; end
        7'd14: begin addr = 15'h0207; data = 8'h04; end
        7'd15: begin addr = 15'h0208; data = 8'h18; end
        7'd16: begin addr = 15'h0209; data = 8'h00; end
        // OUT9, OUT10 = 1.25 GHz / 10 = 125 MHz LVDS.
        // OUT8 divider remains 1.25 GHz / 32 = 39.0625 MHz, but the
channel is powered down above.
        // OUT13 = 1.25 GHz / 1 = 1250 MHz LVDS.
        7'd17: begin addr = 15'h0315; data = 8'h00; end
        7'd18: begin addr = 15'h0316; data = 8'h00; end
        7'd19: begin addr = 15'h0317; data = 8'h09; end
        7'd20: begin addr = 15'h0318; data = 8'h00; end
        7'd21: begin addr = 15'h0319; data = 8'h00; end
        7'd22: begin addr = 15'h031A; data = 8'h1F; end
        7'd23: begin addr = 15'h031B; data = 8'h00; end
        7'd24: begin addr = 15'h031C; data = 8'h00; end
        7'd25: begin addr = 15'h031D; data = 8'h09; end
        7'd26: begin addr = 15'h031E; data = 8'h00; end
        7'd27: begin addr = 15'h031F; data = 8'h00; end
        7'd28: begin addr = 15'h0320; data = 8'h09; end
        7'd29: begin addr = 15'h0327; data = 8'h00; end
        7'd30: begin addr = 15'h0328; data = 8'h00; end
        7'd31: begin addr = 15'h0329; data = 8'h00; end
        7'd32: begin addr = 15'h032B; data = 8'h00; end
        7'd33: begin addr = 15'h032C; data = 8'h00; end
        7'd34: begin addr = 15'h000F; data = 8'h01; end
        // Start VCO calibration with the R1-divider path still enabled.
        7'd35: begin addr = 15'h0203; data = 8'h15; end
```

```
      // PLL2 PFD is 50 MHz. Keep the conservative 100 ms calibration
wait.
      // Wait 100 ms before issuing SYNC so calibration has ample
margin.
      7'd36: begin addr = 15'h000F; data = 8'h01; delay_cycles =
32'd5000000; end
      7'd37: begin addr = 15'h032A; data = 8'h01; end
      7'd38: begin addr = 15'h032A; data = 8'h00; end
      7'd39: begin addr = 15'h000F; data = 8'h01; end
```

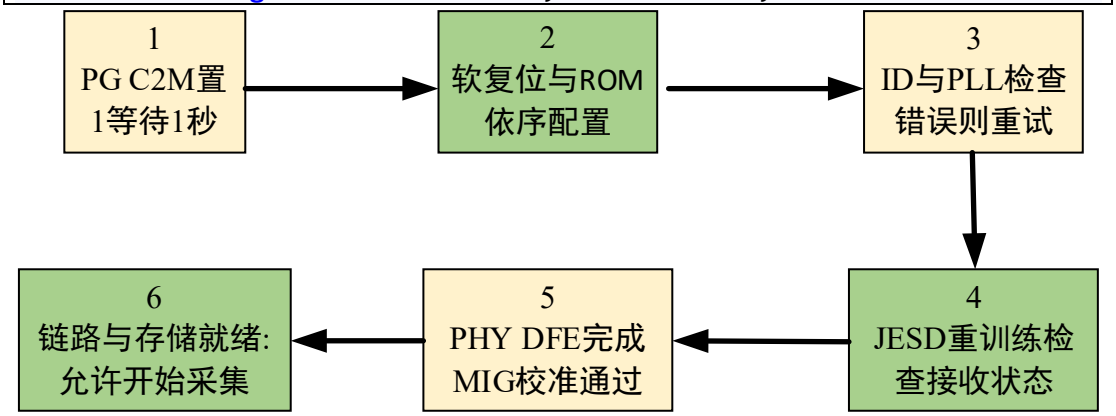


图 1-6 初始化和可采集条件

表 1-6 ADC 关键初始化

地址	值	作用
0000 / 0008	81 / 03	复位和初始工作设置
0016 / 0018	00 / 10	工程保留的模拟输入配置
0571 / 0572	14 / 20	FACI 相关配置及 SYNCINB 内部极性
058B / 058C / 058D	03 / 00 / 1F	4 lane、F=1、K=32 对应配置
058E / 058F / 0590	01 / 0F / 0F	M=2 及 16 位容器参数
0591 / 0592	20 / 80	按 ROM 配置的 JESD 设置
05BC 至 05BF	各 02	输出驱动配置
05C2 至 05C5	各 00	lane 相关参数
0550	00	当前有效代码默认正常模拟数据； 04 为棋盘测试

上电默认进入正常模拟数据模式。测试码须通过 TCP 命令显式切换。

```
// CM3453 JESD204B initialization for 1250 MHz sample clock:
// L=4, M=2, F=1, N=16, N'=16, lane rate = 12.5 Gbps.
7'd40: begin target_adc = 1'b1; addr = 15'h0000; data = 8'h81;
delay_cycles = 32'd5000000; end
7'd41: begin target_adc = 1'b1; addr = 15'h0008; data = 8'h03; end
```

```
// Preserve the verified analog input settings: 400 ohm input
termination and CM3452 harmonic optimization.
7'd42: begin target_adc = 1'b1; addr = 15'h0016; data = 8'h00; end
7'd43: begin target_adc = 1'b1; addr = 15'h0018; data = 8'h10; end
7'd44: begin target_adc = 1'b1; addr = 15'h0571; data = 8'h14; end
7'd45: begin target_adc = 1'b1; addr = 15'h056E; data = 8'h00; end
// Disable JESD scrambling; the FPGA lite receiver does not
descramble.
7'd46: begin target_adc = 1'b1; addr = 15'h058B; data = 8'h03; end
7'd47: begin target_adc = 1'b1; addr = 15'h058C; data = 8'h00; end
7'd48: begin target_adc = 1'b1; addr = 15'h058D; data = 8'h1F; end
7'd49: begin target_adc = 1'b1; addr = 15'h058E; data = 8'h01; end
7'd50: begin target_adc = 1'b1; addr = 15'h058F; data = 8'h0F; end
7'd51: begin target_adc = 1'b1; addr = 15'h0590; data = 8'h0F; end
7'd52: begin target_adc = 1'b1; addr = 15'h0591; data = 8'h20; end
7'd53: begin target_adc = 1'b1; addr = 15'h0592; data = 8'h80; end
// JESD output drive remains 483 mVpp; verify link margin at 12.5
Gbps.
7'd54: begin target_adc = 1'b1; addr = 15'h05BC; data = 8'h02; end
7'd55: begin target_adc = 1'b1; addr = 15'h05BD; data = 8'h02; end
7'd56: begin target_adc = 1'b1; addr = 15'h05BE; data = 8'h02; end
7'd57: begin target_adc = 1'b1; addr = 15'h05BF; data = 8'h02; end
7'd58: begin target_adc = 1'b1; addr = 15'h05C2; data = 8'h00; end
7'd59: begin target_adc = 1'b1; addr = 15'h05C3; data = 8'h00; end
7'd60: begin target_adc = 1'b1; addr = 15'h05C4; data = 8'h00; end
7'd61: begin target_adc = 1'b1; addr = 15'h05C5; data = 8'h00; end
7'd62: begin target_adc = 1'b1; addr = 15'h0001; data = 8'h02; end
7'd63: begin target_adc = 1'b1; addr = 15'h0001; data = 8'h00;
delay_cycles = 32'd5000000; end
// Enable FACI insertion so fixed test patterns retain explicit frame
alignment.
7'd64: begin target_adc = 1'b1; addr = 15'h0571; data = 8'h14;
delay_cycles = 32'd5000000; end
// Invert CM3453 SYNCINB input internally for the verified board
polarity.
7'd65: begin target_adc = 1'b1; addr = 15'h0572; data = 8'h20; end
// Power-up default is checkerboard test mode. Change data to 8'h00
for normal ADC samples.
7'd66: begin target_adc = 1'b1; addr = 15'h0550; data = 8'h00; end
//7'd66: begin target_adc = 1'b1; addr = 15'h0550; data = 8'h04; end
```

1.6.3 PHY 和 RX 配置流程

表 1-7JESD204B 参数与接收初始化

参数	说明
M	2 个 ADC converter
L	4 条 JESD lanes
F	每 frame 每 lane 1 octet
K	32 frames/multiframe
N	样点格式参数
N'	JESD sample container 为 16 bit
Subclass	不使用 SYSREF
Scrambling	CM3452 与 FPGA RX 均关闭
AXI-Stream	JESD RX 输出，每拍 4 lane × 4 octet
PHY	读回 0x608，检查 bit0=0
RX reset	frontend AXI-Lite 序列， retrain 后 rx_reset_done_rx=1
QPLL	125 MHz 参考；FBDIV=100，REFCLKDIV=1
RX 初始化写序列	020=0；024=1F；028=F；02C=0；004=1

cm3453_phy_config.v 在 100 MHz 域等待 receiver_ready，然后依次写 0x608=0、0x60C=1、0x60C=0，选择 DFE 并复位适应逻辑；读 0x608 验证 bit0 为 0。通过后拉高 receiver_reset 32 拍，等待 100000 拍约 1 ms，再进入 READY。每一事务阶段具有超时保护；WAIT_RX 阶段不提前消耗事务看门狗。

JESD RX 寄存器写入状态机分别等待 AW 和 W 握手，再等待 B 响应，不能假定地址和数据同拍被接受。该 RX 初始化状态机当前未把 B 响应错误与超时单独暴露给软件，链路健康与 phy_configured 共同决定 DATA_READY；调试时需要结合 ILA 观察事务。

差分 SYNC 由 OBUFDS 输出~rx_sync，ADC 0x0572 又写 0x20，两处极性应按当前板级连接整体保留。不要在 GT 串行 lane 上增加普通 SelectIO IDELAYE2；JESD 串行恢复、对齐和均衡由 GTX 及接收核完成。

```
    assign awaddr = (index == 0) ? 12'h608 : 12'h60C;
    assign wdata = (index == 1) ? 32'd1 : 32'd0;
    assign awvalid = (state == WRITE) && !address_done;
```

```
    assign wvalid = (state == WRITE) && !data_done;
    assign bready = (state == RESPONSE);
    assign araddr = 12'h608;
    assign arvalid = (state == READ_ADDR);
    assign rready = (state == READ_DATA);
    assign configured = (state == READY);
    assign failed = (state == FAILED);
    assign receiver_reset = (state == RESET_RX);
```

1.7 自定义 RTL 模块逐项解析

1.7.1 top_tcp 顶层连接

文件 top_tcp.v 声明的模块名是 top。外部端口包括 PS DDR 和 FIXED_IO、PL DDR3 总线、两组系统时钟、低有效复位、FMC SPI、MGT 参考、四路串行接收和差分 SYNC。fan_pwm 固定为 1，维持风扇全功率运行。

顶层实例 i_cm3453 产生 128 位 din_0、dvalid_0 及恢复时钟 M_AXIS_FIFO_CLK_0；design_1_wrapper 把它们连接到双 DDR 采集模块。控制寄存器、命令 toggle、状态字和回读字跨越两部分形成闭环。修改顶层后应同时检查 BD 端口、wrapper 和 XDC，不能仅让 Verilog 语法通过。

```
cm3453_tcp_frontend i_cm3453 (
  .clk50(clk_50M_0),.clk100(clk_100M_0),.mmcm_locked(mmcm_locked),
  .capture_debug(capture_debug),
  .rst_n(reset_n_0),
  .reg_conf(reg_conf_0),
  .reg_cmd_toggle(reg_cmd_toggle_0),
  .ad_sample_en(ad_sample_en_0),
  .channel_sel(ChannelSel_0),
  .speed_set(ADC_Speed_Set_0),
  .storage_status(storage_status_0),.capture_overflow(capture_overflow
_0),
  .cm_status(cm_status_0),
  .cm_readback(cm_readback_0),
  .ad_out(din_0),
  .ad_out_valid(dvalid_0),
  .adc1_clk(M_AXIS_FIFO_CLK_0),
  .PG_C2M_HPC(PG_C2M_HPC),
  .spi_sclk(spi_sclk),
  .spi_sdio(spi_sdio)
  ,.spi_sdio_dir(spi_sdio_dir),
  .ad9528_csb(ad9528_csb),
  .adc_csb(adc_csb),
  .adc_pd(adc_pd),
```

```
  .led_busy(led_busy),
  .led_error(led_error),
  .adc_refclk_p(adc_refclk_p),
  .adc_refclk_n(adc_refclk_n),
  .adc_rxp(adc_rxp),
  .adc_rxn(adc_rxn),
  .adc_sync_p(adc_sync_p),
  .adc_sync_n(adc_sync_n)
);
```

1.7.2 cm3453_tcp_frontend 前端封装

前端将供电、SPI、PHY、RX、transport 和采集监视器封装为一个模块。clk50 负责控制，clk100 负责 JESD AXI-Lite，rx_core_clk 负责样点。rst_n 与 mmcm_locked 共同形成内部复位，fmc_power_ready 再控制 SPI 释放。

PS 先写 32 位 reg_conf，控制 IP 翻转 reg_cmd_toggle。前端将 toggle 同步三拍，用相邻位异或检测事件；命令数据通过寄存器稳定传送。普通 SPI 命令完成时由 spi_ack_toggle 变化翻转 done_toggle；重新初始化则先观察 busy，再等 busy 结束。软件通过 done_toggle 相对旧值的变化确认本次完成，不能只等 BUSY=0。

reg_conf[31]优先表示重新初始化，[30]表示掉电控制，[25]选择 AD9528，[24]表示读，[22:8]为地址，[7:0]为数据。转成 spi_control 后，内部 target 位的含义变为 1 表示 ADC，两种编码不可混淆。

前端还用 RX 时钟分频心跳在 50MHz 域判断恢复时钟是否活动，50ms 没有变化会清除 rx_clk_alive。DATA_READY 要求 init_done、PHY 配置成功、RX 状态健康且编码错误为零，READY 则只表示命令接口空闲；READY 为 1 不能单独作为可采集依据。

命令事件处理和 SPI 命令编码：

```
// Only one request may be pending. The request/completion toggles
allow
// software to distinguish a completed operation from an idle
controller.
always @(posedge clk50) begin
    if (!ctrl_rst_n) begin
        command_sync       <= 0;
        command_meta       <= 0;
        command_word       <= 0;
        spi_control        <= 0;
        spi_execute_toggle <= 0;
        spi_reinit_toggle  <= 0;
```

```
        adc_pd_request     <= 0;
        command_pending    <= 0;
        ack_before         <= 0;
        reinit_busy_seen   <= 0;
        done_toggle        <= 0;
        command_failed     <= 0;
        last_read          <= 0;
    end else begin
        command_sync <= {command_sync[1:0], reg_cmd_toggle};
        command_meta <= reg_conf;
        command_word <= command_meta;

        if (command_event) begin
            if (spi_ctrl_busy || command_pending != 0) begin
                command_failed <= 1;
                done_toggle    <= ~done_toggle;
            end else if (command_word[31]) begin
                spi_reinit_toggle <= ~spi_reinit_toggle;
                command_pending   <= 2;
                reinit_busy_seen  <= 0;
                command_failed    <= 0;
            end else if (command_word[30]) begin
                adc_pd_request <= command_word[0];
                last_read      <= {7'd0, command_word[0]};
                command_failed <= 0;
                done_toggle    <= ~done_toggle;
            end else begin
                // Legacy READ bit24 and address[22:8] stay unchanged.
                // Bit25 selects AD9528; zero selects the ADC.
                spi_control <= {
                    7'd0,
                    !command_word[25],
                    command_word[24],
                    command_word[7:0],
                    command_word[22:8]
                };
                ack_before         <= spi_ack_toggle;
                spi_execute_toggle <= ~spi_execute_toggle;
                command_pending    <= 1;
                command_failed     <= 0;
            end
        end else if (command_pending == 1 && spi_ack_toggle !=
ack_before) begin
            last_read       <= spi_read_data;
            command_pending <= 0;
            done_toggle     <= ~done_toggle;
            command_failed  <= 0;
```

```
        end else if (command_pending == 2) begin
            if (spi_ctrl_busy) begin
                reinit_busy_seen <= 1;
            end else if (reinit_busy_seen) begin
                command_pending <= 0;
                done_toggle     <= ~done_toggle;
                command_failed  <= (boot_status[7:4] == 4'hE);
            end
        end
    end
end
```

1.7.3 ad9528_cm3453_init_rom 初始化 ROM

ROM 是组合查表模块，输入 7 位 index，输出 valid、目标芯片、15 位地址、8 位数据和 32 位延迟。索引 0 至 39 写 AD9528，40 至 66 写 ADC；越界返回 valid=0，由控制 FSM 结束配置并转入检查。

delay_cycles 使用控制时钟周期计数。当前 50MHz 时 5000000 拍对应 100ms。若只改变 CLK_HZ 而不同时检查 ROM 中的常数等待值，实际延时会改变。ROM 本身不发送 SPI、不判断锁定、不重试，所有执行顺序和异常处理由 cm3453_spi_ctrl 承担。

表 1-8 AD9528 与 CM3452 上电 SPI 详细序列

器件/地址	值	阶段	含义
AD9528 0x0108	0x01	输入配置	显式启用差分 VCXO 输入
AD9528 0x0200	0x80	PLL2	启用/选择 PLL2 路径
AD9528 0x0201	0xD2	PLL2	VCO calibration divider，匹配 N2×M1=75
AD9528 0x0202	0x23	PLL2	PLL2/M1 相关配置
AD9528 0x0203	0x14→0x15	校准	先配置 R1=/4，再启动 VCO calibration
AD9528 0x0204/0x0205	0x03/0x1F	PLL2	M1/N2 分频路径
AD9528 0x0208	0x18	PLL2	反馈分频修正
AD9528 0x031D/0x0320	0x09	输出	OUT9/OUT10 = 125 MHz MGT refclk
AD9528 0x0329	0x00	输出	OUT13 = 1250 MHz ADC clock
AD9528 0x000F	0x01	IO update	提交寄存器更新
CM3452 0x0000	0x81	复位	ADC soft reset
CM3452 0x0571	0x14	JESD/FACI	启用 FACI/帧对齐相关设置

CM3452 0x058B/0x058C	0x03/0x00	JESD	关闭 scrambling
CM3452 0x058D	0x1F	JESD	K-1=31，K=32
CM3452 0x058F/0x0590	0x0F/0x0F	JESD	4 lane 配置
CM3452 0x05BC..0x05BF	0x02	输出驱动	四 lane 驱动幅度设置
CM3452 0x05C2..0x05C5	0x00	输出驱动	保持默认输出配置
CM3452 0x0572	0x20	SYNC	按板级极性反相 SYNCINB
CM3452 0x0550	0x00	数据模式	正常 ADC 数据；测试时临时写 0x04

ROM 尾部与实际默认数据模式：

```
7'd61: begin target_adc = 1'b1; addr = 15'h05C5; data = 8'h00; end
7'd62: begin target_adc = 1'b1; addr = 15'h0001; data = 8'h02; end
7'd63: begin target_adc = 1'b1; addr = 15'h0001; data = 8'h00;
delay_cycles = 32'd5000000; end
// Enable FACI insertion so fixed test patterns retain explicit frame
alignment.
7'd64: begin target_adc = 1'b1; addr = 15'h0571; data = 8'h14;
delay_cycles = 32'd5000000; end
// Invert CM3453 SYNCINB input internally for the verified board
polarity.
7'd65: begin target_adc = 1'b1; addr = 15'h0572; data = 8'h20; end
// Power-up default is checkerboard test mode. Change data to 8'h00
for normal ADC samples.
7'd66: begin target_adc = 1'b1; addr = 15'h0550; data = 8'h00; end
// 7'd66: begin target_adc = 1'b1; addr = 15'h0550; data = 8'h04; end
default: valid = 1'b0;
```


1.7.4 spi_3wire_dual_csb 三线 SPI 引擎

SPI 引擎以 50MHz 时钟分频，HALF_DIV=5，因此 SCLK 为 5MHz。每次事务发送{rw,addr_word,wdata}共 24 位，最高位先发。ADC 地址只保留低 13 位，AD9528 使用 15 位地址；target_adc 选择两个低有效片选之一。

空闲 SCLK 为低，下降沿切换输出，上升沿采入读数据。读操作在 16 位指令结束后关闭 sdio_oe，同时 spi_sdio_dir 变低，使外部电平转换器和 FPGA 三态方向一致。若 ID 读回全 00 或全 FF，应同时检查片选、模块供电、方向脚和读阶段三态，而不只是检查 SCLK。

ST_IDLE 等待 start，ST_RUN 完成 48 个半周期，ST_DONE 输出单拍 done。

busy 与 done 不同：done 只标记一次事务结束；控制器必须锁存读回数据后才启动下一次访问。

读操作采样和 SDIO 方向切换：

```
ST_RUN: begin
    busy <= 1'b1;
    ad9528_csb <= active_adc ? 1'b1 : 1'b0;
    adc_csb <= active_adc ? 1'b0 : 1'b1;
    if (div_cnt == DIVIDER - 1) begin
    div_cnt <= 16'd0;
    spi_sclk <= ~spi_sclk;

    if (!spi_sclk) begin
        if (rw && bit_cnt >= 6'd16) begin
        rdata <= {rdata[6:0], sdio_i};
        end
    end else begin
        bit_cnt <= bit_cnt + 1'b1;
        if (bit_cnt == 6'd23) begin
        spi_sclk <= 1'b0;
        ad9528_csb <= 1'b1;
        adc_csb <= 1'b1;
        sdio_oe <= 1'b0;
        state <= ST_DONE;
        end else begin
        if (rw && bit_cnt >= 6'd15) begin
            sdio_oe <= 1'b0;
        end else begin
            sdio_oe <= 1'b1;
            sdio_o <= tx_shift[22 - bit_cnt];
        end
        end
    end
    end else begin
    div_cnt <= div_cnt + 1'b1;
    end
end
```

1.7.5 cm3453_spi_ctrl 自动初始化与手动访问

该模块将 SPI 引擎、初始化 ROM 和启动检查组合起来。ST_POWERON_WAIT、ST_BOOT_SPI_START/WAIT/DELAY 处理软复位与验证；ST_INIT_START/WAIT/DELAY 逐条执行 ROM；ST_IDLE 接受手动访问或重新初始化请求；ST_MANUAL_WAIT/DELAY 完成手动事务与链路更新等待。

boot_status 以 0x80 标记启动成功，以 E1 至 E5 保存失败阶段。led_busy 对

应 state 非 IDLE，led_error 只看 boot_status 高半字节是否为 E。因此日常采集 FIFO 溢出不一定点亮 led_error，应读完整软件状态。

当前 manual_adc_jesd_update_addr 包含 0x0550、0x0571 至 0x0574、0x05BC 至 0x05BF、0x05C2 至 0x05C5。向这些 ADC 地址写值后产生 adc_mode_update_pulse 并等待约 50ms。该延时不是模拟信号质量验收，切换后仍应轮询 DATA_READY 和 TRAIN_PASS。

健康条件和触发重训练的地址范围：

```
// ----------------------------------------------------------------------
// Synchronize receiver status into the controller clock domain.
// ----------------------------------------------------------------------
    (* ASYNC_REG = "TRUE" *) reg [7:0] status_flags0_meta = 8'h00;
    (* ASYNC_REG = "TRUE" *) reg [7:0] status_flags0_sync = 8'h00;
    (* ASYNC_REG = "TRUE" *) reg [7:0] status_flags2_meta = 8'h00;
    (* ASYNC_REG = "TRUE" *) reg [7:0] status_flags2_sync = 8'h00;

    wire boot_link_healthy =
            (status_flags0_sync[7:4] == 4'hF) && status_flags0_sync[0]
&&
            !status_flags0_sync[1] && (status_flags2_sync == 8'h00);

    wire manual_adc_jesd_update_addr =
            (spi_addr == 15'h0550) ||
            ((spi_addr >= 15'h0571) && (spi_addr <= 15'h0574)) ||
            ((spi_addr >= 15'h05BC) && (spi_addr <= 15'h05BF)) ||
            ((spi_addr >= 15'h05C2) && (spi_addr <= 15'h05C5));

    assign adc_pd = adc_pd_request;
    assign spi_busy = (state != ST_IDLE);
    assign led_busy = spi_busy;
    assign led_error = (boot_status[7:4] == 4'hE);
```

1.7.6 cm3453_phy_config 均衡配置

该模块是专用 AXI-Lite 主机，不是 JESD 样点数据模块。configured 只在 READY 状态为 1，failed 只在 FAILED 状态为 1，receiver_reset 只在 RESET_RX 状态为 1。当前前端没有把 failed 独立引出，而通过 configured 阻止 DATA_READY 置位。

地址和数据通道分别记录 address_done 和 data_done，保证在背压下 VALID 保持到对应 READY。写响应必须为 OKAY；读响应必须为 OKAY 且 RXLPMEN 读回为 0。任何一步失败进入 FAILED，恢复需要复位该配置器；仅重复 ADC SPI 初始化不一定清除此状态。

均衡读回检查和接收器重置：

```
READ_DATA: begin
    if (rvalid) begin
        state <= (rresp == 0 && !rdata[0]) ? RESET_RX : FAILED;
        count <= 0;
    end
end
RESET_RX: begin
    if (count == RESET_CYCLES - 1) begin
        state <= SETTLE;
        count <= 0;
    end
end
SETTLE: begin
    if (count == SETTLE_CYCLES - 1) begin
        state <= READY;
        count <= 0;
    end
end
```

1.7.7 cm3453_jesd_transport 样点重排

transport 输入是 4 lane 合并的 128 位 rx_tdata，每 lane 含 4 个 octet。它按同一 frame 内的相邻 lane 高低字节配对，提取容器[15:2]，得到 14 位有符号样点，再扩展为 16 位。该模块每拍并行处理 8 个样点，不进行降采样。

rst 有效时清空输出和 sample_valid；rx_tvalid 无效时保留上一拍数据但 sample_valid 为 0。因此调试软件不能看到重复的 sample_frames 就认定是 ADC 输出不变，必须同时观察 valid。双 DDR 采集中途 valid 丢失会被视为本帧失效，而不是暂停等下一拍补齐。

当前工程的 lane 到通道关系由 cm3453_jesd_transport.v 的实际位选决定，而不是由通道命名推测。每个 16 bit raw container 由两个相邻 lane 的 octet 拼接得到，之后取 [15:2] 作为 14 bit 有效数据，再将 bit13 复制到高两位，形成有符号 16 bit。

图 1-7 JESD lane 到 ADC channel 映射

对象	来源	当前定义
frame0_ch0_raw	{rx_tdata[7:0], rx_tdata[39:32]}	通道 A 第 0 个容器
frame0_ch1_raw	{rx_tdata[71:64], rx_tdata[103:96]}	通道 B 第 0 个容器
frame1/2/3	相同规律，按 8 bit 偏移	每拍各通道 4 个容器
sample_ch0/1	raw[15:2]	14 bit 原始码，供诊断/ILA

sample_frames[63:0]	frame3_ch0_ext...frame0_ch0_ext	通道 A，低 64 bit
sample_frames[127:64]	frame3_ch1_ext...frame0_ch1_ext	通道 B，高 64 bit
ad_out_valid	rx_tvalid 延迟一拍	有效样点帧指示

四帧原始容器重组：

```
  // M=2, L=4, F=1, N'=16: each frame carries one byte per lane.
  // Observed ramp mode shows converter samples are adjacent lane
pairs:
  // channel 0 = {gt0, gt1}, channel 1 = {gt2, gt3}.
  wire [15:0] frame0_ch0_raw = {rx_tdata[7:0],    rx_tdata[39:32]};
  wire [15:0] frame0_ch1_raw = {rx_tdata[71:64],  rx_tdata[103:96]};
  wire [15:0] frame1_ch0_raw = {rx_tdata[15:8],   rx_tdata[47:40]};
  wire [15:0] frame1_ch1_raw = {rx_tdata[79:72],  rx_tdata[111:104]};
  wire [15:0] frame2_ch0_raw = {rx_tdata[23:16],  rx_tdata[55:48]};
  wire [15:0] frame2_ch1_raw = {rx_tdata[87:80],  rx_tdata[119:112]};
  wire [15:0] frame3_ch0_raw = {rx_tdata[31:24],  rx_tdata[63:56]};
  wire [15:0] frame3_ch1_raw = {rx_tdata[95:88],  rx_tdata[127:120]};
```

1.7.8 cm3453_dual_capture 同步采集与分流

采集模块接收 channel_bytes 作为每通道字节数，要求非零且为 32 的整数倍，并在整个采集期间保持稳定。enable 分别同步到 RX、PS 和 PL 域。所有 5 个 writer 已 armed 且 FIFO 写侧复位忙信号均清除后，RX 域置 started，从下一次 take 起同时采两个通道。

RX 每拍接收每通道 8 字节。CH1 用 rx_word[5:4]选择 HP0 至 HP3，连续 16 拍即 128 字节归同一端口，再轮换下一个；四个端口分别写基址+0、+128、+256、+384，并以 512 字节为地址步长，重建连续 PS 缓冲区。

CH2 把 4 个 RX 拍的 64 位数据拼成 256 位，最早一拍在最低位，再写 MIG FIFO。pack_count 从 0 到 3，第四拍入 FIFO；每个 256 位 beat 包含 16 个样点。n 为 32 的整数倍保证最后没有未完成的拼包。

四个 HP FIFO 均为 4096×64 位异步 FWFT FIFO，每个 32KiB；MIG FIFO 为 4096×256 位，共 128KiB。FIFO 用于吸收时钟域和 DDR 短期仲裁差异，不能永久弥补平均吞吐不足。FIFO 满、writer 响应错误、采集中 sample_valid 掉低均会使帧失效。

共同启动条件与溢出检测：

```
always @(posedge rx_clk) begin
    rx_en      <= {rx_en[1:0], capture_enable};
    bytes_meta <= channel_bytes;
    bytes_sync <= bytes_meta;
    ready_meta <= writer_armed;
```

```
    ready_sync <= ready_meta;

    if (!rx_en[2]) begin
        rx_word    <= 0;
        words_remaining <= 0;
        accepting  <= 0;
        started    <= 0;
        overflow   <= 0;
        pack_count <= 0;
        pack_data  <= 0;
    end else begin
        // Both channels start on the same RX edge, only after every
        // writer is armed and every FIFO write side has left reset.
        if (!started && (&ready_sync) && !(|wr_busy)) begin
            started <= 1;
            words_remaining <= bytes_sync[31:3];
            accepting <= (bytes_sync[31:3] != 0);
        end

        if (take) begin
            if (fifo_full[owner] || (pack_count == 3 && fifo_full[4]))
begin
                overflow <= 1;
            end

            rx_word    <= rx_word + 1'b1;
            words_remaining <= words_remaining - 1'b1;
            if (words_remaining == 1) begin
                accepting <= 0;
            end
            pack_data  <= {sample_frames[127:64], pack_data[191:64]};
            pack_count <= pack_count + 1'b1;
        end

   // A missing sample invalidates the frame; the ADC cannot stall.
        if (accepting && !sample_valid) begin
            overflow <= 1;
        end
    end
end
```

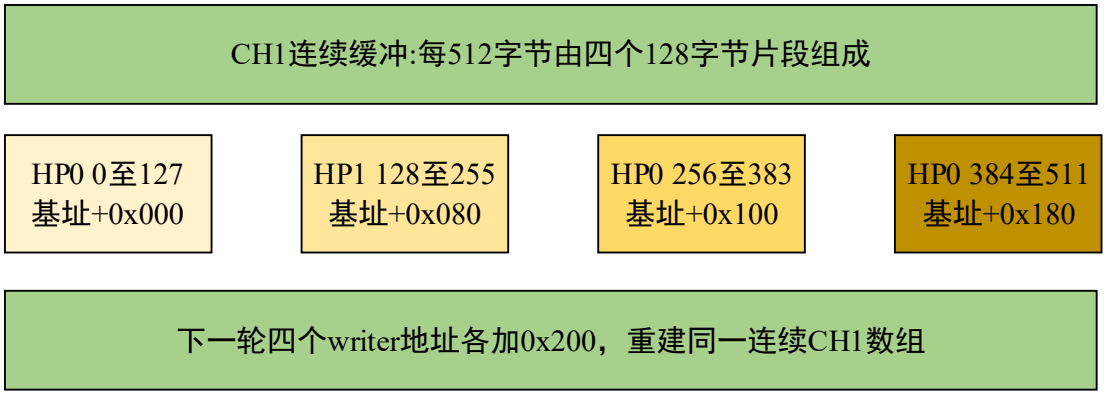


图 1-8 CH1 的 HP 分片与连续地址重建

表 1-9 CH1 的四路地址分配

端口	首地址	每完整 stipe 数据	下次 burst 地址增量
HP0	01800000	前 128 字节	512 字节
HP1	01800080	前 128 至 255 字节	512 字节
HP2	01800100	前 256 至 383 字节	512 字节
HP3	01800180	前 384 至 511 字节	512 字节
MIG	80000000	CH2 连续写，16x32=512 字节/burst	512 字节

长度不足 512 字节时，各 HP 的 target_words 根据完整 stripe 数量与尾部所属范围计算，没有分到数据的端口仍可完成零长度 writer。这使每通道 32 字节等短帧仍可由全 5 个 done 聚合判定。storage_status[7]表示仍有 writer 持有或排空事务，[6]表示全 5 个 done，[5]为 MIG 校准，[4:0]分别为 MIG、HP3、HP2、HP1、HP0 完成位。

HP0 尾部字数计算：

```
    // Every complete 512-byte stripe contributes 16 beats to this HP
port.
    // The final partial stripe contributes only the beats owned by
HP0.
    wire [31:0] tail0 = bytes_sync0 / 8 % 64;
    wire [31:0] words0 = (bytes_sync0 / 512) * 16
    + ((tail0 > 0) ? ((tail0 >= 16) ? 16 : tail0 - 0) : 0);
```

1.7.9 cm3453_axi_writer 突发写入器

writer 对 FWFT FIFO 执行 AXI4 INCR 写入，每 burst 最多 16 拍，最后 burst 可不足 16 拍。发 AW 前先检查 fifo_count 至少等于 next_words，减少已接受地址后因输入不足导致的 W 通道停顿。AWLEN 为拍数减 1。

W 通道同时只处理一个 burst，最多保留 8 个未返回 B 响应。AW 握手增加 outstanding，B 握手减少；同拍发生保持数量。非 OKAY 的 BRESP 锁存 error。WREADY 拉低时保持 WDATA、WLAST 和 WVALID 稳定；允许上一拍被接受时立即装入下一拍。

关闭 enable 不能撤销已经被接受的 AXI 事务。writer 进入 aborting 路径，完成已发出的 burst，必要时用 0 补齐，等待全部 B 响应后 idle。取消后的缓冲区内容不得作为有效采集结果。done 仅在 remaining 为 0、AW/W 空闲并且 outstanding 为 0 时置位，含义是 DDR 写响应全部返回，而非最后一个 FIFO 数据被取走。

突发发起及地址推进：

```
    // Reserve a full burst before issuing AW. The final burst may
    // contain fewer than 16 beats when target_words is not aligned.
    if (active && enable && !aborting
        && !awvalid && !writing && !wvalid
        && remaining != 0 && outstanding < 8
        && fifo_count >= next_words) begin
        awvalid     <= 1;
        awlen       <= next_words - 1'b1;
        burst_words <= next_words;
    end

    if (aw_fire) begin
        awvalid   <= 0;
        writing   <= 1;
        loaded    <= 0;
        remaining <= remaining - burst_words;
    end

    if (wvalid && wready) begin
        wvalid <= 0;

        if (wlast) begin
            writing <= 0;
            awaddr  <= awaddr + ADDR_STRIDE;
        end
    end
```

数据保持与最终完成判定：

```
// This block intentionally follows W consumption: a replacement
// beat keeps WVALID asserted, with WDATA/WLAST held if stalled.
          if (load_word) begin
              wdata  <= fifo_pop ? fifo_data : {DATA_WIDTH{1'b0}};
              wlast  <= (loaded == burst_words - 1'b1);
```

```
                wvalid <= 1;
                loaded <= loaded + 1'b1;
            end

    // A frame is complete only after every write response returns.
    // done stays high until enable is cleared by the caller.
    if (active && enable && !aborting && remaining == 0
          && !awvalid && !writing && !wvalid && outstanding == 0) begin
                done <= 1;
            end
        end
     end
```

1.7.10cm3453_dual_capture_bd 接口适配

该文件是 Block Design 使用的 module reference 包装，保留五组 AXI 写端口、PS/PL/RX 时钟、采样使能、每通道字节数和状态输出。内部实例 cm3453_dual_capture 执行实际采集，包装自身不增加缓存、不重排样点。

X_INTERFACE_INFO 和 X_INTERFACE_PARAMETER 帮助 Vivado 识别 64 位 HP 与 256 位 MIG 接口、独立时钟和复位。修改内核端口宽度或命名后，应同步修改包装并 Refresh Module，再 Validate Design 及生成 wrapper；否则 BD 缓存可能仍按旧接口连线。

BD 包装实例与采样控制接口：

```
    cm3453_dual_capture u_capture (
        .ps_clk (ps_clk),
        .ps_rst_n (ps_rst_n),
        .pl_clk (pl_clk),
        .pl_rst_n (pl_rst_n),
        .rx_clk (rx_clk),
        .capture_enable (capture_enable),
        .channel_bytes (channel_bytes),
        .sample_frames (sample_frames),
        .sample_valid (sample_valid),
        .calib_complete (calib_complete),
        .capture_error (capture_error),
        .storage_status (storage_status),
        .capture_debug (capture_debug),
        .m0_axi_awid (m0_axi_awid),
        .m0_axi_awaddr (m0_axi_awaddr),
        .m0_axi_awlen (m0_axi_awlen),
        .m0_axi_awsize (m0_axi_awsize),
        .m0_axi_awburst (m0_axi_awburst),
        .m0_axi_awlock (m0_axi_awlock),
        .m0_axi_awcache (m0_axi_awcache),
```

```
        .m0_axi_awprot (m0_axi_awprot),
        .m0_axi_awqos (m0_axi_awqos),
        .m0_axi_awvalid (m0_axi_awvalid),
        .m0_axi_awready (m0_axi_awready),
        .m0_axi_wdata (m0_axi_wdata),
        .m0_axi_wstrb (m0_axi_wstrb),
        .m0_axi_wlast (m0_axi_wlast),
        .m0_axi_wvalid (m0_axi_wvalid),
        .m0_axi_wready (m0_axi_wready),
        .m0_axi_bid (m0_axi_bid),
        .m0_axi_bresp (m0_axi_bresp),
        .m0_axi_bvalid (m0_axi_bvalid),
        .m0_axi_bready (m0_axi_bready),
        .m1_axi_awid (m1_axi_awid),
        .m1_axi_awaddr (m1_axi_awaddr),
        .m1_axi_awlen (m1_axi_awlen),
        .m1_axi_awsize (m1_axi_awsize),
        .m1_axi_awburst (m1_axi_awburst),
        .m1_axi_awlock (m1_axi_awlock),
        .m1_axi_awcache (m1_axi_awcache),
        .m1_axi_awprot (m1_axi_awprot),
        .m1_axi_awqos (m1_axi_awqos),
        .m1_axi_awvalid (m1_axi_awvalid),
        .m1_axi_awready (m1_axi_awready),
        .m1_axi_wdata (m1_axi_wdata),
        .m1_axi_wstrb (m1_axi_wstrb),
        .m1_axi_wlast (m1_axi_wlast),
        .m1_axi_wvalid (m1_axi_wvalid),
        .m1_axi_wready (m1_axi_wready),
        .m1_axi_bid (m1_axi_bid),
        .m1_axi_bresp (m1_axi_bresp),
        .m1_axi_bvalid (m1_axi_bvalid),
        .m1_axi_bready (m1_axi_bready),
        .m2_axi_awid (m2_axi_awid),
        .m2_axi_awaddr (m2_axi_awaddr),
        .m2_axi_awlen (m2_axi_awlen),
        .m2_axi_awsize (m2_axi_awsize),
        .m2_axi_awburst (m2_axi_awburst),
        .m2_axi_awlock (m2_axi_awlock),
        .m2_axi_awcache (m2_axi_awcache),
        .m2_axi_awprot (m2_axi_awprot),
        .m2_axi_awqos (m2_axi_awqos),
        .m2_axi_awvalid (m2_axi_awvalid),
        .m2_axi_awready (m2_axi_awready),
        .m2_axi_wdata (m2_axi_wdata),
        .m2_axi_wstrb (m2_axi_wstrb),
        .m2_axi_wlast (m2_axi_wlast),
```

```
        .m2_axi_wvalid (m2_axi_wvalid),
        .m2_axi_wready (m2_axi_wready),
        .m2_axi_bid (m2_axi_bid),
        .m2_axi_bresp (m2_axi_bresp),
        .m2_axi_bvalid (m2_axi_bvalid),
        .m2_axi_bready (m2_axi_bready),
        .m3_axi_awid (m3_axi_awid),
        .m3_axi_awaddr (m3_axi_awaddr),
        .m3_axi_awlen (m3_axi_awlen),
        .m3_axi_awsize (m3_axi_awsize),
        .m3_axi_awburst (m3_axi_awburst),
        .m3_axi_awlock (m3_axi_awlock),
        .m3_axi_awcache (m3_axi_awcache),
        .m3_axi_awprot (m3_axi_awprot),
        .m3_axi_awqos (m3_axi_awqos),
        .m3_axi_awvalid (m3_axi_awvalid),
        .m3_axi_awready (m3_axi_awready),
        .m3_axi_wdata (m3_axi_wdata),
        .m3_axi_wstrb (m3_axi_wstrb),
        .m3_axi_wlast (m3_axi_wlast),
        .m3_axi_wvalid (m3_axi_wvalid),
        .m3_axi_wready (m3_axi_wready),
        .m3_axi_bid (m3_axi_bid),
        .m3_axi_bresp (m3_axi_bresp),
        .m3_axi_bvalid (m3_axi_bvalid),
        .m3_axi_bready (m3_axi_bready),
        .m4_axi_awid (m4_axi_awid),
        .m4_axi_awaddr (m4_axi_awaddr),
        .m4_axi_awlen (m4_axi_awlen),
        .m4_axi_awsize (m4_axi_awsize),
        .m4_axi_awburst (m4_axi_awburst),
        .m4_axi_awlock (m4_axi_awlock),
        .m4_axi_awcache (m4_axi_awcache),
        .m4_axi_awprot (m4_axi_awprot),
        .m4_axi_awqos (m4_axi_awqos),
        .m4_axi_awvalid (m4_axi_awvalid),
        .m4_axi_awready (m4_axi_awready),
        .m4_axi_wdata (m4_axi_wdata),
        .m4_axi_wstrb (m4_axi_wstrb),
        .m4_axi_wlast (m4_axi_wlast),
        .m4_axi_wvalid (m4_axi_wvalid),
        .m4_axi_wready (m4_axi_wready),
        .m4_axi_bid (m4_axi_bid),
        .m4_axi_bresp (m4_axi_bresp),
        .m4_axi_bvalid (m4_axi_bvalid),
        .m4_axi_bready (m4_axi_bready)
    );
```

1.7.11cm3453_capture_monitor 故障留痕

监视器在 RX 域对 6 类故障做寄存并粘滞保存，直到 enable 释放。first_phase 只记录第一次故障阶段，causes 不断按位或累积；错误已经消失也能从诊断字读出历史原因。

诊断字[7:6]为阶段：0 无故障，1 首样点前，2 采集中，3 最后样点后。[5:0]依次保存 not-in-table、disparity、frame_error、sample_valid 丢失、reset_done 丢失和 axis 复位有效。阶段 3 仍可能在 writer 排空时发生，不能因为所有样点已进入 FIFO 就忽略。

故障先进入 faults_q 和 phase_q，再并入历史，输出同时包含待合并故障，因此尽量保留首边沿信息。状态是电平或粘滞信号跨域采样，多个寄存器读取不构成原子快照，定位时应重复读取并结合采集阶段。

故障原因累积及第一次阶段保存：

```
// Capture the raw GT flags once before the sticky feedback logic. The
// pending flags also drive the outputs, preserving first-edge
detection.
assign error = seen || (|faults_q);
assign diagnostic = {seen ? first_phase : (|faults_q) ? phase_q :
2'b00,
                        causes | faults_q};

always @(posedge clk or posedge rst) begin
    if (rst) begin
        seen        <= 0;
        faults_q    <= 0;
        phase_q     <= 0;
        causes      <= 0;
        first_phase <= 0;
    end else if (!enable) begin
        seen        <= 0;
        faults_q    <= 0;
        phase_q     <= 0;
        causes      <= 0;
        first_phase <= 0;
    end else begin
        faults_q <= faults;
        phase_q  <= !started ? 2'd1 : samples_done ? 2'd3 : 2'd2;
        causes   <= causes | faults_q;
        if (|faults_q) begin
            seen <= 1;
            if (!seen) begin
                first_phase <= phase_q;
```

```
            end
        end
    end
end
```

1.8 Block Design 与约束文件

1.8.1 PS 和 PL 的连接关系

表 1-10 BD 各组件作用

组件	接口和作用
processing_system7_0	Cortex-A9、PS DDR、GEM 以太网；GP0 控制，GP1 读取 PL DDR；HP0 至 HP3 接收采集写入
adc_sample_ctrl_0_0	基址 43C10000；寄存器、toggle、状态同步
clk_wiz_0	输入 50MHz，输出 50/100/200MHz；AXI 诊断基址 43C00000
dual_capture	cm3453_dual_capture_bd 模块引用；M0 至 M4 写主机
hp0_sc 至 hp3_sc	独立 SmartConnect 桥接对应 HP 端口
pl_sc	将 MIG 写主机和 PS GP1 读回路径接至 PL DDR
axi_smc	PS GP0 连接控制 IP 与 Clocking Wizard
pl_mig	32 位 DDR3，256 位 AXI，映射 80000000 至 BFFFFFFF
pl_reset / rst_hp125 / rst_ps7_0_100M	分别管理 MIG UI、HP125 及 PS 控制域复位
rst_clk_wiz_0_100M / mig_reset_inv	控制时钟域复位和 MIG 复位极性适配
zero / hp_clk_locked	提供固定输入电平，满足复位模块连接

PS GP1 在此用于 CPU 读 PL DDR。高速 CH2 采集直接由 M4_AXI 进入 pl_sc/MIG；CH1 由 4 个 HP 接口进入 PS。采集写入由自定义 AXI writer 完成，CPU 在采集结束后读取两片 DDR 中的样点并通过 TCP 发送。

MIG 校准信号一方面进入 storage_status，另一方面由软件在启动前检查。32 位 PL DDR 物理引脚和差分系统时钟由 MIG 生成的约束配合管理，不能只复制 HPC1.xdc 后重新建立空工程。

1.8.2 AXI 控制 IP 两层模块

adc_sample_ctrl_0_v1_0.v 是 IP 封装层，声明外部控制与状态接口并实例化

adc_sample_ctrl_0_v1_0_S00_AXI.v。后者实现 8 个 32 位寄存器、AXI-Lite 读写握手、byte strobe 写掩码和状态跨域同步。

写第 5 号寄存器 REG_CONF 时翻转 reg_cmd_toggle，即使连续写入完全相同的命令字也会发起两次事务。前端只允许单个挂起请求，软件必须等完成再写下一条。第 6/7 号读地址返回同步后的 cm_status/cm_readback，不是普通 slv_reg6/7 存储值。

REG_CONF 写入同时翻转请求标志：

```
3'h5:
begin
    for ( byte_index = 0; byte_index <= (C_S_AXI_DATA_WIDTH/8)-1;
byte_index = byte_index+1 )
    if ( S_AXI_WSTRB[byte_index] == 1 ) begin
        // Slave register 5 is the CM3432 command word.
        slv_reg5[(byte_index*8) +: 8] <= S_AXI_WDATA[(byte_index*8) +:
8];
    end
    reg_cmd_toggle <= ~reg_cmd_toggle;
end
```

控制寄存器与实时状态读回：

```
// Address decoding for reading registers
case ( axi_araddr[ADDR_LSB+OPT_MEM_ADDR_BITS:ADDR_LSB] )
3'h0   : reg_data_out <= slv_reg0;
3'h1   : reg_data_out <= slv_reg1;
3'h2   : reg_data_out <= slv_reg2;
3'h3   : reg_data_out <= slv_reg3;
3'h4   : reg_data_out <= slv_reg4;
3'h5   : reg_data_out <= slv_reg5;
3'h6   : reg_data_out <= cm_status_sync;
3'h7   : reg_data_out <= cm_readback_sync;
default : reg_data_out <= 0;
endcase
```

ChannelSel 在 IP 内部取 slv_reg1[2:0]，顶层前端使用 2 位，软件限定 0 至 3。ByteToTrans 仅输出寄存器 3 的低 26 位，是保留接口；当前采集长度由寄存器 4 的完整 Data_Sum 传入双 DDR 模块，不能把 BYTES_Trans 误作本版主长度寄存器。

1.8.3 HPC1 与 CDC 约束

表 1-11 主要板级引脚

信号	FPGA 位置或配置

clk / reset_n_0	F15 / G12；LVCMOS18
PG_C2M_HPC	AA13；LVCMOS25
ad9528_csb / adc_csb	AG22 / AJ24
spi_sclk / spi_sdio / spi_sdio_dir	AH22 / AJ23 / AK22
adc_pd	AH23
adc_refclk_p/n	AF10 / AF9；8.000ns
adc_rxp[0..3]	AE8 / AH10 / AG8 / AJ8
adc_rxn[0..3]	AE7 / AH9 / AG7 / AJ7
adc_sync_p/n	AD23 / AE23；LVDS_25
fan_pwm / led_busy / led_error	C13 / C11 / B11；LVCMOS18

CM3453_HPC1.xdc 明确 MGT 参考连接 HPC1 GBTCLK1_M2C，而不是普通 LA00 ADC_REFCLK。串行 lane 的载板 DP 编号顺序与逻辑 lane 号不同，应按现有引脚映射检查。普通 SPI 控制接口 VADJ 为 2.5V。

1.9 JESD PHY 与 RX IP 配置

1.9.1 JESD204 PHY

图 1-9 JESD204 PHY 关键参数

参数	当前值
Transceiver	GTXE2
Lanes	4
GT Line Rate	10.0 Gbps
GT Reference Clock	100.000 MHz
PLL Selection	QPLL
QPLL FBDIV	100
QPLL REFCLKDIV	1
RX user width	32 bit/lane
RX internal width	40 bit
DRP clock	50 MHz
TX	保持 reset，仅使用 RX

说明：不要在 JESD serial lane 前添加 IDELAYE2。MGT serial 接收的 CDR、串并转换、字节对齐和均衡由 GTXE2 完成，IDELAYE2 适用于普通 SelectIO。

1.9.2 JESD204 RX

图 1-10 JESD204 RX 关键参数

参数	当前值
Node	Receive

Lanes L	4
Lanes L	1
Default K	32
Scrambling	Disabled
SYSREF required	No
Subclass	0
AXI4-Stream data width	128 bit
Line rate / refclk	10.0 Gbps / 100 MHz

1.10SDK 应用程序逐项解析

1.10.1main 与平台初始化

APP/.project 声明处理器为 ps7_cortexa9_0 并依赖 APP_bsp。main.c 先初始化 Clocking Wizard 软件实例，执行平台初始化，配置 MAC 和 IPv4，初始化 lwIP 并添加网卡，最后启动 TCP 服务。APP_USE_DHCP=0，当前默认静态 IP 为 192.168.0.2，掩码 255.255.255.0，网关 192.168.0.1。

主循环轮询 TCP 快慢定时标志、xemacif_input 和各项应用服务。不能在任一服务中长时间阻塞，否则 ACK、重传和命令都无法及时处理。transfer_data 当前直接返回 0，实际数据回传由 Data_Recive_Trans 执行。

主循环应用服务顺序：

```
    /* receive and process packets */
    while (1) {
        if (TcpFastTmrFlag) {
            tcp_fasttmr();
            TcpFastTmrFlag = 0;
        }
        if (TcpSlowTmrFlag) {
            tcp_slowtmr();
            TcpSlowTmrFlag = 0;
        }
        xemacif_input(echo_netif);

        //判断是否有相应指令，并执行对应内容
        Send_ADC_Info();//发送 ADC 信息
        clk_wiz_Config(PLL_Reconfig);//PLL 配置
        CM3453_CommandService();//TCP 下发的 CM3453 寄存器读写/状态回传
        Restart_ADC_Sample(DataNum);//开始采样
        Data_Recive_Trans();//开始 DMA 接收

        transfer_data();
    }
```

表 1-12 平台及板级辅助源文件

源文件	本工程中的作用与启用条件
platform_zynq.c	当前 ARM Zynq 平台；SCU 定时器、GIC、中断及网络定时标志
platform_config.h / platform.h	选择 PLATFORM_ZYNQ；网卡基址与平台函数声明
platform.c	通用模板分支，按处理器宏编译；当前不能当作 Zynq 实现入口
platform_mb.c	MicroBlaze 专用分支，当前不构成本机运行路径
platform_ppc.c	PowerPC 专用分支，当前不构成本机运行路径
platform_zynqmp.c	ZynqMP/AArch64 分支，当前 Zynq-7000 不启用
si5324.c	SGMII 或 1000BASE-X 条件编译时的外部时钟配置
sfp.c	SFP 相关辅助配置；仅匹配主程序宏条件时调用
i2c_access.c	辅助器件的 I²C 访问封装，不承担 AD9528/ADC 三线 SPI
iic_phyreset.c	含板级 I²C PHY 复位支持；main 中特定 ZCU102 宏分支才调用

platform_zynq.c 的定时器回调产生 TcpFastTmrFlag、TcpSlowTmrFlag，并按配置维护链路检测等任务。main.c 在主循环调用 tcp_fasttmr 和 tcp_slowtmr，网络输入由 xemacif_input 持续推进。辅助源码虽然存在于 APP 目录，其有效逻辑仍取决于处理器与板级宏，不能把所有模板文件都描述成上电必执行。

APP/src/README.txt 仍是通用 lwIP echo 例程说明，其中端口 7、192.168.0.10 等文字与本工程有效 main.c、echo.c 不一致；实验连接应使用 192.168.0.2:5000。外层 sdk/src 存在另一份同名源文件，修改和编译前须确认 SDK Project Explorer 所选的是 APP/src。

1.10.2echo 网络连接与 TCP 组帧

echo.c 创建 TCP PCB，绑定 5000 端口并监听。accept_callback 把新连接保存为 serpcb，注册接收和错误回调，关闭旧的空闲连接；采集传输期间拒绝新连接。应用只维护一个有效会话，不支持多客户端同时控制。

recv_callback 遍历 pbuf 链的每个字节，command_stream_byte 先寻找 55 A5，再累积到 8 字节交给 cmd_analysis。这样能处理一个命令被 TCP 拆包以及多个命令被粘包的情况。但组帧不是命令排队：寄存器响应仍使用单个挂起槽，上位机必须发一条、等待一条，不能以支持粘包为由并发下发 SPI 事务。

连接断开时清除组帧位置和部分待处理请求，采集服务随后释放硬件。原始采样流中没有可插入的控制帧位置；传输中收到其他命令会被丢弃，因此状态轮询应在采集开始前或完整接收后进行。

TCP 字节流累积为 8 字节命令：

```
static void command_stream_byte(unsigned char value)
{
    if (command_pos == 0) {
        if (value == 0x55) {
            command_frame[0] = value;
            command_pos = 1;
        }
        return;
    }

    if (command_pos == 1) {
        if (value == 0xA5) {
            command_frame[1] = value;
            command_pos = 2;
        } else if (value != 0x55) {
            command_pos = 0;
        }
        return;
    }
```


1.10.3CMD_Handle 命令服务

cmd_analysis 检查帧头 55 A5 和第 7 字节 F0，然后按 cmd[2]区分采集设置、SPI 读写、状态和信息查询。CMD02 将 cmd[3:6]按大端拼成 32 位总回传字节数；SPI 类命令只使用 cmd[4:5]作为 16 位地址和 cmd[6]作为数据。

CM3453_SubmitCommand 在空闲且 READY 时把 SPI 命令写入 REG_CONF，保存 DONE_TOGGLE 旧值和开始时间。CM3453_CommandService 持续检查完成 toggle，读取 READBACK 低 8 位并排队 8 字节响应。普通寄存器事务超时为 8 秒，返回错误 0x04；单次只能挂起一条。

PING 和状态查询由 PS 直接组织响应；TAP 命令仅回报训练相关状态，返回 tap 值 0，当前串行 JESD 系统没有并行 ADC 输入延迟 tap。CMD05 只允许对齐且范围合规的 Clocking Wizard 寄存器读取，不允许改变共享时钟的分频、相位或占空比。

SPI 命令转换成 PS 寄存器控制字：

```
{
uint32_t command = (((uint32_t)addr & 0x7FFFU) << 8) | data;
```

```
if (op == CM_CMD_READ || op == CM_CMD_CLK_READ) command |= 1U << 24;
if (op == CM_CMD_CLK_WRITE || op == CM_CMD_CLK_READ) command |= 1U <<
25;
if (op == CM_CMD_REINIT) command = 1U << 31;
if (op == CM_CMD_POWERDOWN) command = (1U << 30) | (data & 1U);
cm3453_start_raw(command, op, addr, data, 1);
}
```

1.10.4Restart_ADC_Sample 采集准备

Restart_ADC_Sample 要求 DATA_READY、TRAIN_PASS 及 DDR_CALIB 同时有效，并等待命令响应与设备信息队列清空。默认 ChannelSel=3，DataNum=1048576，Speed_Set=0。双通道回读时 n=Bytes/2，单通道或测试斜坡回读时 n=Bytes；n 始终是每个 DDR 实际保存的字节数。

有效长度要求 Bytes 非零、4 字节对齐、n 为 32 字节整数倍且不超过 0x3E800000。双通道总 Bytes 应为 64 的整数倍，单通道或斜坡为 32 的整数倍。0x3E800000 等于 1000MiB，为 PS DDR 从 0x01800000 到 0x3FFFFFFF 保留的整段；这是代码允许上限，不表示所有板卡和网络测试已经通过最大长度验收。

准备时先写 SAMPLE_EN_GO=0，最多等 10ms 使 writer busy/done/error 等清除，再留 10μs 供 FIFO 复位。PL DDR 映射设为 normal cached 读回，采集前 flush 并关闭 DCache，避免 CPU 脏缓存覆盖新数据。写入每通道长度、通道设置与零抽取参数，读回长度并再等 10μs，最后写 0x101 启动。

给出的采集数据量是两个通道的总数量，每个通道采集的数据是下发指令的一半。

每通道长度校验及抽取限制：

```
n = (ChannelSel == 3U) ? Bytes / 2U : Bytes;
if (ChannelSel > 3U || Bytes == 0 || (Bytes & 3U) ||
    (n & 31U) || n == 0 || n > CAPTURE_MAX_BYTES || serpcb == NULL) {
    cm3453_queue_response(CM_RSP_ERROR, CM_ERR_LENGTH, 0, 32);
    return;
}
if (Speed_Set != 0) {
    cm3453_queue_response(CM_RSP_ERROR, CM_ERR_BANDWIDTH, 0, 0);
    return;
}
/* A new byte count must never reuse the previous frame's enable/DONE.
    * Wait for all five writers and sticky errors to clear, then leave
time
    * for both sides of each asynchronous FIFO to complete reset. */
ADC_SAMPLE_CTRL_0_mWriteReg(SAMPLE_CTRL_ADDR, SAMPLE_EN_GO, 0);
```

```
XTime_GetTime(&rearm_started);
do {
    status = cm3453_status();
    XTime_GetTime(&rearm_now);
    if ((rearm_now - rearm_started) > CAPTURE_REARM_TIMEOUT_TICKS) {
        xil_printf("Capture rearm timeout: status=%08lx\r\n",
(unsigned long)status);
        cm3453_queue_response(CM_RSP_ERROR, CM_ERR_TIMEOUT, 0, 0);
        return;
    }
} while (status & (CM3453_ST_CAPTURE_BUSY | CM3453_ST_CAPTURE_DONE |
                   CM3453_ST_LINK_ERROR | CM3453_ST_OVERFLOW | (0x1FU
<< 8)));
usleep(10);
```

1.10.5Data_Recive_Trans 缓存回读与发送

采集开始后软件轮询状态，任何存储错误、链路故障、校准丢失或 DATA_READY 丢失都会终止本帧。硬件完成等待上限为 5 秒；完成后先清 enable，再开启 DCache 并对两个实际采集地址范围执行 invalidate，保证 CPU 读到 DDR 的新数据。

双通道模式从两片 DDR 同时读取 uint16_t 样点，以 CH1[0]、CH2[0]、CH1[1]、CH2[1]的顺序填入临时数组，再交给 tcp_write。单通道模式直接发送对应 DDR；ChannelSel=0 则生成递增 16 位斜坡供传输格式测试，但采集准备和双 DDR 硬件流程仍会运行。

每次最多排队 1460 字节，同时受 tcp_sndbuf 限制并向下按 4 字节对齐。TCP_WRITE_FLAG_COPY 允许在完整数据复制进 lwIP 队列后释放采集缓冲，函数 finish_acknowledged_capture 的名字并不意味着等待所有远端 ACK。没有发送进展持续 30 秒则超时；ERR_MEM 保持当前游标等待重试，其它 TCP 错误中止。

原始数据尚未排队时，abort_capture 可发送 8 字节错误帧；若已有部分样点进入 TCP，软件会关闭连接而不把错误帧混入样点流。上位机遇到提前断流应丢弃整帧并重建连接。

双 DDR 样点交织和 TCP 数据源选择：

```
    available = tcp_sndbuf(serpcb);
    if (available >= 4 && send_data > 0) {
        chunk = send_data > TCP_CHUNK_BYTES ? TCP_CHUNK_BYTES :
send_data;
        if (chunk > available) chunk = available;
```

```
        chunk &= ~3U;
        if (ChannelSel == 3U) {
            const uint16_t *ch1=(const uint16_t *)(ADC_Data_Addr +
uiIdx/2U);
            const uint16_t *ch2=(const uint16_t *)(PL_DDR_BASE +
uiIdx/2U);
            for (i=0;i<chunk/4U;i++) {
                interleaved[2U*i]=ch1[i];interleaved[2U*i+1U]=ch2[i];
            }
            payload=interleaved;
        } else if (ChannelSel == 0U) {
            /* Preserve the transport-test ramp without changing ADC
sampling. */
            for (i=0;i<chunk/2U;i++)
interleaved[i]=(uint16_t)(uiIdx/2U+i);
            payload=interleaved;
        } else {
            payload=(const void *)((ChannelSel==2U ? PL_DDR_BASE :
ADC_Data_Addr)+uiIdx);
        }
        err=tcp_write(serpcb,payload,chunk,TCP_WRITE_FLAG_COPY |
                      (chunk<send_data ? TCP_WRITE_FLAG_MORE : 0));
        if (err==ERR_OK) {
            send_data-=chunk;uiIdx+=chunk;tcp_output(serpcb);
            XTime_GetTime(&send_progress_time);
        } else if(err!=ERR_MEM) { abort_capture(CM_ERR_TCP,
0);return; }
    }
    finish_acknowledged_capture();
    if (!sample_send_flag) return;
    XTime_GetTime(&now);
    if ((now-send_progress_time) >
((uint64_t)TCP_SEND_TIMEOUT_SECONDS*COUNTS_PER_SECOND))
        abort_capture(CM_ERR_TIMEOUT, 1);
}
```

1.10.6ADC_Info 与链接脚本

ADC_Info.h 定义 packed 设备信息结构，直接把结构内存作为二进制发送；按字段大小共 48 字节。字段包含 ID、16 字节型号、采样率、通道数、补码标记、14位分辨率、预留字段、浮点量程、时钟和周期。当前标称Max_Sample与Clk_Freq 均为 1250000000，Cycle=0.8f，量程字段为±0.7V。这些量程值属于软件元数据，不能代替实际前端增益与输入规格校准。

APP/src/lscript.ld 把程序、堆栈和普通数据放在 0x00100000 开始、长度

0x01700000 的区域，确保 0x01800000 以上保留给 CH1 采集。调整采集基址或最大长度时必须同步检查链接脚本、writer 参数与 CPU 地址，否则可能写坏代码和 lwIP 内存。驱动头文件的 mReadReg/mWriteReg 宏只是 MMIO 封装，不负责采集缓存一致性。

应用内存与采集缓冲区隔离：

```
/* Define Memories in the system */

MEMORY
{
   /* Reserve 0x01800000..0x3FFFFFFF exclusively for CH1 acquisition.
*/
   ps7_ddr_0 : ORIGIN = 0x100000, LENGTH = 0x01700000
   ps7_qspi_linear_0 : ORIGIN = 0xFC000000, LENGTH = 0x1000000
   ps7_ram_0 : ORIGIN = 0x0, LENGTH = 0x30000
   ps7_ram_1 : ORIGIN = 0xFFFF0000, LENGTH = 0xFE00
}
```

设备信息标称参数：

```
static const ADC_info CM3453_info = {
    .ID = 0x00000001,
    .Label = "CM3453",
    .Max_Sample = 1250000000,
    .Channel_Num= 2,
    .Data_Type   = 1,
    .Resolution   = 14,
    .mode = 1,
    .reserved = 0,
    .Max_Volta  = 0.7,
    .Min_Volta  = -0.7,
    .Clk_Freq   = 1250000000,
    .Cycle   = 0.8f,
};
#endif /* SRC_ADC_INFO_H_ */
```

1.10.7驱动和启动支持代码

ip_repo 中的 adc_sample_ctrl_0.c 配合头文件提供自定义 IP 驱动入口；主要访问通过宏完成。adc_sample_ctrl_0_selftest.c 属于模板自检，可能写入寄存器，不能在运行中的采集链路随意调用，尤其 REG_CONF 写入会翻转请求。

SDK 的 fsbl 目录包含 main、image_mover、pcap 以及 SD/QSPI/NAND/NOR 等启动介质支持代码，fsbl_hooks 提供启动钩子，md5/rsa 等文件属于镜像校验或认证支持。它们决定 BOOT 镜像装载，不参与运行中的 TCP 样点交织。JTAG

调试可直接初始化 PS、下载 bit 和 APP.elf；制作 SD 启动镜像时才按 FSBL、bitstream、APP.elf 顺序组合。ps7_init.c/h/tcl 由硬件平台产生，负责 PS 时钟、DDR 和 MIO 初始化，应随硬件导出同步更新。

1.11AXI 寄存器与状态字

表 1-13 控制 IP 寄存器 基址 43C10000

偏移	名称	用途
00	SAMPLE_EN_GO	bit0 采集 enable，bit8 Go；软件写 101 启动、0 释放
04	CH_SEL	0 斜坡；1 CH1；2 CH2；3 双通道
08	SPEED_CTRL	当前必须为 0
0C	BYTES_Trans	保留接口，非本版每通道采集长度入口
10	DATA_SUM	每通道采集字节数 n；32 的整数倍
14	REG_CONF	命令控制字；每次写入触发 toggle
18	CM3453_STATUS	同步状态只读视图
1C	CM3453_READBACK	[7:0] SPI 读值；[23:8]采集调试；[31:24]链路诊断

表 1-14 CM3453_STATUS 全部位域

位	名称或含义	使用方法
31:24	boot_status	80 启动通过；E1 至 E5 失败阶段
23	CAPTURE_DONE	全 5 个 writer 最终 B 响应完成
22	CAPTURE_BUSY	有 writer 持有或排空一帧
21	DDR_CALIB	MIG 校准完成
20	LINK_ERROR	采集监视器粘滞故障
19 / 18 / 17	MMCM 锁定 / QPLL 锁定 / RX 时钟活跃	分别判断控制时钟、GT PLL 与恢复时钟
16	OVERFLOW	采样丢失、FIFO 满或 writer error 的汇总
15	storage_status[7]	与 bit22 重复的 busy
14	storage_status[6]	与 bit23 重复的 done
13	storage_status[5]	与 bit21 重复的校准位
12:8	writer_done[4:0]	MIG、HP3、HP2、HP1、HP0
7	TRAIN_PASS	当前实现与 data_ready 相同

6	TRAIN_DONE	同步后的 rx_reset_done
5	DATA_READY	初始化、PHY 配置和链路检查通过
4	INIT_READY	SPI 启动 init_done
3	CMD_FAILED	本次控制命令失败
2	DONE_TOGGLE	相对提交前变化表示一次命令完成
1	BUSY	SPI 控制或前端命令未完成
0	READY	命令可接收，和 DATA_READY 含义不同

状态查询 CMD17 通过 DATA 低 2 位选择上述 32 位状态中的某个字节：索引 0 读取[7:0]，索引 1 读取[15:8]，索引 2 读取[23:16]，索引 3 读取[31:24]。空闲健康时最低字节常见 F1 或 F5，取决于 DONE_TOGGLE；应检查掩码而非固定比较完整字节。

READBACK[23:8]映射 capture_debug[15:0]：debug[4:0]为 writer_armed，[9:5]为 FIFO 写侧复位忙，[10]为 RX 同步 enable，[11]为 started，[12]为采样数比较结束，[13]/[14]为 HP0/MIG FIFO 空，[15]为 take。原 capture_debug[23:16]的 rx_word 低 8 位被诊断字替换，不再从 READBACK 得到。CMD17 只能读 STATUS，不能用它直接读取 READBACK 高 24 位；这些细节应通过 SDK 调试器 MMIO 或新增诊断命令读取。

1.12TCP 通信协议与常用命令

1.12.1请求和响应格式

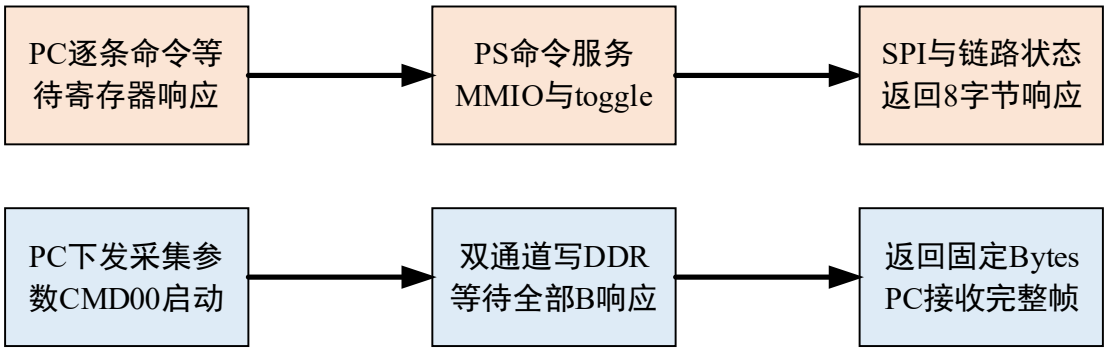


图 1-11 控制事务与原始采集流的两个阶段

表 1-15 8 字节帧格式

字节	寄存器请求	寄存器响应
0 / 1	55 / A5	55 / A5
2	操作码	响应码

3	通常 00；采集设置时属于 32 位参数高字节	状态码，00 成功
4 / 5	地址高/低；或 32 位参数中间字节	返回地址高/低
6	写数据/状态字节索引/参数最低字节	读数据或错误附加信息
7	F0	F0

表 1-16 控制命令及响应

命令	功能	正常响应或行为
00	开始采集	成功直接返回固定长度裸数据，无 8 字节成功头
01 / 02 / 03	通道 / 总回传字节数 / 速度设置	只更新软件参数，无普通 ACK
04	32 位寄存器控制字	写入控制字，不返回普通 ACK
05	Clocking Wizard 诊断	只接受合规寄存器读，写操作被拒绝
10 / 11	ADC 写 / 读	84 / 81
12	训练状态兼容查询	85，状态 bit0 done、bit1 pass，tap=0
13 / 14	AD9528 写 / 读	86 / 87
15 / 16	重新初始化 / ADC 掉电控制	88 / 89
17	读取 STATUS 指定字节	8A，地址字段返回字节索引
1F	PING	80，DATA=5A
FF	设备信息	直接返回 48 字节 packed 结构

CMD00、01、02、03 采用第 3 至 6 字节作为 32 位大端参数，其中 01 实际只用最低字节选择通道。寄存器命令的地址也是高字节在前，但回传 ADC 样点直接来自 ARM 内存，是小端 16 位补码；不能把命令参数端序与样点端序混为一谈。

采集成功没有头、尾、长度字段和校验字段，必须严格接收 CMD02 配置的总 Bytes。若首 8 字节恰好等于合法 E0 错误帧，需结合尚未开始原始数据的协议阶段、长度和状态判断；裸样点本身理论上也可能出现相同字节序列。可靠应用可在未来扩展明确的采集头，但不能假定当前版本已经存在。

1.12.2常用命令速查

表 1-17 可直接输入 HEX 工具的命令

用途	发送字节	响应或者检查
PING	55 A5 1F 00 00 00 00 F0	55 A5 80 00 00 00 5A F0
AD9528 ID 低	55 A5 14 00 00 0C 00 F0	55 A5 87 00 00 0C 56 F0
AD9528 ID 高	55 A5 14 00 00 0D 00 F0	55 A5 87 00 00 0D 04 F0
AD9528 PLL2	55 A5 14 00 05 08 00 F0	返回值与 82 掩码比较
ADC JESD PLL	55 A5 11 00 05 6F 00 F0	返回 DATA bit7 应为 1
正常模拟模式	55 A5 10 00 05 50 00 F0	84 响应，随后等待链路恢复
正满量程测试	55 A5 10 00 05 50 02 F0	预期每样点 1FFF
棋盘测试	55 A5 10 00 05 50 04 F0	预期 1555 和 EAAA 交替，起始相位实测确认
状态字节 0	55 A5 17 00 00 00 00 F0	8A 响应，检查 READY、DATA_READY、TRAIN_PASS
状态字节 2	55 A5 17 00 00 00 02 F0	DATA bit5 为 DDR 校准；bit0/4 为溢出/链路错误
状态字节 3	55 A5 17 00 00 00 03 F0	DATA 为 boot_status，健康 80
重新初始化	55 A5 15 00 00 00 00 F0	88 为当前 reinit 完成响应；异常返回 E0

例如，PC 端要设置采样数据个数(DataNum 寄存器，地址为 2)为 16384 (0x4000)个，发送数据帧内容：0x55 0xA5 0x02 0x00 0x00 0x40 0x00 0xF0。

这里的采样数量是两个通道总共需要采集的字节数量，也就是两个通道各自采集 16384/2=8192 个字节数据，同时，这里是字节数量，程序中是将原始数据 14 位按照 16 位进行缓存传输，

所以这里的 16384 是两个通道总共采集的字节数量，是两个通道各自采集 8192 字节数据，是两个通道各自采集 4096 个数据。

当通道 / 总回传字节数 / 速度寄存器参数设置完成，就可以向 0 号寄存器写入任意值，来开始一次采样传输了。数据帧内容可以为 0x55 0xA5 0x00 0x00

0x00 0x00 0x00 0xF0，这里需要注意的是 0 号寄存器必须放在最后，因为 0 号寄存器负责启动 ADC，ADC 在未配置完全的情况下开始启动，数据很容易输出错误值。

开始传输数据帧命令发送完成之后，ACH3452 就能实现对两个通道进行全速（1250MHz）的采样速率进行采样，共采集 8192 个数据。五个寄存器对应的配置如下表所示：

表 1-18 通用数据帧指令格式表

寄存器名称	数据帧指令
DataNum	55 A5 02 00 00 40 00 F0
ChannelSel	55 A5 01 00 00 00 03 F0
ADC_Speed_Set	55 A5 03 00 00 00 00 F0
RestartReq	55 A5 00 00 00 00 00 F0

配置成网络调试助手发送的数据格式如下：

```
55 A5 01 00 00 00 03 F0
55 A5 02 00 00 40 00 F0
55 A5 03 00 00 00 00 F0
55 A5 00 00 00 00 01 F0
```


1.12.3错误码和连接恢复

表 1-19 E0 响应状态码

错误码	含义	处理
03	操作或受限时钟修改不支持	检查命令和 CMD05 读操作条件
04	命令、采集、重装或发送超时	按阶段检查时钟、写响应或网络
05	尚未就绪	检查 DATA_READY、TRAIN_PASS 和 DDR_CALIB
06	控制命令忙	等待上一条响应后重试
07	控制命令报告训练或初始化失败	读取 boot_status 并检查链路
08	非零 Speed_Set	将 CMD03 设 0
09	采集长度或参数不合法	按每通道 32 字节对齐，检查通道和上限
0A	采集链路故障	读取 READBACK 诊断阶段和原因

0B	存储/FIFO/长度回读错误	检查 HP/MIG、BRESP 和溢出
0C	TCP 失效	关闭旧会话，重新连接

错误 DATA 在 0A/0B 时可能携带 RX 诊断字；没有相应原因时也可能为 0。错误响应只保证在未发送任何裸样点时可返回。若连接在固定字节数接收完成之前关闭，应丢弃已接收内容，再连接、查询状态、恢复并重新采集。

1.13硬件准备与实验平台连接

1. AC709 开发板
2. ACFH3452 模块
3. 信号发生器与适配的 SMA/BNC 连接线
4. Type B 线
5. 千兆网线
6. 按开发板与采集模块要求提供稳定电源
7. 读卡器和 SD 卡（可选）

ACFH3452 接入 AC709 的 FMC HPC1 接口后，应确认方向与机械接触可靠；模块相关跳线按现有硬件配置到 **2.5 V**。模拟输入经模块巴伦进入 ADC，实际测试建议避开巴伦的低频衰减区。

千兆网线连接到 PS 网口处。本次实验通过 SD 卡启动运行程序，将准备好的 BOOT.bin 文件复制到 SD 卡中，接入到开发板的 SD 卡槽中，并且启动拨码开关处于 1 下 2 上（SD 启动模式）。

信号发生器暂时先不接入，不工作。

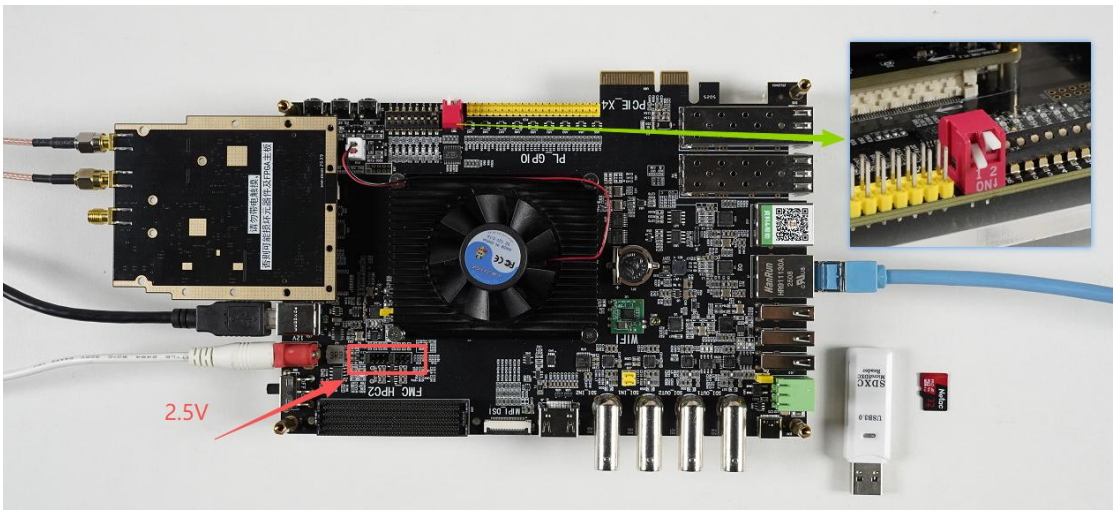


图 1-12 硬件连接

1.14 上电工作流程与最小自检

1.14.1串口打印

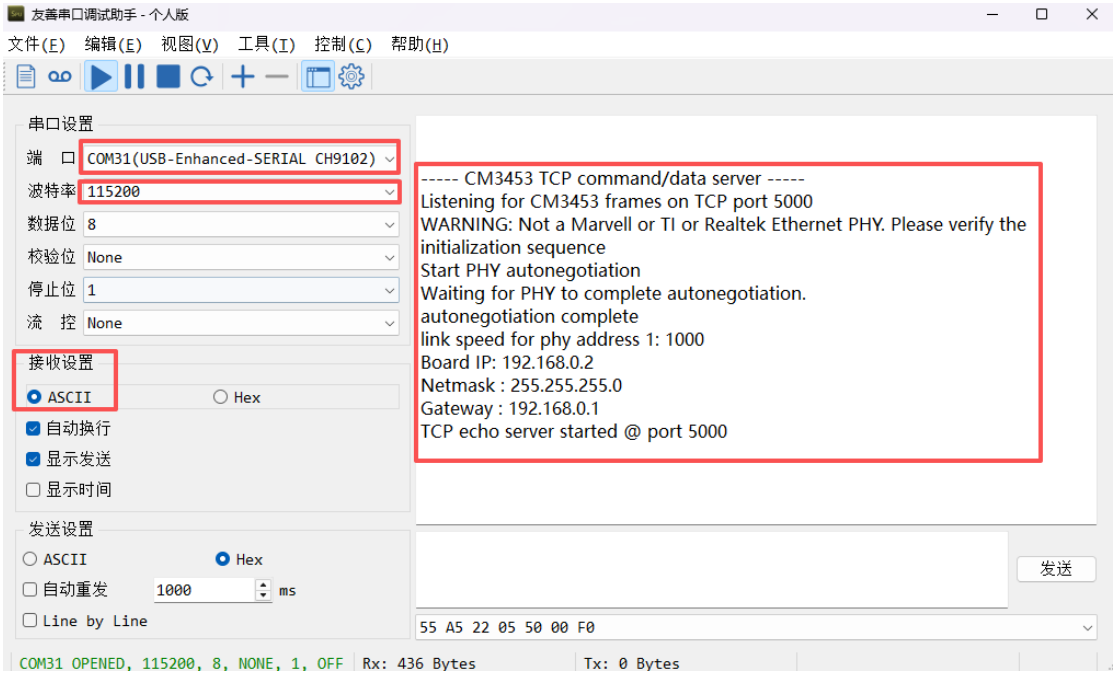


图 1-13 串口信息打印

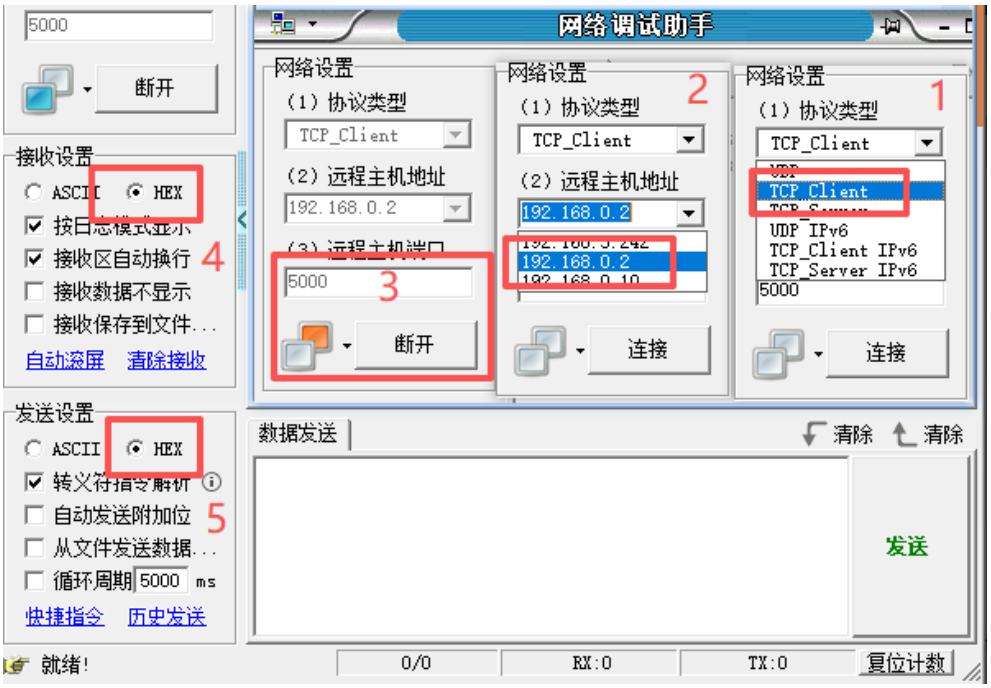
开发板暂时不供电。暂时先不接入信号发生器，模块信号输入端空接状态。

使用串口调试助手，端口选择 PS 的串口，本次实验选择 CH9102，端口号根据实际选择，波特率选择 115200。接收设置选中 ASCII，点击连接。

现在开发板使用 DC 12V 供电。串口开始打印信息。

如上图中右侧打印信息，表示程序运行成功，TCP 连接成功。

1.14.2 网络调试助手调试与测试



打开网络调试助手进行设置：

1. 协议类型选择：TCP_Client；
2. 远程主机地址：192.108.0.2；
3. 远程主机端口：5000；
4. 接收选中 HEX 模式；
5. 发送设置选中 HEX 模式。

这里使用前面表 1-17 可直接输入 HEX 工具的命令中的准备的指令进行测试。这里选择部分指令进行测试。

PING 功能测试：发送“55 A5 1F 00 00 00 00 F0”， 返回值“55 A5 80 00 00 00 5A F0”

进入棋盘测试模式：发送“55 A5 10 00 05 50 04 F0”， 返回值“55 A5 84 00 05 50 04 F0”，往 0x550 寄存器写入 0x04。

回读 0x550 寄存器，确认当前工作模式：发送“55 A5 11 00 05 50 00 F0”，返回值“55 A5 81 00 05 50 04 F0”。 0x550 寄存器值为 0x04。

在棋盘测试模式下，双通道全速采集传输 16384 字节数据。

```
55 A5 01 00 00 00 03 F0
55 A5 02 00 00 40 00 F0
```

```
55 A5 03 00 00 00 00 F0
55 A5 00 00 00 00 01 F0
```

测试结果如下图：



图 1-14 指令测试

棋盘码测试默认应该是 0x1555 和 0x2AAA 交替出现，但是现在回传数据实际为 0x1555 和 0xEAAA 交替出现，着并不是数据错误，而是程序进行 14 位到 16 位符号扩展后的正常结果。

棋盘码原本是 14 位数据，最高位 bit13 被当作符号位。扩展为 16 位时，将这个符号位复制到新增的高两位，所以，按每通道观察，收到 1555、EAAA 交替符合当前程序设计，低 14 位棋盘码没有改变。

```
wire [13:0] frame0_ch0 = frame0_ch0_raw[15:2];
wire [13:0] frame0_ch1 = frame0_ch1_raw[15:2];
wire [13:0] frame1_ch0 = frame1_ch0_raw[15:2];
wire [13:0] frame1_ch1 = frame1_ch1_raw[15:2];
wire [13:0] frame2_ch0 = frame2_ch0_raw[15:2];
wire [13:0] frame2_ch1 = frame2_ch1_raw[15:2];
```

```
wire [13:0] frame3_ch0 = frame3_ch0_raw[15:2];
wire [13:0] frame3_ch1 = frame3_ch1_raw[15:2];

wire [15:0] frame0_ch0_ext = {{2{frame0_ch0[13]}}, frame0_ch0};
wire [15:0] frame0_ch1_ext = {{2{frame0_ch1[13]}}, frame0_ch1};
wire [15:0] frame1_ch0_ext = {{2{frame1_ch0[13]}}, frame1_ch0};
wire [15:0] frame1_ch1_ext = {{2{frame1_ch1[13]}}, frame1_ch1};
wire [15:0] frame2_ch0_ext = {{2{frame2_ch0[13]}}, frame2_ch0};
wire [15:0] frame2_ch1_ext = {{2{frame2_ch1[13]}}, frame2_ch1};
wire [15:0] frame3_ch0_ext = {{2{frame3_ch0[13]}}, frame3_ch0};
wire [15:0] frame3_ch1_ext = {{2{frame3_ch1[13]}}, frame3_ch1};
```

现在处于棋盘测试模式，恢复到正常采集需要退出棋盘测试模式，发送指令如下，往 0x550 寄存器写入 0x00，即回复正常采集模式。

```
55 A5 10 00 05 50 00 F0
```

可以使用回读指令来确认写入结果。

```
55 A5 11 00 05 50 00 F0
```

1.15 功能验证

现在将信号源设置为通道一输出 5MHz 1.0vpp 的正弦波，通道二也输出正弦波，不过是 10MHz 0.5vpp。

接着，通过我们提供的上位机软件“ 小梅哥数据采集仪”采集数据。该软件的最新下载链接如下所示：

小梅哥数据采集系统上位机软件使用说明【基于 QT 开发】

上位机版本不同，UI 界面会有不同，操作基本相似。

1. 通信设置，选择通信方式为以太网 TCP，点击 OK；
2. ADC 设置选择 ACFH3452，如果没有可以自己新建 ADC，点击 OK；
3. 最后点击红色的连接按钮，连接程序会变绿。

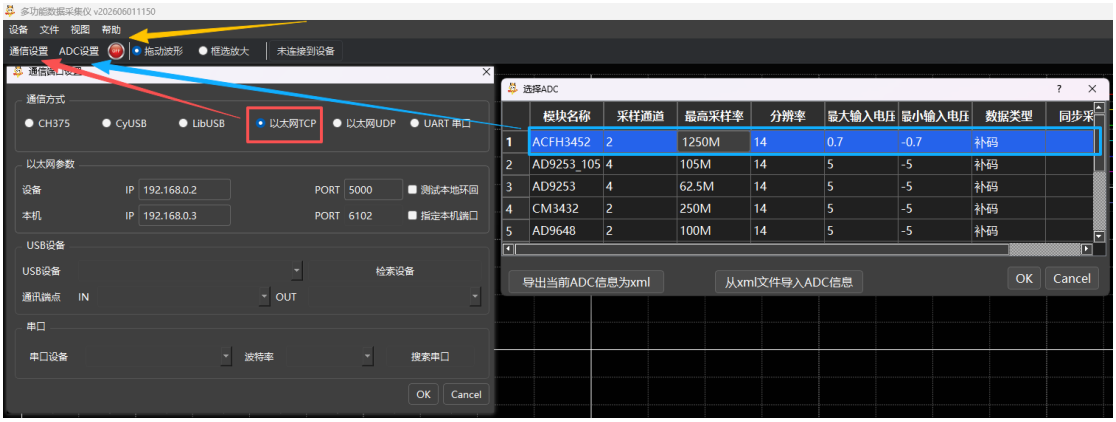


图 1-15 小梅哥 QT 上位机设置

这里采样速率默认为全速，也就是 1250M（显示不全），采样通道默认为双通道，采样数量默认为 4K，用户也可以手动进行更改。

接下来，点击左上角的“点击开始采样”；

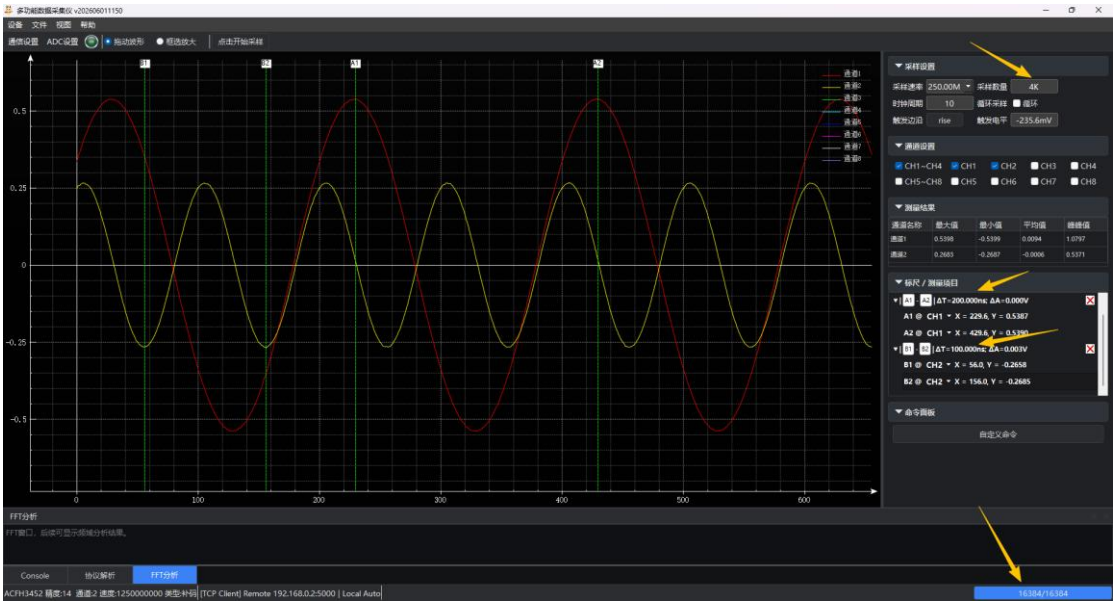


图 1-16 上位机采集和测量

可以看到，两个通道采集绘制波形都是正弦波，两个通道对应的峰峰值分别为 1.0797V 和 0.5371V，属于正常范围内。

同时，默认设置的采样数量为 4K，每个数据大小为 2 字节，因此双通道通道情况下一共需要采集 2*1024*2*4=16384 字节，实际获取到的字节数与该数量一致。

添加光标测量得知，红色通道一的一个周期为 200ns，即 5MHz。黄色通道二的一个周期为 100ns，即 10MHz。符合信号发生器输出设置频率。

综上所述，采集绘制的波形完整，并未出现毛刺现象。

1.16思考与总结

本次实验介绍了基于 ACFH3452 的双通道采集以太网 TCP 收发设计，将两个通道的数据分别缓存到 PS DDR3 和 PL DDR3，再由 Zynq PS 通过千兆以太网 TCP 接口回传至上位机。

实现了对 ADC 的采样控制、数据存储、波形显示。通过这样一套方案，用户可以灵活简便地控制 ADC 以预期的模式运行，并且直观地观察到采样结果。

设计中只是 ACFH3432 以及以太网 TCP 为例进行讲解，实际使用时，用户

也可以基于该架构，根据自身需求，修改 ADC 驱动以及通讯端口，从而应用于更多的场景。

最后，这里再为大家补充两个可能遇到的问题：

1. 无法修改 IP 为静态 IP 地址

出现这种情况，通常都是相关驱动缺少部分文件造成的，此时我们可以使用管理员权限打开 cmd，在其中输入以下代码来修改静态 IP：

```
netsh interface ip set address "以太网名" static 静态 IP 255.255.255.0
```

其中，红色部分为需要用户自己输入的部分，以太网名通常由“以太网”+“空格”+“数字”组成，用户输入时，注意不要忘记空格。

如果想将 IP 恢复到动态 IP 地址，只需要使用以下语句：

```
netsh interface ip set address name="以太网名" source=dhcp
```